Agostino Lorenzi
Andrea Rizzi

Java
PROGRAMMAZIONE AD OGGETTI E APPLICAZIONI ANDROID

INFORMATICA

PER COMPUTER, TABLET E LIM

LIBRO MISTO

Atlas

Agostino Lorenzi
Andrea Rizzi

Java

PROGRAMMAZIONE AD OGGETTI E APPLICAZIONI ANDROID

PER COMPUTER, TABLET E LIM

ATLAS

ISBN 978-88-268-9110-1

Edizioni:
1 2 3 4 5 6 7 8 9 10
2013 2014 2015 2016 2017

Direzione editoriale: Roberto Invernici
Copertina: Vavassori & Vavassori
Videoimpaginazione: Claudio Tognozzi
Disegni: Claudio Tognozzi - Vavassori & Vavassori
Stampa: Vincenzo Bona - Torino

Con la collaborazione della Redazione e dei Consulenti dell'I.I.E.A.

L'Editore si impegna a mantenere invariato il contenuto di questo volume, secondo le norme vigenti.

Il presente volume è conforme alle disposizioni ministeriali in merito alle caratteristiche tecniche e tecnologiche dei libri di testo.

L'Editore dichiara la propria disponibilità a regolarizzare errori di attribuzione o eventuali omissioni sui detentori di diritto di copyright non potuti reperire.

PRESENTAZIONE

UN'OPERA MISTA, MULTIMEDIALE E DIGITALE

Quest'opera propone lo studio e l'applicazione del linguaggio Java secondo le *Linee Guida* per il secondo biennio degli Istituti Tecnici.

È una proposta editoriale **mista**, composta di materiali a stampa, materiali digitali integrativi *on line*, materiali multimediali su supporto ottico; inoltre è disponibile in **forma digitale** (**E-book**) su piattaforma dedicata.

MATERIALI A STAMPA

Il testo rappresenta un'edizione aggiornata che riduce gli aspetti teorici su macchine e sistemi operativi a un solo capitolo di richiami, per sistemare le conoscenze di base provenienti dal biennio, e sviluppa ampiamente la metodologia di programmazione con l'impostazione ad oggetti, limitando all'essenziale la programmazione procedurale.

Il testo comprende anche una parte sulle applicazioni per sistemi Android, per lo stretto legame tra linguaggio Java e sviluppo di applicazioni per l'informatica mobile.

I contenuti seguono puntualmente le *Linee guida* per il secondo biennio negli Istituti Tecnici.

- Richiami sul Prompt dei comandi, perché lo sviluppo dei programmi viene fatto in grande parte da linea comandi
- Linguaggi e programmi
- Sintassi e istruzioni del linguaggio Java
- Ambienti di sviluppo in Java (con guida operativa agli ambienti *NetBeans* ed *Eclipse*)
- Classi e oggetti
- Strutture di dati e file
- Programmazione guidata dagli eventi e interfacce grafiche (con le librerie Java più recenti)
- Applet
- Accesso ai database con JDBC
- Servlet e pagine JSP
- Applicazioni dell'informatica mobile per sistemi Android
- Sicurezza (argomento previsto nelle *Linee guida*).

Le attività proposte sono presentate sotto forma di **progetti** guidati, in modo che siano facilmente realizzate nel laboratorio scolastico o con il computer personale dello studente a casa. I progetti sono sviluppati passo dopo passo con le videate di spiegazione.

Al termine di ogni capitolo si trovano le **domande** di autoverifica e i **problemi** applicativi. Il riferimento alle domande e ai problemi è contenuto all'interno della trattazione dei diversi argomenti.

Per ogni capitolo sono presenti anche le schede **Focus notes** per l'utilizzo del lessico e della terminologia di settore in lingua inglese (come previsto nelle *Linee guida*).

Infine l'**appendice** contiene le **tabelle di riferimento** per il linguaggio Java e Android, una **guida rapida** per i linguaggi HTML e SQL, il **glossario** dei termini dell'informatica, le **soluzioni** alle domande strutturate di autoverifica, e l'**indice analitico**.

E-BOOK PER COMPUTER, TABLET E LIM

L'opera è disponibile anche in **versione digitale E-book per computer, tablet e LIM** sul sito *Scuolabook*: **www.scuolabook.it.**

Tale versione digitale comprende il testo sfogliabile e numerose **espansioni multimediali** quali:

- Test strutturati interattivi
- Lezioni multimediali (videoanimazioni con commento vocale)
- Progetti aggiuntivi di approfondimento.

MATERIALI DIGITALI INTEGRATIVI ON LINE

Le espansioni multimediali sopra indicate insieme ad altri materiali quali:

- Approfondimenti e integrazioni dei contenuti trattati nel testo
- Note operative sull'uso dei supporti software per lo sviluppo delle applicazioni
- Aggiornamenti sui software presentati nel testo

sono disponibili per Studenti e Docenti sulla **Libreria Web**, accessibile tramite un collegamento diretto al sito dell'Atlas: *http://libreriaweb.edatlas.it* oppure con il seguente codice QR per dispositivi mobili.

I riferimenti ai *Materiali on line* sono indicati, in modo puntuale e con numerazione progressiva, al termine dei paragrafi di ogni capitolo, richiamati con un apposito simbolo.

I riferimenti sono inoltre elencati nell'indice generale del testo.

MATERIALI MULTIMEDIALI E DIDATTICI PER L'INSEGNANTE

A disposizione del Docente ci sono innanzitutto i Materiali didattici per l'Insegnante, disponibili nell'*area riservata* del sito della Casa Editrice, a cui i Docenti possono accedere con password, e contemplano:

- note per la compilazione dei Piani di lavoro per i Consigli di classe
- tracce di soluzione ai problemi del testo
- repertorio di esercizi da assegnare come verifiche in classe.

Inoltre i Docenti possono disporre, a richiesta, di materiali multimediali su supporto digitale:

- presentazioni in *PowerPoint* e in *pdf* che illustrano i contenuti dei capitoli e che possono essere utilizzati con la LIM per lezioni multimediali in classe;
- codici sorgente completi dei progetti presentati nel volume;
- ulteriore repertorio di esercizi che possono essere assegnati come autoverifiche agli studenti.

L'Editore

INDICE

Indice dei Materiali *on line*

libreriaweb.edatlas.it

Capitolo 5 - Strutture di dati e file
1. Accesso alle strutture dinamiche usando gli iteratori
2. Struttura di dati dinamica per gestire le *Hash Table*
3. La classe *Stack* e l'interfaccia *Queue* nel package *java.util*
4. Il *Java Collections Framework* per gestire gli insiemi di dati
5. Lettura dei dati da tastiera con la classe *Console*
6. Gestione dei file ad accesso casuale
7. Operazioni sulle directory e sui file

Programmazione guidata dagli eventi e interfaccia grafica
1. I contenitori in AWT
2. La palette delle componenti Swing e AWT
3. Il codice generato automaticamente da NetBeans
4. La componente per gestire un'area di disegno
5. I contenitori *ScrollPane* e *TabbedPane*
6. Evento sulla modifica delle caselle di testo
7. Simulatore di un miscelatore di acqua
8. Rilevazione di dati in un stazione meteorologica

Capitolo 6 - Applet
1. Richiami sugli elementi del linguaggio HTML
2. Chiamata delle funzioni JavaScript da un'applet
3. Grafici statistici
4. Grafico della parabola

Capitolo 7 - Accesso ai database con JDBC
1. Richiami sui database e il linguaggio SQL
2. Dati storici sulle vendite per reparto
3. Forum per Internet

Capitolo 8 - Servlet e pagine JSP
1. Installare e configurare *Apache* Web Server
2. Creare le servlet con l'ambiente di sviluppo NetBeans
3. *Application Server* per sviluppare applicazioni *Java Enterprise*
4. Gestire le sessioni con le servlet
5. Creare i pacchetti per le applicazioni Web (JAR, WAR, EAR)

Capitolo 9 - Applicazioni per l'informatica mobile
1. Versioni del sistema operativo Android
2. Richiami sul linguaggio XML
3. Aggiornare ed estendere l'*Android SDK*
4. Le componenti grafiche di Android
5. Il layout degli oggetti
6. La geolocalizzazione

1

parte prima

Premesse introduttive

Macchine e sistemi operativi

Concetti fondamentali
Rappresentazione delle informazioni
Algebra booleana
Dispositivo automatico
Struttura generale del sistema di elaborazione
Il software
Il sistema operativo
Interprete dei comandi
Il sistema operativo Windows
Multitasking
L'interfaccia standard delle applicazioni
Copie di sicurezza

OBIETTIVI DI APPRENDIMENTO

In questo capitolo potrai rivedere i concetti fondamentali riguardanti l'informatica di base e l'uso operativo di un personal computer, sistematizzando le conoscenze e le abilità acquisite nei precedenti anni di studio.
Potrai quindi avere una visione d'insieme su: sistema di elaborazione e logica di funzionamento, caratteristiche delle risorse hardware e software e funzioni complessive del sistema operativo.

1 Concetti fondamentali

Vengono illustrati di seguito i concetti fondamenti dell'informatica e del funzionamento del computer che saranno usati nei capitoli del testo.

Informazione

In generale si può dire che l'**informazione** è tutto ciò che possiede un significato per l'uomo, e che viene conservato o comunicato in vista di un'utilità pratica, immediata o futura.

L'informazione presuppone da una parte un fenomeno del mondo reale oppure una persona che possiede una conoscenza, e dall'altra una persona che non sa, ma vuole ottenere una maggiore conoscenza.
Le informazioni presuppongono l'esistenza di un insieme di regole comuni all'emittente e al ricevente, chiamate **codice**, secondo le quali le informazioni vengono trasformate per essere comprese dal ricevente.
L'esempio più semplice di codice è costituito dalla lingua naturale (per esempio, la lingua italiana).

La persona A (**emittente**), che vuole comunicare un'informazione, opera una trasformazione (**codifica**) prima di trasmettere alla persona B (**ricevente**), la quale a sua volta deve completare la trasformazione inversa (**decodifica**) per comprendere il significato dell'informazione. Il procedimento viene ripetuto a rovescio quando la persona B diventa l'emittente e la persona A diventa il ricevente.
Questo vale anche quando l'entità B, anziché essere una persona, è costituita da un computer che non è in grado di comprendere il significato delle informazioni nelle forme che noi utilizziamo nei rapporti sociali e interpersonali.

Dati ed elaborazione

L'osservazione del mondo reale ci consente di ottenere gli elementi che utilizziamo per esprimere un'opinione, per caratterizzare un fenomeno oppure per risolvere un problema.
A questi elementi diamo il nome di **dati**: il concetto di dato presuppone la volontà di volere conservare qualcosa nella memoria o negli archivi, in vista di un successivo trattamento per produrre le informazioni, che forniscono una maggiore conoscenza della realtà sulla quale si intendono attivare operazioni di controllo, modifica o direzione.

Quindi c'è una distinzione tra il dato e l'informazione nel senso che:

- il *dato* descrive aspetti elementari di entità o fenomeni
- l'*informazione* è un insieme di dati elaborati e presentati sulla base dell'esigenza di utilizzo pratico da parte delle persone interessate.

Il trattamento dei dati per ottenere le informazioni viene indicato con il termine **elaborazione**.

Il caso più semplice di elaborazione è costituito dalla trascrizione (per esempio, l'impiegato dell'ufficio dell'anagrafe copia i dati di un cittadino sulla tessera elettorale).
Un'elaborazione più complessa è il calcolo matematico, che opera su dati numerici e presuppone la conoscenza di regole predefinite.
Altri tipi di elaborazione ancora più complessi sono: la trasformazione di figure piane, la definizione di uno schema, la costruzione di una teoria scientifica.
Ogni tipo di elaborazione necessita di dati in ingresso (**input**) e produce dati in uscita (**output**).
Il risultato del lavoro di elaborazione è costituito da altri dati, che possono essere a loro volta utilizzati per altre elaborazioni a un livello superiore.

Sistema e modello

Nell'osservazione di fenomeni del mondo reale, specie se complessi, si è soliti utilizzare il concetto di sistema per descriverne il comportamento, in quanto il funzionamento del tutto può essere spiegato solo attraverso il comportamento delle parti.

Un **sistema** viene definito come un insieme anche complesso di elementi (sottosistemi) di natura differente che interagiscono tra loro in modo dinamico e comunque finalizzato al raggiungimento di certi obiettivi.

Ogni sistema è definito quando sono definite:

- le **parti** che lo compongono,
- le **correlazioni** tra esse,
- le **finalità** del sistema.

Le finalità servono per comprendere le ragioni che muovono il sistema, mentre le parti e le relative correlazioni servono per definirne il funzionamento.

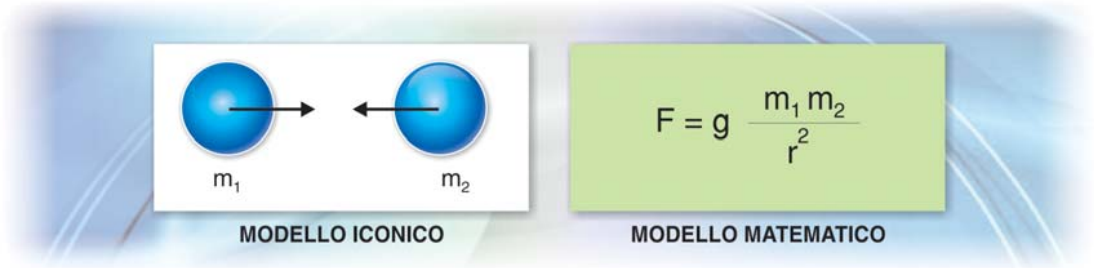
Nella vita comune la visione sistemica di una realtà viene usata in modo molto frequente: sistema nervoso del corpo umano, sistema dei trasporti di una città, sistema politico di una nazione. Per esempio, con il termine Sistema Sanitario Nazionale si indica l'insieme delle strutture, delle attrezzature, delle persone, degli organi di gestione, che forniscono i servizi sanitari nei diversi comuni e località.

Per definire il sistema dobbiamo descrivere le parti (farmacie, reparti ospedalieri, pazienti, medici di base, medici specialisti, ambulatori specialistici, direzione sanitaria, fornitori di materiali), le correlazioni (per esempio, la richiesta di un medico di base attiva l'accesso alla prestazione specialistica di un ambulatorio), le finalità (gestire gli interventi di diagnosi e di terapia, promuovere la prevenzione delle malattie, controllare la produzione e il consumo di farmaci).

Nel linguaggio corrente dell'informatica si usa spesso il termine **sistema di elaborazione**, anziché il termine computer, perché il lavoro con l'elaboratore si avvale non di un solo oggetto, ma di un insieme organizzato di apparecchiature e di programmi che interagiscono fra loro, finalizzati all'elaborazione automatica delle informazioni.

Così come il termine **sistema operativo** indica in informatica l'insieme organizzato dei programmi di base che fanno funzionare l'hardware di un computer, rendendolo disponibile per le applicazioni dell'utente.

L'osservazione del funzionamento di un sistema, oppure l'analisi di un problema da risolvere, viene favorita dalla descrizione mediante un **modello**, cioè dalla rappresentazione semplificata della realtà osservata, che ne evidenzia gli aspetti fondamentali riducendoli in termini trattabili con il lavoro manuale, mentale o automatizzato.



Il tipo di modello che viene utilizzato è determinato da quello che si vuole fare: si va da un livello di astrazione bassa rispetto alla realtà osservata, come nel caso di modelli **iconici**, fino a un livello di astrazione alta come nel caso dei modelli **matematici**.
Un esempio di modello iconico è rappresentato dalla piantina di una città; il piano dei conti della contabilità di un'azienda è un esempio di modello matematico.
Nel lavoro comune di tutte le discipline e di tutte le attività, i modelli vengono sempre utilizzati, perché consentono di concentrare l'attenzione sugli aspetti fondamentali di un sistema o di un problema, trascurando quelli che hanno un'influenza minore sul funzionamento del sistema o sulla possibilità di risoluzione di un problema.

Processo e processore

Quando si deve eseguire un lavoro occorre conoscere l'elenco delle cose da fare e i dati su cui operare.

Con il termine **processo** si indica l'insieme formato dalle operazioni da svolgere in sequenza e dai dati che vengono elaborati durante queste operazioni per svolgere il compito assegnato.

Anche nel linguaggio comune si usa il termine processo per indicare un'attività, con l'idea di mettere in risalto lo svolgersi, il progredire, il compiersi nel tempo: per esempio processo penale, processo produttivo, processo infiammatorio.

Le caratteristiche fondamentali del concetto di processo sono:

- l'evoluzione del processo nel tempo
- la sua sequenzialità.

Il **processore** è l'esecutore del processo, cioè l'ente che causa l'evoluzione del processo nel tempo.

Lo studente, quando svolge una prova scritta in classe, svolge le funzioni di processore rispetto al processo costituito dalla risoluzione degli esercizi assegnati.
Il treno è il processore nel trasporto delle persone da una stazione all'altra, il trasporto è il processo.
Il robot è il processore nel processo di produzione industriale automatizzato.
Il computer, come vedremo in seguito, è un particolare tipo di processore che esegue una sequenza di operazioni per elaborare in modo automatico i dati ad esso forniti.

2 Rappresentazione delle informazioni

Le cifre del sistema di numerazione binario 0 e 1 (**bit**) sono gli elementi fondamentali della scienza dei calcolatori per la rappresentazione delle informazioni all'interno dell'elaboratore.

Solitamente le cifre binarie all'interno di un calcolatore vengono trattate a gruppi o pacchetti contenenti un numero costante di bit: in particolare, per essere elaborate, le cifre binarie vengono raggruppate in sequenze o stringhe di 8 bit. Una stringa di 8 bit prende il nome di **byte**.

Per il trattamento dei dati, gli elaboratori operano su sequenze composte da un numero fisso di byte. Tali stringhe di byte prendono il nome di **parole**.

La rappresentazione dei numeri può essere fatta usando

- la virgola fissa (**fixed point**):

1.5 0.00123 12.564

oppure

- la virgola mobile (**floating point**):

3E–4 12E+18 1.47E–3

(Si noti che per scrivere i numeri decimali viene usato sempre il punto al posto della virgola, secondo la notazione anglosassone).

Il secondo modo di rappresentare i numeri si chiama anche **notazione scientifica** o **rappresentazione esponenziale**.

La lettera E sta al posto di **10 elevato a** ed è seguita dall'esponente a cui elevare la base 10: la potenza di 10 va moltiplicata per il numero (**mantissa**) che precede la lettera E.

Per esempio:

7.2342 E–5

significa

$$7.2342 \times 10^{-5} = 0.000072342$$

Questo tipo di rappresentazione viene usato solitamente in calcoli scientifici, quando si devono trattare numeri molto grandi o molto piccoli scrivendoli in una forma compatta e facilmente leggibile.

La rappresentazione interna dei numeri nella memoria dell'elaboratore subisce una limitazione dovuta alle dimensioni fisiche della cella di memoria, che si traduce in un limite per il numero di cifre significative associate al numero.

Tutto questo viene descritto con il termine **precisione** della rappresentazione interna dei numeri: in genere si chiama **precisione semplice** (o **precisione singola**) quando i numeri reali sono rappresentati, per esempio, con 4 byte (cioè 32 bit) , e **precisione doppia** nel caso di rappresentazione con un numero doppio di byte (per esempio, 8 byte, cioè 64 bit). Il numero di byte utilizzati per la rappresentazione dei numeri può variare per diversi sistemi di elaborazione in commercio.

Tutte le informazioni **non numeriche** (alfabetiche o alfanumeriche) sono esprimibili mediante una combinazione di lettere, cifre o caratteri speciali: affinché un elaboratore riesca a riconoscere e a trattare tali informazioni deve essere stabilita una corrispondenza che ad ogni carattere utilizzato per rappresentare le informazioni associ una particolare configurazione degli 8 bit di un byte.

La rappresentazione dei dati all'interno di un elaboratore è quindi realizzata attraverso l'associazione di una combinazione binaria del byte ad un determinato simbolo (lettera, cifra o carattere speciale): questa associazione è chiamata **codifica**.

La codifica di base per i caratteri di un testo si chiama **ASCII** (*American Standard Code for Information Interchange*, in italiano *codice americano standard per lo scambio di informazioni*) che utilizza 7 bit per codificare un singolo carattere; per esempio a 0110000 corrisponde il carattere "0" (cifra zero) o a 1110001 corrisponde la lettera "q" (minuscola).

I 7 bit del codice consentono di rappresentare 128 (2^7) caratteri diversi tra loro e quindi non sono sufficienti per codificare tutte le variazioni e tutti i simboli utilizzati nelle varie lingue.

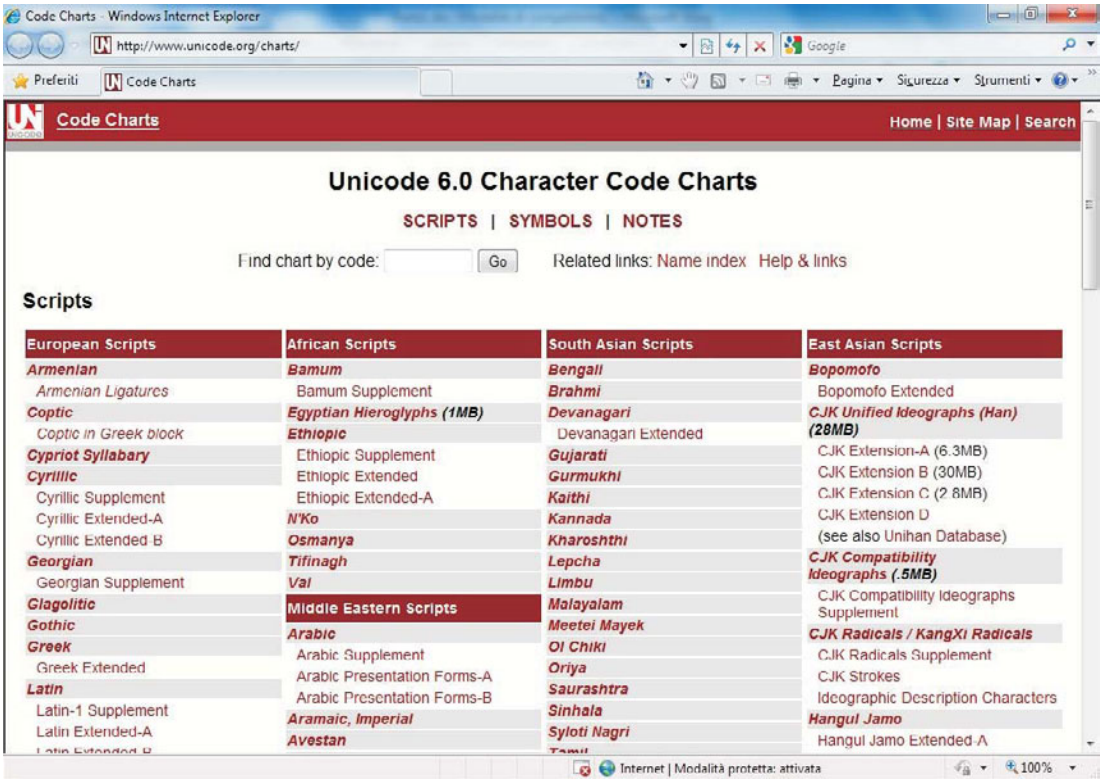
Questo sistema è stato esteso a 8 bit, raddoppiando il numero di caratteri disponibili (256 = 2^8), con la definizione del codice chiamato **ASCII esteso**.

Nel codice ASCII standard ci sono 32 caratteri (i primi 31 e l'ultimo) **non stampabili**, ma che hanno un significato particolare, per esempio *carattere vuoto, fine del testo, segnale acustico, Cancel, Esc, Delete*.

Successivamente si trovano i caratteri stampabili:

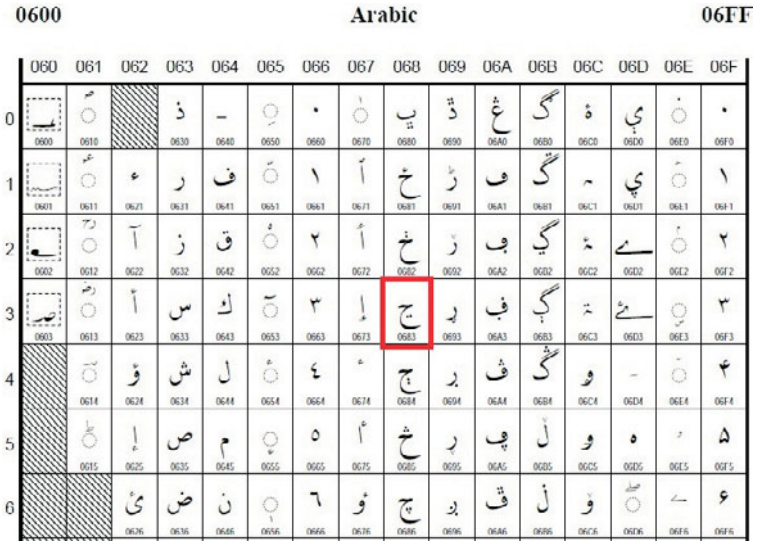
- da **32** (010 0000) a **47**, segni di punteggiatura: punto esclamativo, punto, virgola
- da **48** (011 0000) a **57**, cifre da 0 a 9
- da **58** (011 1010) a **64**, altri segni di punteggiatura: due punti, maggiore, uguale, minore
- da **65** (100 0001) a **90**, lettere maiuscole da "A" a "Z"
- da **91** (101 1011) a **96**, altri segni di punteggiatura: per esempio, apostrofo, parentesi quadre
- da **97** (110 0001) a **122**, lettere minuscole da "a" a "z"
- da **123** (111 1011) a **126**, altri segni di punteggiatura: per esempio, parentesi graffe, tilde.

Nel 1991 è stata sviluppata una nuova codifica, chiamata **Unicode**, che si è diffusa rapidamente ed è in continua evoluzione: è diventata lo standard di fatto per la rappresentazione delle informazioni nei documenti elettronici, in particolare nelle pagine Web, utilizzando i simboli delle numerose lingue esistenti nel mondo.
Le tabelle dei codici *Unicode* sono disponibili sul sito *http://www.unicode.org/charts.*



I primi caratteri di *Unicode* sono esattamente gli stessi della codifica ASCII, in modo da mantenere la compatibilità con il sistema preesistente. All'inizio la codifica utilizzava 2 byte (16 bit, con la possibilità di codificare 65.536 caratteri), ma poi è stata estesa a 32 bit, permettendo la rappresentazione di più di un milione di caratteri differenti. Ovviamente non sono stati codificati subito tutti i caratteri: essi vengono via via assegnati nelle nuove versioni.
L'obiettivo generale di *Unicode* è di creare una codifica che comprenda tutti i caratteri, con tutte le variazioni possibili, di tutte le lingue esistenti, oltre ai simboli utilizzati in matematica e nelle scienze.
Per semplificare le operazioni sono state poi create versioni ridotte del codice che permettono di scrivere i caratteri di uso più frequente in modo più breve: **UTF-8** (a 8 bit), **UTF-16** (a 16 bit) e **UTF-32** (a 32 bit).
Tutte queste codifiche sono compatibili tra loro almeno per quanto riguarda i caratteri standard (lettere maiuscole, minuscole e numeri), mentre può causare problemi con altri simboli (per esempio le lettere accentate o i simboli di frazione).
Riassumendo, ogni carattere viene convertito in bit per essere trasmesso. Più bit vengono utilizzati, più caratteri differenti possono essere utilizzati e più lunga sarà la sequenza di bit da trasmettere.
Per esempio la stringa di testo "*Ciao, mondo!*" contenente 12 caratteri (occorre tenere conto di tutti i simboli, compresi lo spazio e il punto esclamativo) occuperebbe 84 bit (12 x 7) se codificata in *ASCII standard*, mentre ne occuperebbe 384 (12 x 32) in *UTF-32.*

La figura seguente mostra la prima parte del documento *Unicode* per la codifica dei caratteri delle lingue arabe (codice *Arabic*): i caratteri corrispondono ai numeri compresi nell'intervallo da *0600* a *06FF* in esadecimale.



Per esempio il carattere evidenziato con un riquadro rosso corrisponde al codice 0683 in esadecimale (colonna 068, riga 3 della tabella).
Volendo inserire per esempio il carattere in un documento Word, si deve prima convertire in decimale (si può usare la *Calcolatrice* di Windows in *Accessori*):

$$0683_{16} = 1667_{10}$$

Poi, in Word, occorre tenere premuto il tasto *Alt* e premere in successione i tasti *1667* sul tastierino numerico (a destra nella tastiera).
In alternativa, si possono scrivere direttamente nel testo le cifre esadecimali del codice Unicode e premere subito dopo la combinazione di tasti **ALT+X**. Per esempio, scrivendo le cifre 20AC seguite da ALT+X si ottiene il simbolo dell'euro **€**.

MATERIALI ON LINE

1. Codice Unicode

3 Algebra booleana

Nel lavoro di programmazione capita spesso di dover ricorrere ai principi della logica degli enunciati e ai concetti di base dell'algebra delle proposizioni. L'algebra delle proposizioni è detta anche **algebra booleana** dal nome del matematico inglese *George Boole* (1815-1864).

Si dice **enunciato** una proposizione che può essere soltanto vera o falsa.

La verità o la falsità di un enunciato sono dette **valori di verità**; un enunciato può essere vero o falso, ma non entrambe le cose.

Esempi:
- "oggi piove", "quel fiore è rosso" sono enunciati.
- "speriamo che non piova", "dove siete stati?" non sono enunciati in quanto non sono né veri né falsi.

Alcuni enunciati possono essere **composti**, vale a dire sono formati da sottoenunciati collegati tra loro da **connettivi**.

Esempio:
- "egli è intelligente oppure studia tutta la notte" è un enunciato composto dai sottoenunciati "egli è intelligente" e "egli studia tutta la notte" collegati tra loro dal connettivo "oppure".

La proprietà fondamentale di un enunciato composto è che il suo valore di verità è interamente definito dai valori di verità dei suoi sottoenunciati e dal connettivo che li unisce.
Scopo di questo paragrafo è presentare i principali connettivi logici e di illustrarne le proprietà.
Per indicare gli enunciati si usano di solito le lettere **p**, **q**, **r**,... .

Congiunzione (AND)

Due enunciati possono essere collegati dal connettivo "**e**" (in inglese e in informatica, **and**), in modo da formare un enunciato composto, detto **congiunzione** degli enunciati di partenza.
In simboli **p and q** denota la congiunzione degli enunciati e viene letto "p e q".
Il valore di verità di **p and q** è dato dalla seguente tabella (*tabella di verità*) che costituisce la definizione di congiunzione:

p	q	p and q
V	V	V
V	F	F
F	V	F
F	F	F

Dove "V" (vero) e "F" (falso) sono valori di verità.
La prima riga indica in modo sintetico che se *p* è vera e *q* è vera, allora **p and q** è vera. Le altre righe hanno significato analogo. Si osservi che **p and q** è vera solo nel caso in cui sono veri entrambi i sottoenunciati.

Esempio:
- Londra è in Inghilterra e 2+2=4
- Londra è in Inghilterra e 2+2=5
- Londra è in Spagna e 2+2=4
- Londra è in Spagna e 2+2=5

Solo il primo enunciato è vero. Gli altri sono falsi perché almeno uno dei sottoenunciati è falso.

Disgiunzione (OR)

Due enunciati possono essere collegati dal connettivo "**o**" (in inglese e in informatica, **or**), in modo da formare un enunciato composto, detto **disgiunzione** degli enunciati di partenza. In simboli **p or q** denota la disgiunzione degli enunciati e viene letto "p o q". Il valore di verità di **p or q** è dato dalla seguente tabella che costituisce la definizione della disgiunzione:

p	q	p or q
V	V	V
V	F	V
F	V	V
F	F	F

Si osservi che **p or q** è falsa solo nel caso in cui sono falsi entrambi i sottoenunciati.

Esempio:
- Londra è in Inghilterra o 2+2=4
- Londra è in Inghilterra o 2+2=5
- Londra è in Spagna o 2+2=4
- Londra è in Spagna o 2+2=5

Solo l'ultimo enunciato è falso. Gli altri sono veri perché almeno uno dei sottoenunciati è vero.

Disgiunzione esclusiva (XOR)

Due enunciati possono essere collegati dal connettivo "**o esclusivo**" (in inglese e in informatica, **xor**), in modo da formare un enunciato composto, detto **disgiunzione esclusiva** degli enunciati di partenza. In simboli **p xor q** denota la disgiunzione esclusiva degli enunciati e viene letto "p xor q". Il valore di verità di **p xor q** è dato dalla tabella seguente che costituisce la definizione della disgiunzione:

p	q	p xor q
V	V	F
V	F	V
F	V	V
F	F	F

Si osservi che **p xor q** è vera solo nel caso in cui i due enunciati **p**, **q** hanno valori di verità diversi, mentre **p xor q** risulta falsa se i valori di verità di **p**, **q** sono uguali.

Esempio:
- Londra è in Inghilterra o 2+2=4
- Londra è in Inghilterra o 2+2=5
- Londra è in Spagna o 2+2=4
- Londra è in Spagna o 2+2=5

Solo il secondo e terzo enunciato sono veri. Gli altri sono falsi perché i due sottoenunciati hanno valore di verità uguale.

Osservazione
Nella lingua italiana la particella "o" può assumere due significati diversi. In alcuni casi viene utilizzata come "p o q o entrambi" (disgiunzione *or*), in altri casi viene intesa con il significato di "p o q ma non entrambi" (disgiunzione esclusiva *xor*). Per esempio, se si dice: "ha vinto alla lotteria o ha avuto una eredità", si intende che può essere vero "ha vinto alla lotteria" (p) o può esser vero "ha avuto una eredità" (q) o possono essere veri entrambi. Se, invece, si dice: "va a Roma o va a Milano", si esclude che possano essere veri entrambi i sottoenunciati.

Negazione (NOT)

Dato un enunciato **p**, è possibile formare un altro enunciato che si indica con **not p** e che è detto **negazione** di **p**. Nel linguaggio corrente la negazione di **p** si ottiene anteponendo a **p** "non è vero che…" oppure inserendo in **p** la parola "non".
Il valore di verità di **not p** è dato dalla tabella:

p	not p
V	F
F	V

Esempio:

- Londra è in Inghilterra
- Londra non è in Inghilterra
- Non è vero che Londra è in Inghilterra

Il secondo e il terzo enunciato sono entrambi la negazione del primo.

È opportuno osservare che esistono diverse notazioni per indicare i connettivi "e", "o" e "non":

"e"	=	**p et q, p & q, p×q, p∧q, p and q**
"o"	=	**p vel q, p+q, p∨q, p or q**
"o esclusivo"	=	**p aut q, p xor q**
"non"	=	**non p, p', $\bar{p}$, ¬p, not p**

I simboli usati più frequentemente in informatica sono **and, or, xor, not.**

AUTOVERIFICA

Domande da 1 a 4 pag. 39
Problemi da 1 a 3 pag. 40

MATERIALI ON LINE

2. Tavole di verità e proprietà dell'algebra booleana
3. Operatori booleani in Excel
4. Calcolatrice di Windows

4 Dispositivo automatico

Consideriamo un esempio di macchina che esegue un lavoro predeterminato in modo automatico, senza cioè l'intervento manuale di una persona, secondo una sequenza di operazioni che devono essere eseguite in un ordine stabilito e che devono produrre i risultati che ci interessano.
Nel trattare questo esempio non ci interessa come è fatto il dispositivo nella sua realizzazione pratica o come siano i componenti fisici che fanno funzionare la macchina.
Si vuole, invece, mettere in evidenza l'aspetto logico della macchina, cioè le operazioni che compie e quale ordine segue per realizzarle. Inoltre i meccanismi interni che la fanno funzionare vengono schematizzati in modo intuitivo.
Si consideri il funzionamento di una macchina per la distribuzione automatica di carburante.
Dall'esterno la macchina accetta la banconota e la pressione del pulsante del carburante.
Verso l'esterno emette un segnale luminoso, per indicare che la colonnina è pronta, e il carburante.

Il funzionamento del dispositivo è regolato da un procedimento che descrive le operazioni da eseguire per ottenere il carburante in corrispondenza dell'introduzione della banconota:
- accettazione di una banconota
- accettazione della pressione di un pulsante
- erogazione del carburante.

Il procedimento viene ripetuto continuamente per tutto il tempo durante il quale la macchina rimane accesa.
In effetti la macchina deve essere in grado di gestire anche le eccezioni, quali:
- introduzione di banconote non valide
- controllo sul livello di carburante nel serbatoio.

Per poter funzionare il dispositivo deve essere dotato di un meccanismo di introduzione (per l'ingresso delle banconote), di un pulsante per la scelta della colonnina, di meccanismo interno per il controllo delle varie fasi del lavoro e di un'apparecchiatura per l'erogazione del carburante.
In modo molto schematico la macchina può essere descritta così:

Possiamo anche immaginare che la macchina durante il suo funzionamento si trovi in situazioni diverse o **stati** del dispositivo, e che possa cambiare la sua situazione interna passando da uno stato all'altro, in corrispondenza di qualche evento interno o esterno. Più precisamente si può supporre che:
- Quando riceve dall'esterno una banconota, trovandosi in stato di accettazione, emette un segnale luminoso e passa allo stato di erogazione del carburante.
- Quando riceve la pressione del pulsante, trovandosi in stato di erogazione, emette il carburante e torna allo stato di accettazione delle banconote.

La macchina inoltre è dotata di un meccanismo che controlla il contenuto del serbatoio del carburante: in questo stato non accetta più le banconote.
A una macchina di questo tipo possiamo dare il nome di **automa**, intendendo indicare con questo termine un dispositivo che sia in grado di eseguire da solo, cioè in modo automatico, senza l'intervento di una persona, una sequenza di azioni stabilite in precedenza, dotato di meccanismi per acquisire elementi dall'esterno e produrre elementi verso l'esterno: durante il suo funzionamento inoltre, all'interno, può assumere stati diversi fra loro.

Gli aspetti che caratterizzano il funzionamento di un automa di questo tipo sono:
- i simboli forniti dall'esterno, che l'automa sa riconoscere nel loro significato;
- gli elementi prodotti all'esterno come risultato del lavoro svolto;
- l'insieme di tutti i possibili stati che l'automa può assumere;
- l'insieme di tutte le possibili transizioni da uno stato all'altro.

MATERIALI ON LINE

5. Definizione formale di automa

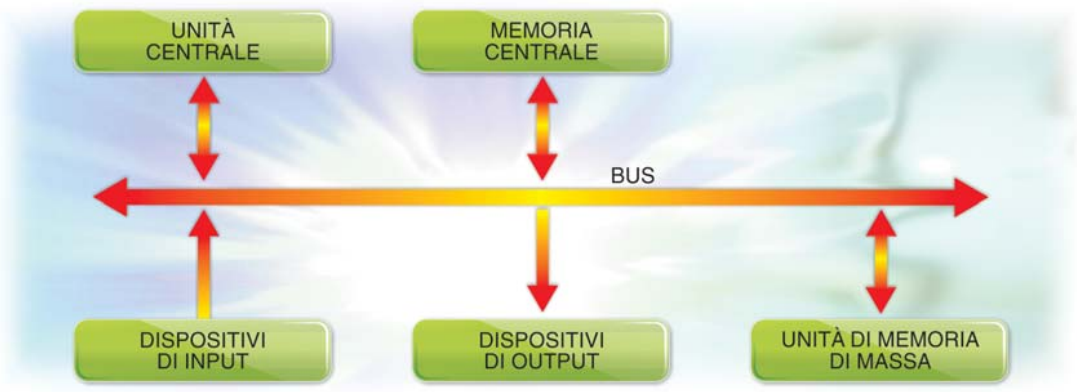
5 Struttura generale del sistema di elaborazione

Verranno esaminate ora le caratteristiche fisiche di una macchina reale per l'elaborazione delle informazioni e le funzioni che vengono attivate per consentire all'utente di utilizzare le risorse a disposizione. Essa deve avere alcune caratteristiche fondamentali:

- può essere utilizzata per problemi di tipo diverso, e non solo per calcoli;
- può eseguire operazioni logiche (per esempio il confronto tra due grandezze);
- può trattare non solo numeri, ma anche caratteri qualsiasi;
- è in grado di comprendere ed eseguire una serie di ordini impartiti dall'esterno, organizzati in un procedimento risolutivo e codificati utilizzando appositi linguaggi di programmazione;
- è in grado di elaborare grandi quantità di dati, con una velocità elevata.

Un'apparecchiatura con queste caratteristiche viene chiamata **computer** o **elaboratore** ed è costituita da un insieme di dispositivi di diversa natura (elettrici, meccanici, ottici), in grado di acquisire dall'esterno dati e algoritmi, e produrre in uscita i risultati dell'elaborazione.

Spesso si usa il termine **sistema di elaborazione**, per evidenziare il fatto che l'elaborazione avviene attraverso un insieme organizzato di risorse diverse. Queste risorse possono essere classificate in due grandi categorie: le risorse **hardware**, che sono la parte fisica del sistema, cioè i dispositivi e le apparecchiature meccaniche, elettriche, elettroniche e ottiche, che si possono riconoscere fisicamente, e le risorse **software**, che sono la parte logica del sistema, cioè i programmi e le procedure che fanno funzionare l'hardware.
Tutte le risorse interagiscono tra loro con la finalità di rendere disponibili le funzioni di elaborazione dei dati all'utente: questo giustifica l'uso del termine *sistema*, secondo la definizione data nei paragrafi precedenti.



Dal punto di vista dell'hardware, la struttura essenziale di un elaboratore comprende:

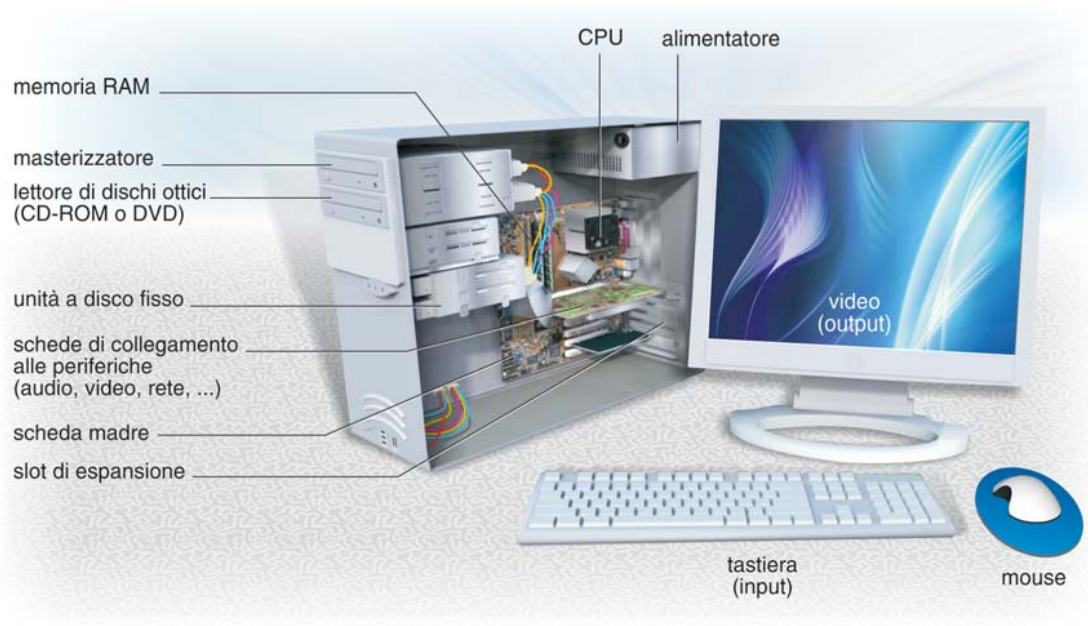
- l'unità centrale di elaborazione (**CPU**, acronimo di *Central Processing Unit,* cioè unità centrale di elaborazione),
- la memoria centrale,
- i dispositivi di input,
- i dispositivi di output,
- le unità di memoria di massa.

Le unità comunicano tra loro seguendo regole codificate, chiamate **protocolli di comunicazione**, usando per i trasferimenti di informazioni uno o più percorsi elettrici chiamati **bus** di sistema.

Questo modello di elaboratore o, come si dice in informatica, questa **architettura**, corrisponde alla **macchina di Von Neumann**, dal nome dello scienziato che nel 1952 elaborò negli Stati Uniti un progetto di computer che costituisce tuttora la base concettuale della maggior parte dei sistemi di elaborazione.

Ogni singolo elemento del sistema è caratterizzato da:
- la capacità di interagire con gli altri elementi e con l'esterno;
- la situazione in cui si trova in ogni istante, cioè il suo *stato*;
- l'evoluzione da uno stato all'altro a seconda dello stimolo ricevuto.

Queste proprietà indicano che i dispositivi sono *automi*, nel senso visto nel paragrafo precedente, e perciò l'insieme formato dal loro assemblaggio è ancora un automa.
Le componenti hardware presenti in un personal computer vengono illustrate con la seguente figura.



I sistemi di elaborazione vengono tradizionalmente classificati secondo diversi livelli, corrispondenti a complessità e potenza di elaborazione crescenti:
1. **calcolatori** tascabili;
2. **piccoli sistemi di elaborazione**: personal computer da scrivania (computer *desktop*), computer portatile (detto anche *laptop* o *notebook*), *tablet* PC, netcomputer o *netbook*, PDA (*Personal Digital Assistant*) o *palmare*, telefono cellulare con funzionalità avanzate (*smartphone*);
3. **medi sistemi di elaborazione**: minicomputer con terminali (tastiera e video) per gli utenti oppure computer collegati in una rete locale;
4. **grandi sistemi di elaborazione**, detti comunemente *mainframe*.

L'unità centrale di elaborazione o **CPU** (*Central Processing Unit*) è il dispositivo che esegue materialmente le operazioni logiche, aritmetiche e di trasferimento sui dati secondo il procedimento richiesto.
Si può considerare come costituita da: una **unità di elaborazione,** in grado di eseguire le istruzioni richieste, e una **unità di governo**, che stabilisce quali operazioni debbano essere eseguite.

La CPU svolge all'interno del sistema di elaborazione le funzioni di processore: i processi sono le diverse attività di elaborazione della CPU.

L'unità di elaborazione contiene elementi di memoria detti **registri**, usati per memorizzare dati e risultati delle operazioni, nonché i codici operativi del linguaggio macchina.
L'unità di governo fornisce all'unità di elaborazione i segnali elettrici che attivano i diversi dispositivi di memoria o di operazione. Questi segnali vengono forniti in sincrono con un orologio interno della macchina, chiamato con il termine inglese **clock**: ad ogni scatto del clock viene inviato un segnale.
La frequenza con cui il clock scatta fornisce perciò un'importante indicazione sulla velocità a cui opera l'unità centrale: la frequenza si misura in Megahertz (**Mhz**) o Gigahertz (**Ghz**), cioè in milioni (Mega) o in miliardi (Giga) di "colpi", o cicli, al secondo.

La **memoria centrale** è un dispositivo elettronico in grado di memorizzare istruzioni e dati codificati in forma binaria. La memoria centrale è pensabile, dal punto di vista logico, come una sequenza finita di locazioni della dimensione di uno o più byte; ogni locazione prende il nome di **parola** (in inglese **word**).
Ogni parola di memoria può essere chiamata anche **cella** di memoria, in quanto è una locazione fisica nella quale è possibile memorizzare uno o più byte.
Per identificare le celle di memoria esse vengono numerate da 0 in poi: ogni cella viene così ad avere un proprio numero identificativo, che viene detto **indirizzo** (in inglese *address*).

I multipli del byte vengono usati per caratterizzare le dimensioni complessive dei dispositivi di memoria (**capacità** della memoria):

1024 byte (2^{10} byte) formano un **Kilobyte**, abbreviato con KB o semplicemente K o Kbyte;

1024 Kbyte formano un **Megabyte**, detto anche MB o Mega o Mbyte;

1024 Mega formano un **Gigabyte** o GB o Giga o Gbyte.

1024 Giga formano un **Terabyte** o TB o Tera o Tbyte.

Tra le memorie di tipo elettronico usate nei calcolatori è importante distinguere tra **ROM** o *Read Only Memory* (memoria di sola lettura) e **RAM** o *Random Access Memory* (memoria ad accesso casuale, nel senso di accesso diretto), sulle quali è possibile anche scrivere.
La memoria RAM è detta anche *memoria utente* perché contiene i programmi e i dati che vengono elaborati dal computer su richiesta dell'utente.
Il funzionamento della RAM è subordinato alla presenza della tensione di alimentazione; in altre parole, se si spegne la macchina tutto il contenuto della RAM viene perduto: si dice che la RAM costituisce una **memoria volatile**.
La struttura delle ROM è invece tale che l'informazione viene ritenuta anche se manca alimentazione (memorie non volatili). Per questo le memorie ROM vengono usate diffusamente in tutti quei casi in cui non serva modificare il contenuto della memoria: tipicamente contengono le istruzioni per la fase detta di **bootstrap**, ovvero di accensione e avvio del sistema. Esse, inoltre, contengono programmi in linguaggio macchina eseguiti spesso per la gestione standard dei dispositivi quali video, tastiera, porte di input/output.

Le **unità di ingresso e uscita** dei dati (*Input* e *Output*) consentono l'acquisizione dall'esterno dei dati che devono essere elaborati dal sistema e la comunicazione verso l'esterno dei risultati dell'elaborazione.
Esempi di unità di input sono: la tastiera, il mouse, il microfono, lo scanner.
Sono unità di output: il video, la stampante, le casse musicali, il plotter.

Le **memorie di massa** o memorie esterne (per distinguerle dalla memoria centrale) sono i supporti che servono per registrare archivi di dati, testi, programmi dell'utente o programmi forniti dalla casa costruttrice del computer. Hanno la caratteristica peculiare di essere memorie permanenti ovvero le informazioni in esse contenute vengono conservate indifferentemente dal fatto che l'unità di elaborazione sia accesa o spenta.
Attualmente le memorie di massa più utilizzate sono costituite da supporti magnetizzabili, oppure da supporti ottici: dischi fissi (*hard disk*), nastri, CD, DVD, chiavette USB, dischi esterni, memorie SD per cellulari e macchine fotografiche.
Sulle memorie di massa, che si chiamano così perché possono contenere quantità rilevanti di dati rispetto alla memoria centrale, vengono fatte *operazioni di input*, quando si leggono i dati registrati sul supporto di memoria e *operazioni di output*, quando si registrano nuovi dati. Quindi possono essere considerate **unità di I/O**.
Tutto quello che può essere registrato su un supporto di memoria di massa si indica con il termine generico di **file** (archivio): un file può essere un testo, un programma, o un archivio di dati.

MATERIALI ON LINE

6. Collegamento delle periferiche

6 Il software

L'insieme delle apparecchiature, dei circuiti, delle memorie, dei cavi, degli interruttori, che formano un sistema di elaborazione si chiama **hardware**, cioè la parte fisica del computer e delle sue periferiche. Ma per poter funzionare un sistema di elaborazione ha bisogno anche del *software***.**

Il **software** è l'insieme dei programmi e delle procedure che servono a finalizzare gli strumenti fisici alla risoluzione del problema presentato dall'utente del sistema.

Con il termine **programma** si indica in generale una sequenza di operazioni che devono essere eseguite dal computer, secondo l'ordine prestabilito. Si usa anche il termine inglese **routine**.

Il termine software comprende:

- il sistema operativo,
- i programmi di utilità e i programmi per la gestione delle periferiche (**driver**),
- i linguaggi di programmazione e i programmi di supporto per il programmatore,
- i programmi applicativi e i programmi scritti dall'utente,
- gli strumenti, indicati con il termine inglese *tools*.

I primi due vengono forniti dalla casa costruttrice del computer e costituiscono parte integrante del sistema di elaborazione. Senza il software infatti un sistema si riduce ad essere solo un insieme di apparecchiature che non producono alcun lavoro, hardware appunto (in inglese *hardware* significa ferramenta). L'insieme dei primi due prodotti software si chiama comunemente anche **software di base** o *software di sistema*. Gli ultimi due tipi di software si indicano con il termine generico di **software applicativo**.

Nel corso del tempo un sistema operativo o un applicativo software necessitano di revisioni per correggere errori, migliorare le prestazioni, aggiungere nuove funzionalità, adattarsi alle nuove caratteristiche fisiche delle piattaforme hardware, fornire modalità di interazione più facili per l'utente.

Questa attività di revisione produce **versioni** successive di un prodotto software indicate con il termine inglese **release** (letteralmente, *rilascio*). Per esempio, nel corso degli ultimi anni il sistema operativo Windows è stato rilasciato nelle versioni Windows 95, Windows 98, Windows ME, Windows 2000, Windows XP, Windows Vista, Windows 7 e Windows 8.
Un programma aplicativo può avere una versione iniziale 1.0 e le successive 1.2, 1.3 o 2.0, 2.1 ecc.: di solito la modifica della parte decimale del numero di release corrisponde a modifiche non sostanziali rispetto alla precedente, mentre la modifica della parte intera del numero di release corrisponde all'aggiunta di nuove funzionalità importanti e migliorative rispetto alla versione precedente.

AUTOVERIFICA
Domande da 5 a 7 pag. 39
Problemi da 4 a 5 pag. 41

7 Il sistema operativo

Il **sistema operativo** è un insieme di programmi che consente all'utente, o alle applicazioni informatiche, di accedere alle operazioni di base per utilizzare le risorse del sistema di elaborazione, risorse sia hardware che software.

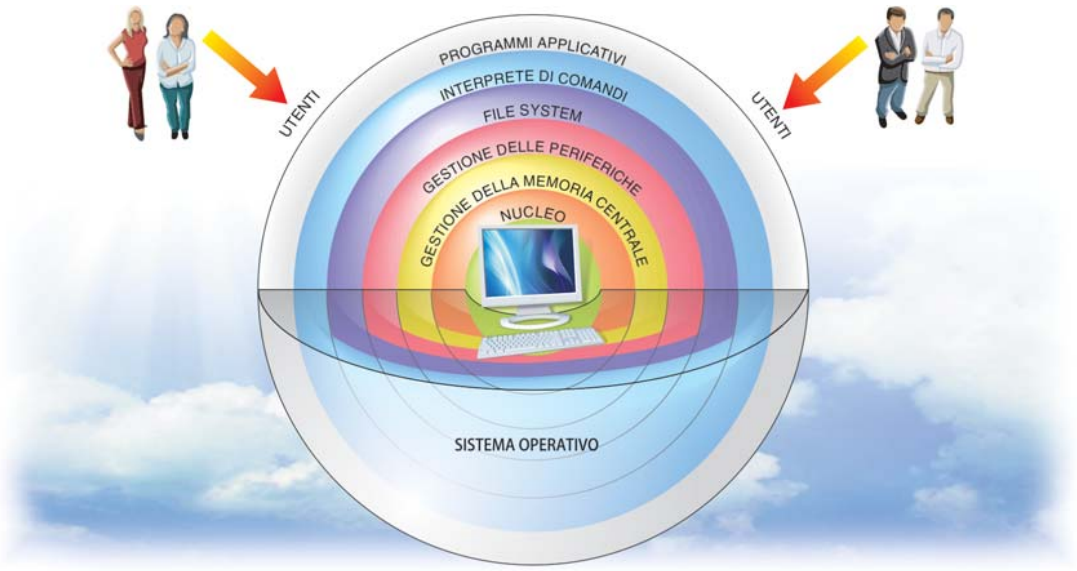
I compiti generali di questi programmi sono:
- offrire un'interfaccia tra gli utenti e la macchina, mettendo a disposizione strumenti di lavoro, facilitazioni, piccole procedure pronte da utilizzare;
- gestire le risorse (hardware e software) al fine di ottimizzarne l'uso da parte degli utenti.

Le principali **funzioni del sistema operativo** possono essere schematizzate secondo questo elenco:
- gestione dell'unità centrale e del processo di elaborazione;
- inizializzazione e terminazione del lavoro della macchina;
- gestione della memoria centrale, nel senso dell'accesso alla memoria e di distribuzione della capacità di memoria in presenza di più utenti e di più lavori;
- gestione dei processi e dell'ordine con il quale vengono eseguiti;
- gestione dell'I/O (*input/output*), cioè l'uso ottimizzato delle periferiche collegate all'unità centrale, anche in presenza di più utenti che usano la stessa risorsa (per esempio una stampante molto costosa condivisa tra più utenti);
- gestione delle informazioni riguardanti i file registrati sulle memorie di massa e dell'accesso ai dati in essi contenuti;
- gestione delle protezioni degli archivi di dati e dei programmi da accessi non autorizzati;
- supporto all'utente-programmatore nella preparazione e nella messa a punto dei programmi.

Gli utenti finali utilizzano le prestazioni del sistema operativo attraverso i programmi applicativi. Gli utenti-programmatori realizzano i programmi applicativi attraverso i linguaggi di programmazione.
Comunemente il sistema operativo viene definito come l'interfaccia tra l'utente e l'elaboratore, nel senso che è costituito da un insieme di comandi e di programmi che consentono di rendere compatibile il software e i lavori dell'utente con l'hardware e le prestazioni della macchina, e nel contempo consente all'utente di attivare le funzioni e i programmi applicativi attraverso un'**interfaccia utente** che può essere di tipo testuale (per esempio nel *Prompt dei comandi*) o di tipo grafico (per esempio in Windows).

Prima dell'avvento dei personal computer ogni casa costruttrice installava sulle macchine sistemi operativi proprietari.
Lo sviluppo recente dei sistemi ha portato a sistemi operativi basati sull'interfaccia grafica, come il sistema **Windows** della Microsoft.
Il sistema operativo **Linux** viene utilizzato su personal computer, ma anche per gestire **server Internet**, ovvero per rendere disponibile l'accesso alle informazioni della rete Internet per gli utenti che lo richiedono. Funzionalità simili per la gestione dei server e delle reti sono disponibili anche nelle versioni *server* del sistema operativo **Windows**.
Altri sistemi operativi sono **Mac OS** per computer *Apple,* **iOS**, **Windows Phone** e **Android** per i *tablet* e i telefoni cellulari che includono funzionalità di *smartphone*.

In generale, un sistema operativo è organizzato a diversi livelli, ognuno con compiti specifici. Ogni livello vede quelli sottostanti come una macchina che svolge determinate funzioni.
Il centro è rappresentato dall'hardware, su cui agisce solamente il nucleo.



Il **nucleo**, chiamato anche **kernel** (nocciolo), è la parte del sistema operativo più vicina alla macchina, ed è strettamente dipendente dall'hardware. I microprogrammi che lo compongono sono chiamati **primitive** del nucleo, e sono scritte in linguaggio macchina, specifico di un determinato microprocessore.
Le funzioni del nucleo sono:
- creazione e terminazione dei processi;
- assegnazione della CPU ai diversi processi;
- sincronizzazione tra i processi;
- sincronizzazione dei processi con l'ambiente esterno.

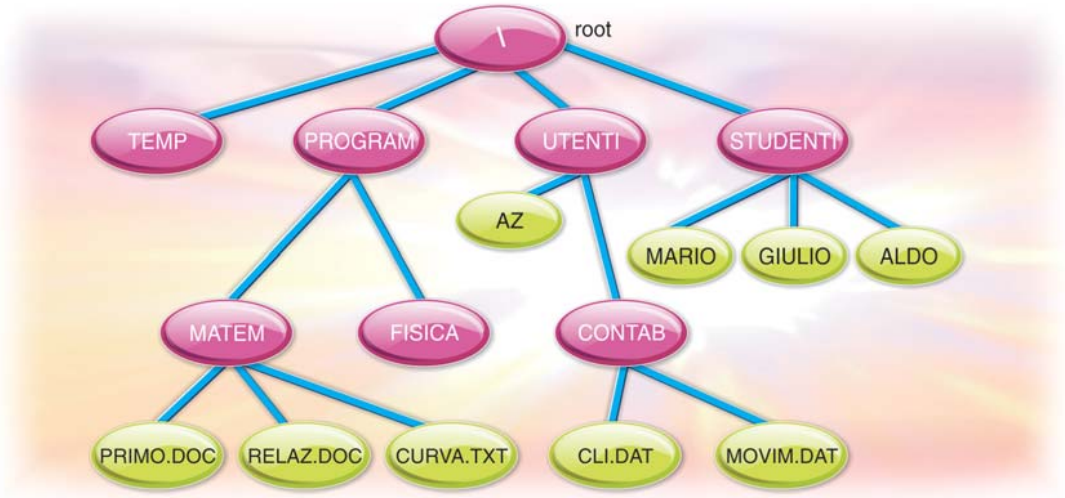
Per la **gestione della memoria**, essa viene divisa in blocchi logici che vengono chiamati *pagine* o *segmenti* a seconda del modo in cui vengono costruiti ed usati.
Una **pagina** di memoria è un blocco di dimensione fissa, che viene assegnato tutto insieme ad un programma che ne faccia richiesta: in altre parole, quando un programma necessita di nuova memoria per la sua prosecuzione, ad esso viene assegnata una pagina intera indipendentemente da quanta ne servirebbe in effetti.
Un **segmento**, viceversa, non ha una dimensione predeterminata, ma variabile a seconda delle richieste del programma: il sistema risponde alle richieste di memoria dei programmi fornendo a ciascuno solo quella necessaria.

Il modulo di **gestione delle periferiche** consente l'utilizzo delle periferiche da parte degli utenti a livello superiore, lasciando al sistema operativo tutti gli aspetti di gestione riguardanti le loro caratteristiche fisiche.
Mette a disposizione le **periferiche virtuali**, gestendo in modo non visibile agli utenti i problemi di utilizzo concorrente, rendendo più efficace e ottimizzato l'uso delle periferiche reali con tecniche quali lo **SPOOL**: l'utente ha quindi l'impressione di avere a disposizione comunque la risorsa richiesta (per esempio una stampante), anche se in realtà questo avviene solo a livello virtuale, in quanto i moduli del sistema operativo assegnano la stessa risorsa a più utenti in tempi diversi.

Il **File System** è il modulo del sistema operativo che gestisce i dati relativamente alla loro organizzazione nelle memorie di massa.
Le sue funzioni principali sono:
- rendere trasparente all'utente l'utilizzo degli archivi su disco, nascondendo i problemi relativi alla memorizzazione fisica e creando una struttura logica;
- consentire l'accesso ai file in lettura e scrittura, risolvendo i problemi di concorrenza che potrebbero verificarsi in regime di multiprogrammazione;
- predisporre funzioni di utilità quali la lista dei file, la loro cancellazione, duplicazione, disposizione nella struttura logica;
- proteggere le informazioni sia per quanto riguarda la loro integrità che la loro riservatezza.

I file sono identificati in modo logico tramite un **nome** che ne ricorda il contenuto e un'**estensione** che ne ricorda il tipo (dati, testo, eseguibile, immagine, ecc.).
Il File System organizza i file nelle memorie di massa con le **directory**.
Quando un disco nuovo viene preparato per essere utilizzato dal computer, attraverso l'operazione detta di **formattazione** del disco, il sistema operativo crea sul disco stesso una directory vuota, detta **root** (*directory radice*).

All'interno della root l'utente poi può creare altre directory, che a loro volta possono contenere altre **sottodirectory** (o *subdirectory*), secondo una struttura organizzata in modo gerarchico detta ad *albero*, nella quale la radice è costituita dalla directory root, i nodi sono le sottodirectory e le foglie sono i file.
Con questa struttura gerarchica è molto comodo accedere a un qualsiasi file contenuto in una sottodirectory diversa dalla directory corrente, indicando il cammino all'interno dell'albero: si costruisce in questo modo un elenco di nomi delle sottodirectory (nel sistema operativo Windows, separati dal segno \ ) che devono essere attraversate per ritrovare il file richiesto.



Per esempio, per accedere al file MOVIM.DAT contenuto nella sottodirectory CONTAB, che a sua volta è contenuta nella directory UTENTI, che è stata creata all'interno della root, si deve indicare come nome del file la seguente scrittura:

\UTENTI\CONTAB\MOVIM.DAT

Questa scrittura prende il nome di **pathname** (nome con indicazione del cammino). Il primo segno di \ indica la root.

MATERIALI ON LINE

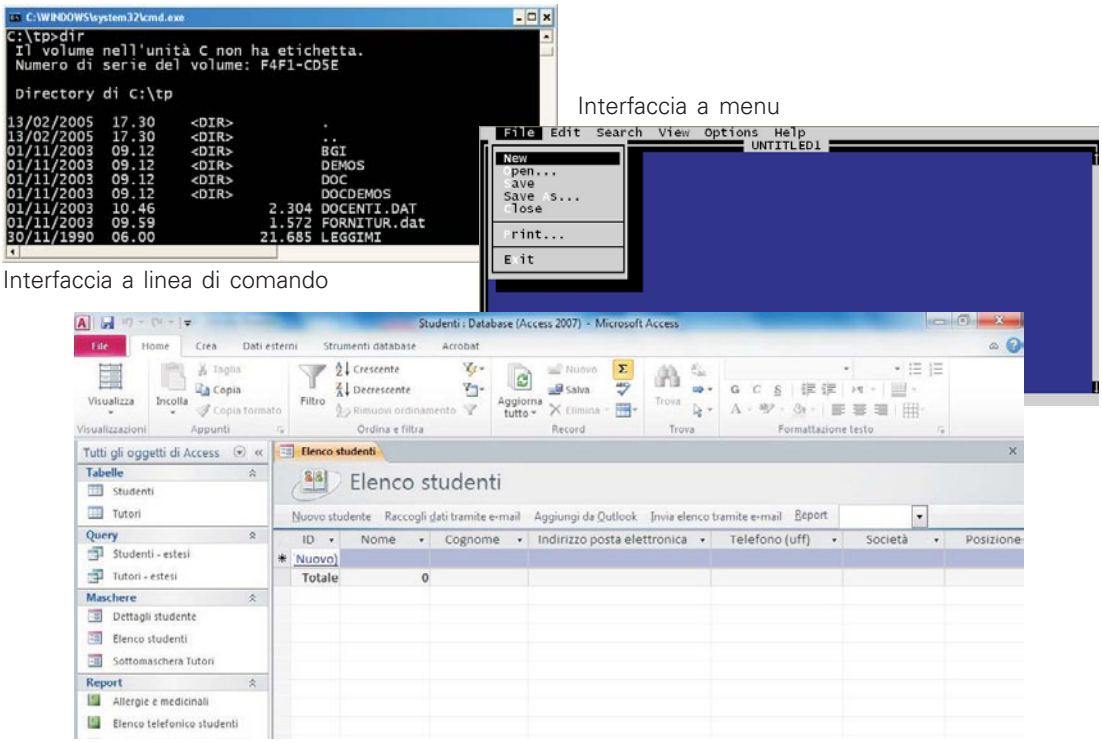
7. La sicurezza

8 Interprete dei comandi

Le prestazioni del sistema operativo vengono attivate dall'utente o dalle applicazioni con modalità differenziate a seconda del grado di sviluppo del sistema operativo stesso, in particolare del livello di **interprete dei comandi**, ossia quel livello direttamente a contatto con l'utente finale. Una possibilità è quella di fornire da tastiera il comando, con i parametri necessari, corrispondente alla funzione richiesta. Il comando viene letto dall'interprete di comandi del sistema operativo, che controlla la correttezza sintattica, e in caso positivo ne attiva la funzione. Se il sistema operativo ha un'interfaccia grafica, le varie funzioni e i vari comandi possono essere attivati attraverso la selezione di menu a tendina o attraverso l'uso di apposite icone.
L'interprete di comandi viene spesso indicato con il termine inglese **shell**.
L'interprete di comandi consente all'utente di specificare le funzioni richieste attraverso un'**interfaccia utente**, che ha subìto negli anni una continua evoluzione per rendere sempre più amichevole l'utilizzo delle risorse di un sistema di elaborazione soprattutto da parte di utenti non specialisti:

- **a linea di comando**: l'utente scrive sulla tastiera il comando e il video svolge la funzione di eco alla battitura; è di questo tipo l'interfaccia del *Prompt dei comandi* di Windows.
- **a menu**: è presente in una parte dello schermo un elenco di comandi, azionabili con la pressione di un tasto, per mezzo dei quali si può accedere a funzioni o a sottomenu.
- **grafica** (*finestre, icone, mouse*): le funzioni di sistema operativo, il software di base, i file, i programmi applicativi vengono rappresentati graficamente sullo schermo nella forma di icone autoesplicative, che vengono selezionate da un puntatore comandato da un mouse. Gli oggetti citati (finestre, icone, mouse) costituiscono gli strumenti di base della cosiddetta **interfaccia grafica** o **GUI** (*Graphical User Interface*), che viene utilizzata soprattutto nei sistemi operativi per personal computer.

I moderni sistemi operativi offrono nuove possibilità per migliorare l'accessibilità del computer. In generale, l'**accessibilità** è la disponibilità di dispositivi e programmi che consentono l'uso del computer e delle sue risorse, oltre che la connessione alle reti e la fruizione dei servizi nel Web, anche per persone con disabilità fisiche (permanenti o temporanee).
Il software per il **riconoscimento vocale** consente agli utenti di inserire informazioni e controllare il computer tramite la voce, quando sono presenti difficoltà nella scrittura o nell'utilizzo delle mani. In questi casi può essere vantaggioso anche l'uso della **tastiera riprodotta sullo schermo**.
I programmi di **screen reader** (lettura dello schermo) producono la lettura vocale del contenuto del video (testo o pagine Web) per persone non vedenti o ipovedenti.
Per le difficoltà visive sono disponibili anche le funzionalità per ingrandire il formato di immagini e caratteri (**zoom**).



Interfaccia a menu

Interfaccia a linea di comando

Interfaccia grafica

MATERIALI ON LINE

8. Traccia riassuntiva su hardware e software

9 Il sistema operativo Windows

Il sistema operativo **Microsoft Windows** si occupa della gestione ottimizzata delle risorse hardware e software di un sistema di elaborazione, consentendo all'utente di riferirsi ad esse con comandi simbolici e rappresentazioni grafiche: il lavoro diventa quindi più semplice e gradevole attraverso un'interfaccia utilizzabile da chiunque intenda operare elaborazioni di dati senza possedere specifiche competenze.

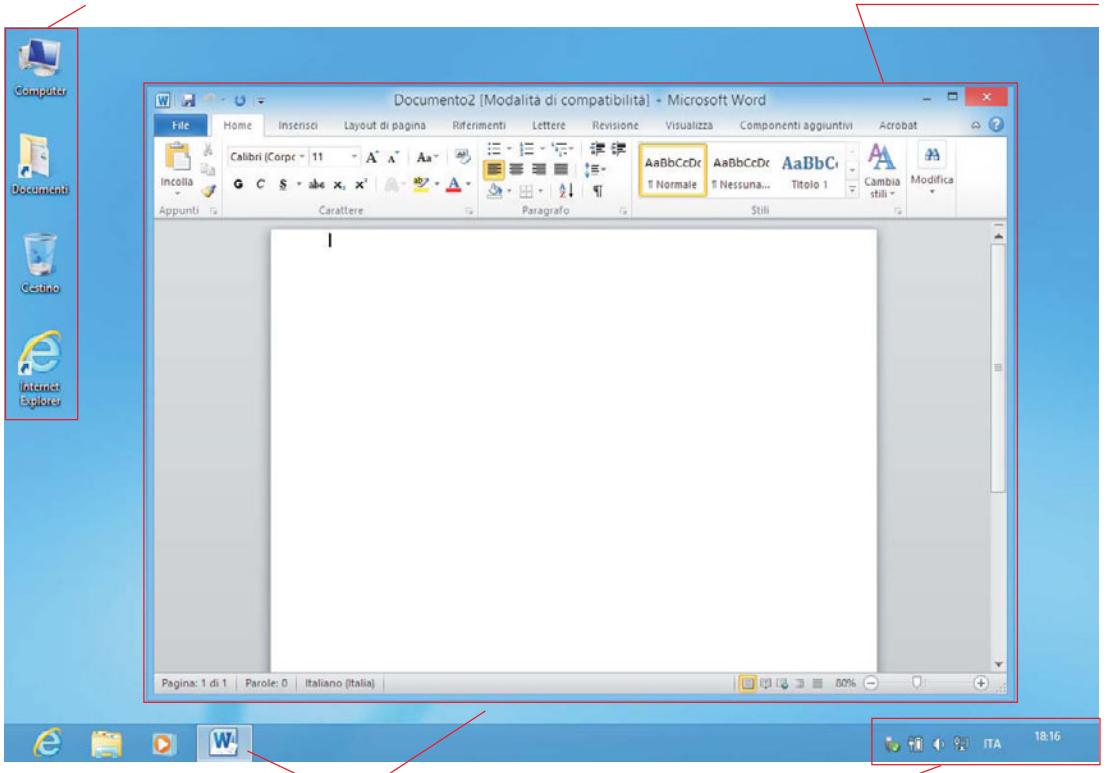
Windows è classificabile come un sistema operativo con un'**interfaccia utente** di tipo grafico (**GUI**, *Graphical User Interface*), a differenza di sistemi meno recenti e più piccoli, quale il DOS, che avevano un'interfaccia utente di tipo testuale. I moduli di Windows includono tutte le funzioni fondamentali di un sistema operativo; vengono inoltre inserite, come programmi applicativi, funzioni aggiuntive destinate a rendere l'utente subito operativo con l'installazione del sistema Windows, e orientate alla gestione delle nuove tecnologie: ci sono moduli per gestire reti locali, Internet, strumenti multimediali. Il tutto è facilitato dall'interfaccia utente, di tipo grafico e di semplice uso, che consente di rendere autonoma sia la gestione delle componenti hardware del sistema (*driver di periferica*), sia la gestione del software e degli archivi di dati presenti sui diversi supporti, dai dischi magnetici ai supporti ottici o altro (*operazioni per la gestione dei dischi*).

Nelle versioni più recenti, poi, il sistema operativo Windows si è integrato in modo completo con l'accesso alla **rete Internet**, fornendo un'unica visualizzazione per le risorse locali e per quelle di rete: le pagine Internet e le risorse presenti sulla rete diventano oggetti trattabili in modo uguale alle risorse del computer dell'utente.

L'ambiente di lavoro di Windows realizza la metafora della scrivania (**desktop**) attraverso l'interfaccia grafica. Il **video** rappresenta il piano di lavoro della scrivania, le **icone** gli oggetti che vengono usati durante il lavoro, le **finestre** i fogli di carta che possono essere messi o tolti dalla scrivania. Ci sono poi altri strumenti che ricordano gli abituali oggetti presenti in un ufficio: per esempio il cestino serve ad eliminare i documenti che non servono più. In pratica le icone rappresentano le applicazioni che possono essere avviate dall'utente; le finestre sono i programmi attivi; in basso la *Barra delle applicazioni* indica i lavori attualmente aperti sulla scrivania. Sul desktop di Windows, presentato sul video, si possono posare tutti gli oggetti disponibili: i programmi, le finestre per interagire con il sistema, i documenti, gli archivi di ogni genere, le risorse del sistema.

Le icone sono piccole immagini che possono rappresentare un programma oppure dei dati. Le principali icone visualizzate di solito sul desktop sono: l'icona della cartella dei **Documenti**, l'icona per gestire il computer (**Computer**), l'icona del **Cestino** (contiene i dati cancellati) e l'icona del programma **Internet Explorer**.

Le **finestre dei programmi** sono aree rettangolari all'interno delle quali vengono eseguiti i programmi.



La **barra delle applicazioni** con i pulsanti dei programmi aperti.

Il **desktop**, o scrivania, è la schermata che compare sul video dopo aver acceso il computer.

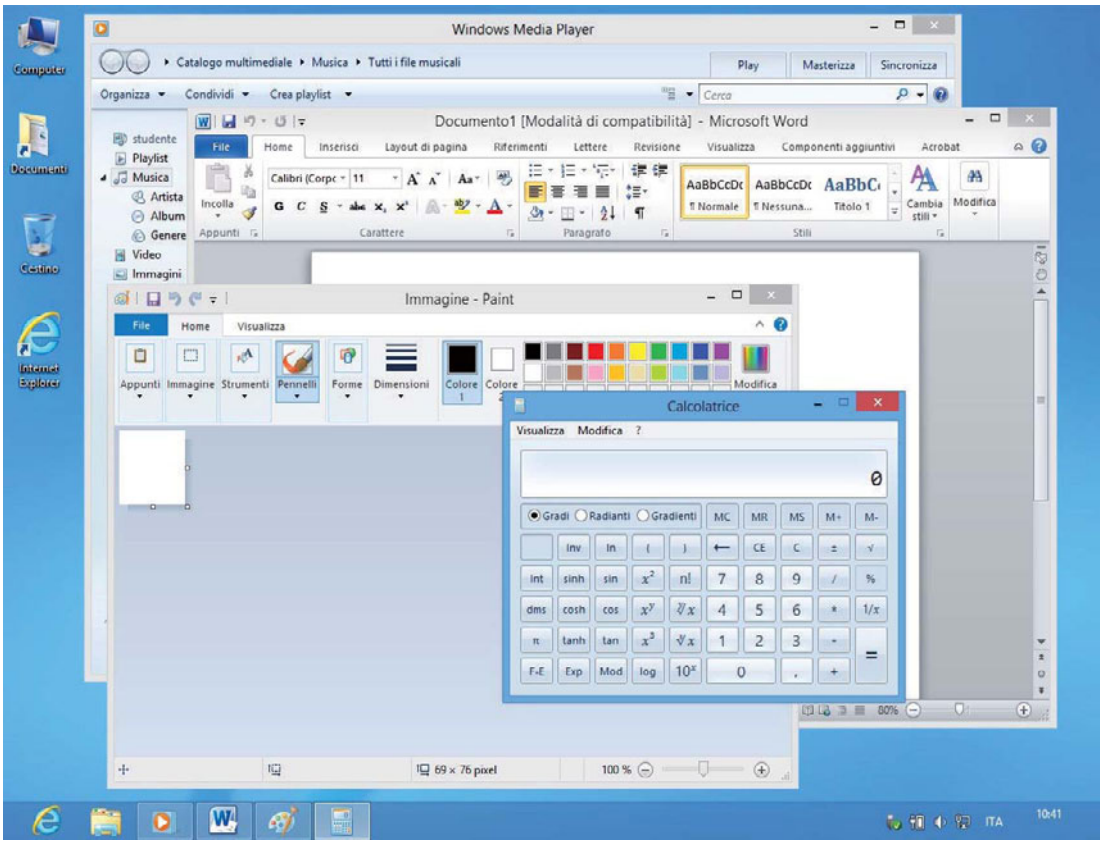
La **tray bar** (o area di notifica) con i programmi di servizio del sistema.

L'aspetto della scrivania è personalizzabile a discrezione dell'utente, il quale può disporre in modo diverso le icone, oppure modificare la risoluzione video, lo sfondo, l'aspetto dei colori e lo **screen saver**, cioè un programma che viene attivato in automatico dopo alcuni minuti di inattività del sistema, per risparmiare l'utilizzo dello schermo e l'energia elettrica che mantiene acceso quest'ultimo.
Tutte le impostazioni di personalizzazione e di configurazione del sistema vengono attivate dal **Pannello di controllo**: scegliere **Impostazioni** nella barra laterale che appare quando ci si sposta con il mouse nell'angolo in alto o in basso a destra dello schermo.

La scorciatoia da tastiera è la combinazione di tasti **logo di Windows + I.**

10 Multitasking

Normalmente nella parte inferiore del desktop è collocata la **Barra delle applicazioni**. A destra è collocata l'**Area di notifica** (*tray bar*) con l'orologio, la regolazione del volume se è presente una scheda audio, l'icona della stampante quando è attivo il gestore delle stampe (*Print Manager*) e il livello della batteria nei sistemi portatili.



La *Barra delle applicazioni* consente all'utente di ricordare i "fogli" che ha posato sulla scrivania, corrispondenti alle finestre, o che ha messo temporaneamente da parte o nascosto sotto altre finestre: in ogni istante è possibile con un colpo d'occhio visualizzare l'elenco delle applicazioni attive e attivare le finestre in quel momento aperte sul desktop.

Per passare da una finestra all'altra

1. Fare clic sul pulsante corrispondente al programma sulla *Barra delle applicazioni*.
2. Se la *Barra delle applicazioni* è nascosta, spostare il puntatore del mouse sull'area dello schermo in cui è posizionata la barra (normalmente nella parte inferiore dello schermo): essa apparirà automaticamente.
   oppure
   tenere premuto il tasto **Alt** e premere ripetutamente il tasto **Tab** (il tasto con le doppie frecce vicino al tasto Q). Alt + ⇄

Si introduce con questo strumento il **multitasking**, una caratteristica molto importante dei moderni sistemi operativi.

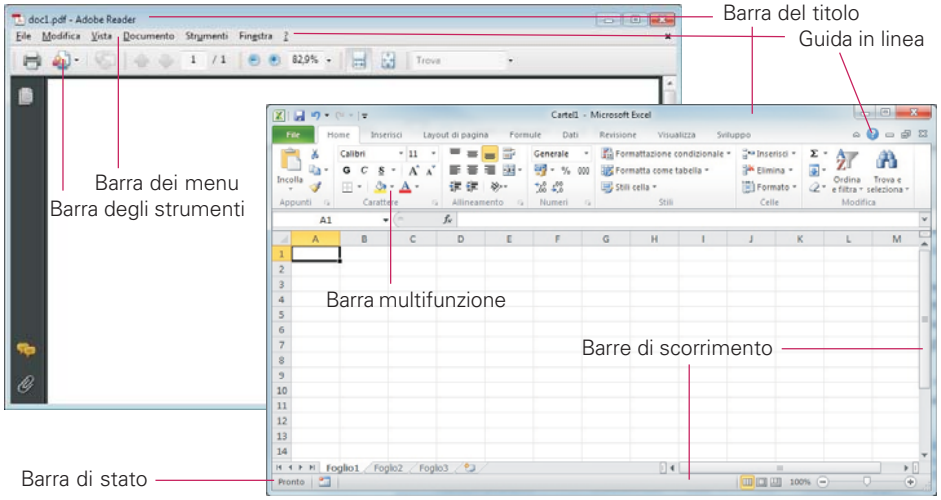
In un sistema monoutente è molto utile far funzionare più applicazioni in contemporanea: per esempio, mentre si scrive un testo, si vuole inserire in esso un grafico creato con un altro programma senza dover terminare il programma di elaborazione dei testi (*word processing*), ma sospendendo temporaneamente l'applicazione, oppure si vuole ascoltare un motivo di sottofondo riproducendo le canzoni contenute in un CD musicale.
Un processo attivo prende il nome di **task** (*attività*); un programma può attivare uno o più processi per una o più funzioni; la possibilità di far funzionare questi processi in contemporanea corrisponde al multitasking.
Occorre far notare che per realizzare il multitasking è necessario disporre anche di risorse hardware molto efficienti e in passato costose (in particolare di una quantità rilevante di memoria RAM): ecco perché i primi personal computer non potevano appesantire il processore (CPU) con una gestione multitasking.
Per ogni programma attivo viene creato sulla *Barra delle applicazioni*, nella parte inferiore dello schermo, un pulsante che ne riporta l'icona, il nome ed eventuali documenti aperti in quel programma. Per passare da un programma all'altro, basta spostarsi con il mouse sul pulsante e fare clic con il tasto sinistro. La *Barra delle applicazioni* diviene così lo strumento principale tramite il quale si attiva il multitasking, mantenendo nel contempo in evidenza le applicazioni aperte sul desktop.

Per ridurre a icona tutte le finestre aperte

Fare clic sul pulsante **Mostra desktop** in fondo a destra nella *Barra delle applicazioni* (vicino alla data e ora). Tutte le finestre aperte diventano pulsanti sulla *Barra delle applicazioni*. Per ripristinare lo stato precedente delle finestre, fare nuovamente clic sul pulsante *Mostra desktop*.
In alternativa, per ridurre a icona tutte le finestre aperte, si può usare la scorciatoia da tastiera: tenere premuto il tasto **logo di Windows** (il tasto vicino al tasto *Alt*) mentre si preme il tasto **M**. Per visualizzare velocemente il desktop si può usare anche la combinazione di tasti **logo di Windows** + tasto **D**.

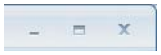
11 L'interfaccia standard delle applicazioni

I programmi e le attività dell'utente vengono rappresentate con riquadri sullo schermo che prendono il nome di **finestre**: esse hanno caratteristiche standard che si presentano simili per i diversi programmi applicativi nell'ambiente Windows.



Gli elementi fondamentali sono:

- La **Barra del titolo** in alto.
- La **Barra dei menu**, che vengono aperti dall'alto verso il basso (menu a tendina) facendo clic su una delle voci della barra.
- La **Barra degli strumenti** oppure la **Barra multifunzione**: è l'insieme delle icone che rappresentano i pulsanti dei comandi posti sotto la barra del titolo. Nella *Barra multifunzione* i pulsanti sono organizzati in *schede*; ogni scheda comprende diversi *gruppi*, ciascuno dei quali include comandi correlati tra loro; un *comando* è un pulsante, una casella per l'immissione di informazioni o un menu.
- Il punto interrogativo o il pulsante per attivare la **Guida in linea** (*help*), che fornisce un aiuto all'utente durante l'esecuzione del programma.
- Le **Barre di scorrimento**, attivate automaticamente in senso verticale e orizzontale quando le dimensioni del documento eccedono le dimensioni della finestra o dello schermo.
- La **Barra di stato** posta nella parte bassa della finestra, riporta informazioni o messaggi relativi al contenuto o alle operazioni della finestra.

Ci sono poi in alto a destra i tre pulsanti standard di ogni finestra: 

- il primo riduce l'applicazione a un'icona sulla *Barra delle applicazioni* (**Riduci a icona**);
- il secondo ingrandisce la finestra a tutto schermo (**Ingrandisci**), oppure la riduce a dimensioni più piccole se la finestra è già aperta a tutto schermo (**Ripristina giù**);
- il terzo chiude l'applicazione arrestandone l'esecuzione (**Chiudi**).

Per *ingrandire* la finestra o *ripristinarne* le dimensioni originali, è inoltre possibile fare doppio clic sulla relativa barra del titolo.

Per muovere una finestra

1. Ridurre la finestra a dimensioni più piccole se la finestra è aperta a tutto schermo (con il pulsante **Ripristina giù**).
2. Tenere il tasto sinistro del mouse premuto sulla barra del titolo della finestra e trascinarla sul desktop fino alla posizione desiderata.

Per ridimensionare una finestra

1. Per **modificare la larghezza** della finestra, posizionare il puntatore del mouse sul bordo sinistro o destro della finestra. Quando il puntatore assume la forma di una freccia orizzontale a due punte, trascinare il bordo verso destra o verso sinistra.
2. Per **modificare l'altezza** della finestra, posizionare il puntatore del mouse sul bordo superiore o inferiore della finestra. Quando il puntatore assume la forma di una freccia verticale a due punte, trascinare il bordo verso l'alto o verso il basso.
3. Per **modificare contemporaneamente l'altezza e la larghezza** della finestra, posizionare il puntatore del mouse su un angolo della finestra. Quando il puntatore assume la forma di una freccia diagonale a due punte, trascinare l'angolo nella direzione desiderata.

Per chiudere una finestra

fare clic sul terzo pulsante x nell'angolo in alto a destra della finestra.
Per chiudere una finestra dalla *Barra delle applicazioni*, fare clic con il tasto destro del mouse sul pulsante nella *Barra delle applicazioni* e quindi scegliere **Chiudi finestra**.
Per la chiusura di una finestra si può anche usare la scorciatoia con la combinazione dei tasti **Alt+F4**. Alt + F4

12 Copie di sicurezza

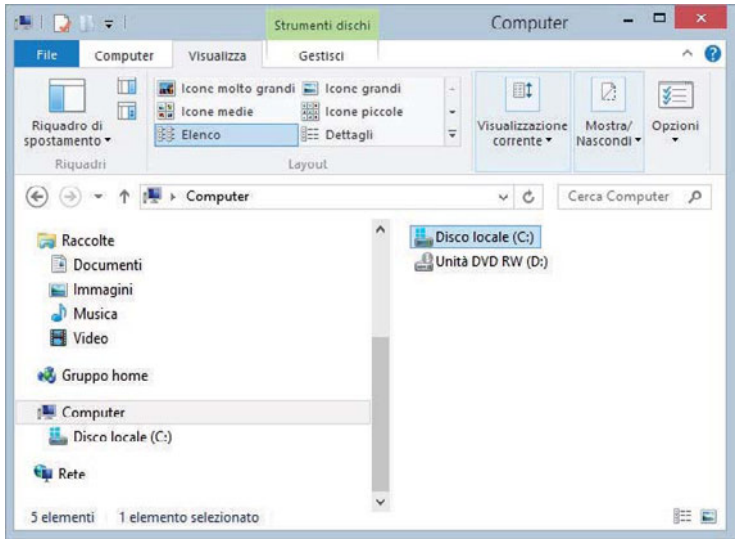
Il termine **backup** indica le copie di sicurezza dei file. Il *backup* è importante perché si può verificare la perdita di file a causa della relativa eliminazione o sostituzione accidentale, dell'attacco di virus, di un errore hardware o software oppure di un errore dell'intero disco rigido.
L'operazione di backup deve essere un'attività svolta con regolarità, anche ogni giorno, soprattutto quando riguarda lavori e dati importanti o che comunque richiederebbero tempi lunghi per il recupero.
L'operazione a rovescio, cioè il recupero di dati o documenti dai supporti contenenti le copie di sicurezza, si chiama **ripristino**, in inglese *restore*.
Le copie di *backup* vanno effettuate su supporti diversi da quelli contenenti i file originali, perciò si usano unità di memoria di massa rimovibili, quali chiavi USB, CD, DVD, hard disk esterni o nastri magnetici.
Si possono copiare singoli file o cartelle importanti per il proprio lavoro oppure si può avviare la procedura di backup: dal **Pannello di controllo, Sistema e sicurezza** e poi **Salva copie di backup**.
È buona norma conservare poi i supporti di backup in luoghi separati da quello di lavoro, per garantire una conservazione più sicura e una protezione da furti o danneggiamenti accidentali.

LAVORARE IN RETE

Se il computer appartiene ad una rete locale, il sistema operativo Windows, alla partenza del sistema, attiva la procedura di identificazione dell'utente di rete, con la richiesta del nome di utente e dell'eventuale password. In ogni istante l'utente, connesso alla rete, riesce a vedere quali siano le risorse condivise nella rete.

Per visualizzare i computer e i dispositivi di rete

Sul desktop fare clic sull'icona **Computer** e selezionare nel riquadro di spostamento a sinistra la voce **Rete** oppure visualizzare la barra laterale **Accessi** e scegliere **Ricerca**: nella casella **Cerca** scrivere **Rete**; fare infine clic sul pulsante che viene visualizzato.

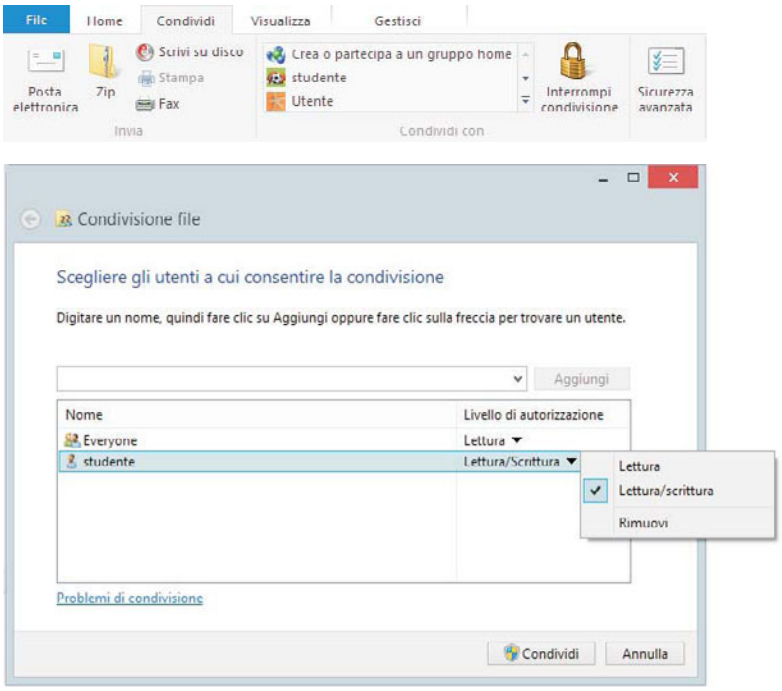


Si possono ottenere informazioni sulla rete anche dal **Pannello di controllo**, scegliendo **Rete e Internet**.

Ogni computer collegato alla rete possiede un nome che lo identifica; facendo doppio clic sull'icona di uno tra i computer elencati, è possibile visualizzare le risorse che il computer ha messo in condivisione (cartella o file).

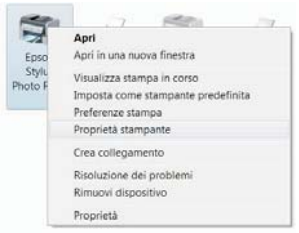
In generale, la risorsa di rete individuata nell'elenco degli oggetti condivisi può essere aperta e utilizzata come se fosse residente sul proprio computer locale.
Il lavoro in rete si basa sulla collaborazione tra gli utenti: quindi, oltre ad utilizzare le risorse di rete, l'utente stesso può mettere in condivisione proprie risorse per gli altri utenti.

Per mettere in condivisione una cartella
Fare clic sulla scheda **Condividi** nella parte superiore della finestra e scegliere l'utente o gli utenti che possono condividere la cartella.
Oppure fare clic con il tasto destro del mouse sull'icona della cartella che si vuole mettere in condivisione; scegliere **Condividi con** dal menu che si apre.

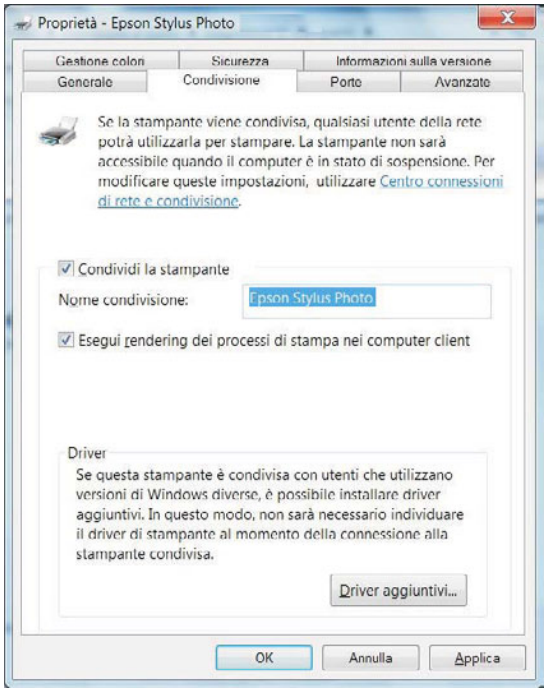


Per condividere con tutti, scegliere **Utenti specifici** e, nella finestra che si apre, nella casella in alto scrivere **Everyone** e poi fare clic su **Aggiungi**. Nella seconda colonna impostare il livello di autorizzazione: *Lettura*, *Lettura/Scrittura*. Fare infine clic sul pulsante **Condividi**.
Quando il computer è collegato ad altri computer in una rete locale, l'utente che non ha a disposizione una propria stampante può utilizzare quella di rete che viene condivisa con altri utenti; in altri casi può egli stesso mettere a disposizione una stampante, connessa al computer in uso, per tutti gli altri utenti della rete.

Per mettere in condivisione una stampante
Nel **Pannello di controllo**, scegliere **Hardware e suoni** e poi **Dispositivi e stampanti**.
Fare clic con il tasto destro del mouse sull'icona della stampante da condividere.
Scegliere **Proprietà stampante** dal menu di scelta rapida.



Nella finestra che si apre fare clic sulla scheda **Condivisione**. Mettere il segno di spunta all'opzione **Condividi la stampante**, assegnando anche un nome logico con il quale la stampante può essere identificata come risorsa di rete. Confermare con **OK**.



L'icona della risorsa viene modificata aggiungendo la figura di due persone. 

Per utilizzare una stampante di rete condivisa

1. Nel **Pannello di controllo**, scegliere **Hardware e suoni** e poi **Dispositivi e stampanti**: fare clic sul pulsante **Aggiungi stampante** nella barra in alto, oppure fare clic con il tasto destro del mouse all'interno della finestra e scegliere **Aggiungi dispositivi e stampanti**.
2. Seguire le istruzioni visualizzate per la procedura di *Installazione guidata stampante*. All'interno di questa procedura si deve scegliere prima di tutto **Aggiungi stampante di rete** e quindi, a seconda dei casi, si può selezionare il nome della stampante dall'elenco delle stampanti disponibili in rete, oppure si può indicare direttamente il nome o l'indirizzo di rete della stampante.

Dopo aver completato l'installazione, sarà possibile utilizzare la stampante come se fosse fisicamente collegata al computer in uso.

AUTOVERIFICA

Domande da 8 a 11 pag. 40
Problemi da 6 a 9 pag. 41

MATERIALI ON LINE

9. Lezioni multimediali su Windows

DOMANDE

Concetti fondamentali

1 Quale delle seguenti definizioni corrisponde al termine *sistema*?

a) Un insieme di concetti relativi a una determinata materia.
b) Un insieme di enti che interagiscono tra loro per raggiungere un obiettivo comune.
c) Un modo di risolvere un problema.
d) Un insieme di problemi che devono essere risolti da più persone.

2 Qual è il significato del termine *modello*?

a) L'osservazione del funzionamento della realtà.
b) L'analisi di un problema da risolvere.
c) La rappresentazione semplificata della realtà osservata.
d) Un programma di gestione aziendale.

3 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?

a) Cifra binaria e bit sono sinonimi V F
b) Un bit è composto da otto byte V F
c) Un byte è composto da otto bit V F
d) Una parola è una stringa di byte V F

4 Completa in modo corretto le frasi seguenti:

a) **p and q** è vera solo nel caso in cui .....
b) **p or q** è falsa solo nel caso in cui .....
c) **p xor q** è vera solo nel caso in cui .....
d) **p and q** è falsa nel caso in cui .....

Sistema di elaborazione

5 Quale delle seguenti definizioni corrisponde al termine *architettura* riferito ad un computer?

a) La disposizione delle unità che compongono un sistema di elaborazione.
b) Il modello utilizzato per la progettazione di un sistema di elaborazione.
c) Le parti hardware di un sistema di elaborazione.
d) Le parti software di un sistema di elaborazione.

6 Quale delle seguenti frasi riferite alla memoria RAM *(Random Access Memory)* è corretta?

a) È la principale area di memoria utilizzata da un programma in esecuzione.
b) Era la memoria principale dei vecchi computer, sostituita ora dalla ROM.
c) Mantiene il proprio contenuto anche dopo lo spegnimento del computer.
d) È una memoria di massa che può essere inserita ed estratta dal computer.

7 Quale delle seguenti definizioni corrisponde al termine *memoria di massa*?

a) La memoria di lavoro della CPU.
b) La riproduzione su carta dei dati di output.
c) Un supporto di dati e di programmi dell'utente di un computer.
d) L'insieme della memoria RAM e della memoria ROM.

AUTOVERIFICA

Sistema operativo

8 Tenendo presente il tipo di organizzazione di un *sistema operativo*, assegna a ciascun modulo della colonna di sinistra una o più funzioni tra quelle elencate a destra.

a) Nucleo	1) ottimizzazione dei tempi di accesso a un disco
b) File System	2) gestione dell'organizzazione di un disco
c) Modulo di gestione delle periferiche	3) gestione delle pagine o dei segmenti di memoria
d) Interprete dei comandi	4) terminazione dei processi
e) Modulo di gestione della memoria	5) assegnazione della CPU ai diversi processi
	6) istruzioni per la formattazione di un disco
	7) assegnazione dei diritti di accesso ad un file
	8) controllo della correttezza sintattica di una linea di comando
	9) sincronizzazione dei processi

9 Quali tra i seguenti software sono sistemi operativi?
a) Windows
b) Excel
c) Android
d) Office

10 Quale tra queste definizioni corrisponde al termine *multitasking*?
a) Lo spegnimento automatico del monitor dopo un certo tempo di inattività.
b) L'apertura di un programma di trattamento del testo.
c) La possibilità di eseguire più applicazioni contemporaneamente.
d) La possibilità di usare filmati e suoni in un computer.

11 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
a) L'utente non può mettere in condivisione proprie risorse per gli altri utenti, perché è un compito dell'Amministrazione del sistema V F
b) La scelta *Everyone* consente di condividere una cartella con tutti gli utenti V F
c) Per utilizzare una stampante di rete condivisa, occorre scegliere *Aggiungi stampante* nel Pannello di controllo V F
d) Ogni computer collegato alla rete possiede un nome che lo identifica V F

PROBLEMI

Concetti fondamentali

1 Scrivere i seguenti numeri in forma esponenziale:
9867.5543
1.3244
87562123
6778899000000000000000

2 Se a = 5, b = 7 l'uso del connettivo and: (a < 3) and (b > 5) produce una proposizione vera o falsa?

3 Se a = 1, b = –6 l'uso del connettivo or:
(a > 2) or (b > 0)
produce una proposizione vera o falsa?

Sistema di elaborazione

4 Classificare gli oggetti elencati mettendo una crocetta nella colonna corrispondente.

	hardware	software
chip		
CPU		
driver		
disco magnetico		
masterizzatore		
microprocessore		
programma di contabilità		
Word		

	hardware	software
periferica		
plotter		
programma utente		
RAM		
scanner		
sistema operativo		
tools applicativi		
video		
CD-R		

5 Descrivere le caratteristiche tecniche del proprio computer domestico oppure del computer utilizzato nel Laboratorio di Informatica della scuola, seguendo la traccia della tabella seguente:

Produttore	
Modello	
Processore	
Memoria cache	
Memoria RAM	
Slot di espansione	
Collegamenti I/O (porte seriali, parallele, USB)	
Schema video	
Memorie di massa	
Multimedia (scheda audio, CD, casse)	
Tastiera	
Mouse	
Modem	
Scheda di rete	
Sistema operativo	
Software pre-installato	

Sistema operativo

6 Verificare le diverse modalità per passare da un'applicazione all'altra tra quelle attualmente aperte sul dektop.

7 Effettuare il backup dei documenti creati nell'ultimo mese.

8 Controllare quali risorse sono disponibili nella rete a cui è collegato il computer.

9 Mettere in condivisione una cartella del disco del computer.

Focus notes

OPERATING SYSTEMS

Architecture

The operating system architecture, at its highest level, acts as an intermediary between the underlying hardware and the user applications. The applications rarely talk to the underlying hardware directly. Instead, applications communicate with the hardware through a set of well-defined system interfaces that make the user application independent of hardware modifications. This abstraction makes it easy to write applications that work consistently on devices with different hardware capabilities.
The implementation of an operating system can be viewed as a set of layers. At the lower layers of the system are the fundamental services and technologies which work with all applications; higher-level layers contain more sophisticated services and technologies.

Memory and storage

A computer memory refers to the electronic holding place for instructions and data whereto a computer's microprocessor can quickly access (RAM and ROM). Computer storage refers to the permanent computer memory that stores data files and instructions even after the computer system is turned off (magnetic tapes, magnetic and optical disks, flash memories).

Virtual memory

In a virtual memory, the memory system is organised in two levels: the primary memory, consisting of only the main memory (RAM), and the secondary memory, consisting of the storage devices (disks). The information not immediately required by the CPU for the execution of processes can be moved, and retained, to the auxiliary memory and brought back into the main memory (from the disk) when required by the CPU. The main memory contains only the parts relative to the running computations. It is so that the operating system can accommodate significantly more processes than the main memory would normally do and can initiate new processes even if there is not sufficient space in the main memory to hold the entire process.

Operating systems for mobile devices

A mobile operating system is an operating system that controls a mobile device such as notebooks, smartphones or tablet computers. The most important are:
- *iOS*

iOS is Apple's mobile operating system for iPhone, iPod Touch and iPad. This operating system manages the device hardware and provides various system applications, such as notifications, mail, Safari browser, message services, newsstand for books and magazines, camera, photos, social networks and other standard system services. The user interface is based on touch controls.
- *Android*

Android, developed by Google, is a software stack for mobile devices that includes an operating system, middleware and applications. Android relies on Linux for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.
- *Windows Phone*

Windows Phone is a mobile operating system developed by Microsoft for smartphones. It is a compact operating system combined with a suite of basic applications and features for mobile devices. These features include multitouch screen, a user interface with a new modern design, social networking services, e-mail and a mobile version of Microsoft Office programs.

Focus notes

Glossary

boot
Term that means to start the computer. Booting a computer generally refers to loading the basic files, such as the operating system, required to run it.

buffer
A temporary storage area in a computer's memory, usually RAM, that holds recent modifications to files and other information to be written later-on to the hard drive.

interrupt
A command that tells the processor to stop what it is doing and wait for further instruction.

kernel
The main part of an operating system. The kernel handles the most basic, yet important tasks, such as starting processes and managing interrupts.

multitasking
A function of the operating system: it means that the operating system can simultaneously perform multiple tasks.

operating system
System software that controls and manages all of the hardware and software. The resources managed by an operating system include memory, files, processes, and input and output.

script
A text that contains a sequence of commands that the processor executes in batch mode.

shell
A program that provides an interface between the operating system and the user.

thread
A small, independent set of operations belonging to a larger program being executed.

Acronyms

CPU	Central Processing Unit
DLL	Dynamic Link Library
FS	FileSystem
GUI	Graphical User Interface
I/O	Input/Output
IPL	Initial Program Loader
MIPS	Million Instructions Per Second
OS	Operating system
RAM	Random Access Memory
ROM	Read Only Memory
SPOOL	Simultaneous Peripheral Operation On Line

SCHEDA DI AUTOVALUTAZIONE

CONOSCENZE

- ☐ Concetti di informazione, dato, elaborazione, sistema, modello, processo, processore
- ☐ Rappresentazione dei dati numerici e alfanumerici
- ☐ Codici ASCII e Unicode
- ☐ Algebra booleana
- ☐ Principi generali di funzionamento di un dispositivo automatico
- ☐ Caratteristiche e funzioni delle componenti fondamentali di un sistema di elaborazione
  - processore
  - memoria centrale
  - unità di input/output
  - memorie di massa
- ☐ Moduli del sistema operativo
- ☐ Significato di multitasking
- ☐ Caratteristiche generali dell'interfaccia delle applicazioni Windows
- ☐ Copie di backup
- ☐ Condivisione di risorse in rete

ABILITÀ

- ☐ Saper spiegare il significato dei termini fondamentali dell'informatica
- ☐ Rappresentare i numeri in notazione esponenziale
- ☐ Riconoscere il significato dei codici utilizzati in informatica per la rappresentazione dei caratteri
- ☐ Saper utilizzare gli operatori and, or, xor e not
- ☐ Spiegare il funzionamento di un dispositivo automatico
- ☐ Saper individuare le unità che compongono un sistema di elaborazione
- ☐ Riconoscere le funzioni fondamentali di un sistema operativo
- ☐ Saper operare con l'interfaccia grafica (mouse, finestre, icone)
- ☐ Effettuare il backup
- ☐ Mettere in condivisione una risorsa di rete
- ☐ Utilizzare una risorsa condivisa nella rete

SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 539

Il Prompt dei comandi di Windows

1 La finestra Prompt dei comandi

Il **Prompt dei comandi** è una *shell*, cioè un interprete di comandi, disponibile nel sistema operativo Windows che consente di eseguire i comandi che vengono scritti su una linea del video (*linea di comando*). La shell esegue un comando attraverso un nuovo processo e fornisce i risultati in modalità testuale.

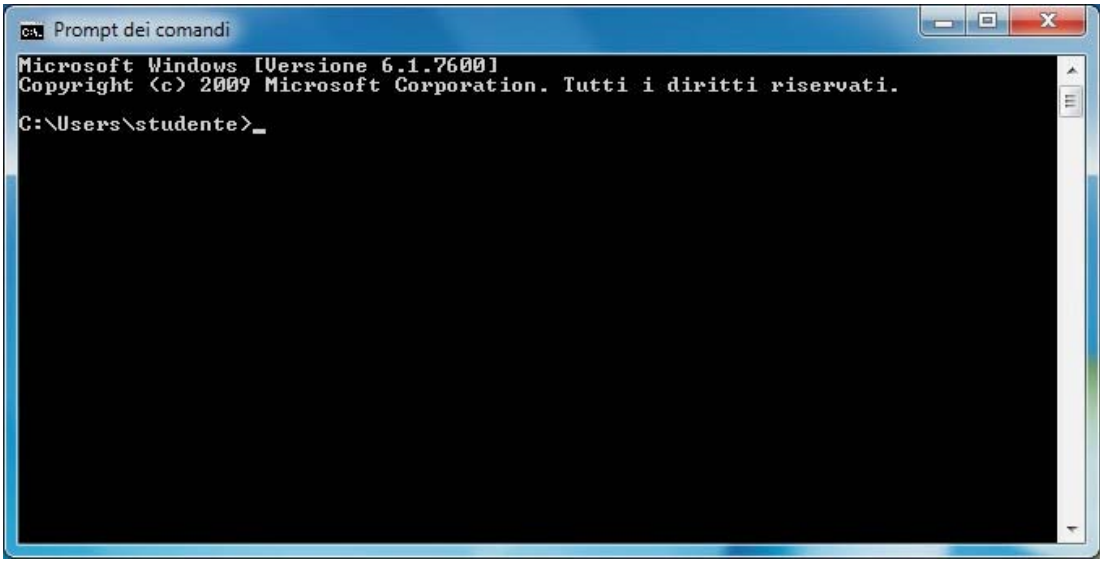
È la funzionalità che permette l'utilizzo dei comandi del sistema operativo MS-DOS, che era il sistema standard dei primi personal computer. Inoltre si possono attivare i comandi, in ambiente Windows, senza utilizzare l'interfaccia grafica con finestre, icone e mouse. Il prompt dei comandi è contenuto nel file **cmd.exe**.

Rispetto all'interfaccia grafica, l'uso del prompt di comandi rende più veloce l'attivazione delle funzionalità del sistema operativo, ma richiede la conoscenza delle parole chiave e della sintassi nella scrittura corretta delle linee di comando.

I comandi disponibili sono in numero limitato, ma l'utente può programmare script di comandi da eseguire in modalità *batch* (**file batch**, che hanno generalmente l'estensione **.bat**).

Il termine **prompt** (letteralmente *sollecito*) in informatica indica precisamente la sequenza di caratteri che compare sullo schermo quando la shell è pronta ad accettare un nuovo comando. Normalmente la sequenza contiene il nome dell'unità disco, il nome della home directory dell'utente e il segno di maggiore >, per esempio:

```
C:\Users\studente>
```



La finestra con la riga di comandi viene spesso indicata nei sistemi operativi anche come finestra di **console**.

Per aprire la finestra Prompt dei comandi

1. Dalla schermata **Start**, aprire **Tutte le app** (scorciatoia da tastiera **logo di Windows + Q**).
2. Nel gruppo **Sistema Windows**, scegliere **Prompt dei comandi**.

Per aprire la finestra Prompt dei comandi dalla casella di ricerca

1. Attivare la ricerca con la combinazione di tasti **logo di Windows + F**.
2. Nella casella **Cerca**, scrivere **cmd** e premere il tasto **Invio**.

In alternativa:

1. Premere il tasto con il **logo di Windows** insieme al tasto **R**.
2. Nella finestra di dialogo che si apre, scrivere **cmd** nella casella **Apri** e poi fare clic su **OK**.

Queste modalità servono anche a lanciare l'esecuzione di un programma o di un applicativo da linea comandi. Per esempio, scrivendo nella casella **Apri**:

```
calc.exe
```

si apre il programma *Calcolatrice*.

Per chiudere la finestra Prompt dei comandi

Scrivere **exit** al prompt della linea di comando
oppure
fare clic sul pulsante di chiusura della finestra in alto a destra.

2 I file e le directory

I file sono identificati attraverso un **nome** simbolico, che può permettere di ricordarne il contenuto, e un'**estensione** che, di solito, ricorda il tipo di file oppure il programma applicativo con il quale è stato creato. Il nome viene separato dall'estensione tramite il carattere . (punto).

Nel sistema operativo DOS il nome del file doveva avere una lunghezza massima di 8 caratteri e l'estensione di 3 caratteri. Nei sistemi Windows i nomi e le estensioni dei file non sono soggetti a queste restrizioni e possono essere molto più lunghi.

Ci sono alcune estensioni di uso comune:

nome.EXE	programma eseguibile
nome.COM	file di comandi
nome.SYS	file di sistema
nome.BAT	file batch, ossia una sequenza di comandi o di programmi da eseguire
nome.BAK	copia di sicurezza di un file
nome.TXT	testo semplice leggibile con un qualsiasi editori di testi.

Se il file che interessa non è registrato sul disco sul quale si sta lavorando, occorre far precedere al nome del file la lettera indicante l'**unità** sulla quale si trova il file (A, B, C, D, F, Z), seguita dal segno : (due punti).

DRIVE : **NOME DEL FILE** . **ESTENSIONE**

Di norma le unità con i floppy disk sono identificate con le lettere A: e B:, l'unità con l'hard disk con la lettera C:, le unità CD-ROM e DVD con le lettere D: ed E:, le memorie USB e i dischi esterni con F:, G:. Le unità e i dischi di altri computer connessi alla rete locale, di solito, sono indicate con le ultime lettere dell'alfabeto, per esempio W: oppure Z:.

Per esempio D:LETTERA.DOC indica un file di nome LETTERA ed estensione DOC che si trova sul disco ottico inserito nell'unità D:.

Per indicare i file in un comando è spesso utile far ricorso ai **caratteri jolly**, che sono il punto interrogativo e l'asterisco.

Il punto interrogativo **?** posto all'interno del nome del file, o dell'estensione, indica che un qualsiasi carattere può occupare quella posizione nel nome o nell'estensione del file.

Per esempio: *Prove?.dat* indica tutti i file che iniziano con *Prove* seguito da un qualsiasi carattere prima del punto, e di estensione *dat* (*Prove1.dat*, oppure *Prove5.dat*, oppure *Provex.dat*.) L'asterisco ***** inserito all'interno del nome del file o dell'estensione sta ad indicare che qualsiasi sequenza di caratteri può trovarsi in quella posizione all'interno del nome o dell'estensione del file.

Alcuni esempi:

MA*.*	indica tutti i file che iniziano per MA, aventi un'estensione qualsiasi
M*.C*	indica tutti i file con nome che inizia per M ed estensione per C
*M.*C	indica tutti i file con nome che termina con M ed estensione che termina con C
*.TXT	indica tutti i file di qualsiasi nome, aventi l'estensione TXT
.	indica tutti i file di qualsiasi nome con qualsiasi estensione.

Dal punto di vista logico, i file sono organizzati su disco attraverso le directory e le sottodirectory. La **directory** è un file particolare che contiene le informazioni di un insieme di file raggruppati di norma secondo un criterio di omogeneità (i file di uno stesso utente oppure i file di una determinata applicazione). Di ogni file vengono ricordate le seguenti informazioni: il nome, le dimensioni, la posizione fisica sul disco, la data e l'ora di creazione o dell'ultimo aggiornamento.

La directory di lavoro nella quale si trovano i file utilizzati in una determinata fase di elaborazione si chiama **directory corrente**.

La directory principale è detta **root** (*directory radice*) perché è la radice di una struttura ad **albero** costituita dalle sottodirectory e dai file.

Per accedere a un qualsiasi file contenuto in una sottodirectory diversa dalla directory corrente, si indica il cammino all'interno dell'albero, cioè l'elenco di nomi delle sottodirectory, separati dal segno \ *(backslash*) che devono essere attraversate per ritrovare il file richiesto.

Per esempio, per accedere al file *Esercizio.txt* contenuto nella sottodirectory *Lavori*, che a sua volta è contenuta nella directory *Utenti*, che è stata creata all'interno della root, si deve indicare come nome del file la seguente scrittura:

\Utenti\Lavori\Esercizio.txt

Questa scrittura prende il nome di **pathname** (*nome con indicazione del cammino*). Il primo segno di **** indica la root.

Se il pathname inizia con il segno \ la ricerca inizia dalla root (**pathname assoluto**), altrimenti inizia dalla directory corrente (**pathname relativo**) e prosegue verso il basso nella struttura gerarchica delle directory.

Naturalmente dalla directory corrente si può tornare anche verso l'alto nella struttura gerarchica o direttamente alla directory root.

3 I comandi

I comandi sono costituiti da una **parola chiave** che ne ricorda la funzione, da uno o più **argomenti**, dal nome del disco o, in generale, dell'unità contenente i file, dal nome dei file, e da una o più **opzioni** precedute dal simbolo **/** (barra o *slash*), che servono a specificare un utilizzo più specifico del comando.

La sintassi generale dei comandi è la seguente:

parola chiave	argomenti	unità:	nomefile	/opzioni

La lettera dell'unità periferica è seguita dal segno : (due punti), le opzioni sono precedute dal segno **/**.

Eventuali opzioni multiple applicate allo stesso comando sono indicate con segni / separati, per esempio:

```
DIR /P /O
```

Alcuni comandi non riguardano i file, pertanto l'indicazione dell'*unità* e del *nome file* può non essere presente. Se non si specifica l'*unità*, si intende quella corrente, normalmente il disco C:.
Inoltre nell'esecuzione dei comandi, il sistema operativo considera la tastiera come **standard input**, il video come **standard output** e come **standard error**, cioè come supporto per i messaggi di errore.
I comandi sono scritti su una riga del video (**linea comandi**) dopo la sequenza di caratteri detta **prompt**

```
C:\>
```

e sono eseguiti dopo aver premuto il tasto **Invio**.

C: significa che si sta lavorando sul disco fisso, **** rappresenta la directory *root*; il segno di **>**, seguito da un trattino di sottolineatura lampeggiante (detto **cursore**), sta ad indicare il punto del video dove l'utente deve digitare il comando.
Secondo le impostazioni predefinite del prompt, quando si cambia la directory corrente, il prompt cambia, specificando il pathname della directory nella quale si sta lavorando. Questa impostazione può essere modificata con il comando **PROMPT**.

L'utente può ottenere informazioni sui comandi utilizzabili tramite l'*help in linea*, attivato con il comando **HELP**. Il comando senza argomenti visualizza un elenco rapido dei comandi disponibili con una breve spiegazione della loro funzione:

```
HELP
```

La descrizione più dettagliata della sintassi di uno dei comandi elencati si ottiene scrivendo la parola *HELP* seguita dal nome del comando:

```
HELP nomecomando
```

In alternativa si può scrivere il nome del comando con l'opzione /?:

```
nomecomando /?
```

La finestra *Prompt dei comandi* non distingue la scrittura dei comandi con lettere maiuscole o minuscole, cioè non è *case sensitive*.

Durante la scrittura o la modifica dei comandi si possono usare i **tasti freccia** della tastiera: i tasti verso l'alto e il basso permettono di scorrere in avanti e indietro la sequenza dei comandi forniti in precedenza, mentre le frecce verso sinistra e destra consentono di spostare il cursore lungo la linea dei comandi, effettuando eventuali modifiche o aggiunte a righe di comando precedenti. Il tasto **Home** consente di posizionarsi all'inizio della linea comandi, il tasto **Fine** alla fine della linea stessa.

MATERIALI ON LINE

1. Editor di testi

4 I comandi per la gestione di directory e file

I comandi che operano su directory o file fanno riferimento ad essi secondo la notazione generale:

[unità:][percorso][nomefile]

Le parentesi quadre indicano l'opzionalità degli argomenti: *unità* o *percorso* possono essere omessi se il file si trova nell'unità corrente oppure nel percorso corrente; il *nomefile* è omesso quando i comandi operano su directory o sottodirectory.
Di seguito vengono elencate le funzioni svolte e la sintassi dei comandi di uso più comune.

• MKDIR
(*make directory*)
crea una nuova directory.
Esempio:

```
MKDIR \Contab
```

crea sul disco C: una nuova sottodirectory di nome *Contab* all'interno della directory root.
Il comando può essere usato anche nella forma abbreviata **MD**.

```
MD \Contab
```

• RMDIR
(*remove directory*)
cancella una directory (la directory non deve contenere file).
Esempio:

```
RMDIR C:\Utenti\Contab
```

la sottodirectory *Contab* della directory *Utenti* viene eliminata dalla struttura delle directory (l'utente deve prima cancellare tutti i file in essa presenti). Il comando può essere usato anche nella forma abbreviata **RD**. Il comando con l'opzione /S rimuove l'intero albero di una directory.

• CHDIR
(*change directory*)
seleziona una nuova sottodirectory come directory corrente.
Esempi:

```
CHDIR \Utenti\Contab
```

consente di posizionarsi in una nuova directory di lavoro *Contab* che è una sottodirectory della directory *Utenti*.

```
CHDIR \
```

consente di tornare alla directory root.

```
CHDIR ..
```

consente di risalire alla directory immediatamente sopra la directory corrente nella struttura gerarchica delle directory.

```
CHDIR
```

senza parametri consente di visualizzare il nome della directory corrente.
Il comando può essere usato anche nella forma abbreviata **CD**.

• **DIR**
(*directory*)
elenca sul video i file contenuti nella directory corrente o nella directory specificata.
Esempi:

```
DIR D:
```

fornisce il catalogo dei file che si trovano nell'unità D:, per ciascuno viene elencato il nome, l'estensione, l'occupazione di spazio su disco (espressa in byte), la data di creazione e l'ora di creazione del file.

```
DIR /W
```

elenca solo i nomi dei file con l'estensione.

```
DIR /P
```

visualizza l'elenco completo, fermandosi dopo aver visualizzato una pagina di video (24 righe), per consentire una comoda consultazione quando l'elenco è lungo.

```
DIR /P /O
```

visualizza l'elenco completo in ordine alfabetico di nome con 24 righe per pagina di video.

```
DIR D:Q*.*
```

fornisce l'elenco dei file il cui nome inizia con la lettera Q, con qualsiasi estensione, e che stanno sull'unità D:

```
DIR C:Prova.*
```

elenca i file che stanno sul disco fisso C: e hanno il nome *Prova*, con qualsiasi estensione.

```
DIR C:*.doc
```

elenca i file che stanno sul disco fisso che hanno l'estensione uguale a *doc* e con qualsiasi nome.
Le diverse opzioni del comando DIR possono essere visualizzate con il comando che attiva l'help in linea:

```
DIR /?
```

• TREE
visualizza con uno schema grafico la struttura dell'albero delle directory di un pathname o di un disco.
Esempio:

```
TREE \Temp
```

Visualizza l'albero delle sottodirectory e dei file nella directory *Temp* che è una sottodirectory della root.

• TYPE
visualizza il contenuto di un file, di cui viene specificato il nome.
Esempio:

```
TYPE Rubrica.txt
```

consente di ottenere sul video il testo delle righe contenute nel file *Rubrica.txt*.

• COPY
copia un file in un altro file.
Il primo argomento del comando indica l'origine, il secondo la destinazione della copia.
Esempio:

```
COPY D:Clienti C:Anag
```

copia il file di nome *Clienti* che si trova sul disco in D: sul disco C: assegnandogli il nome *Anag*.

• DEL
(*delete*)
cancella dal disco il file di cui viene specificato il nome. Il comando può eliminare anche un insieme di file specificati tramite i caratteri jolly (? e *).
Esempio:

```
DEL Prova.*
```

cancella tutti i file di nome *Prova* e di qualsiasi estensione.
Per eliminare un file si può usare anche il comando **ERASE**.

• RENAME
consente di cambiare il nome a un file registrato su disco.
Il primo argomento del comando indica il nome attuale, il secondo il nuovo nome.
Esempio:

```
RENAME Clienti Cliold
```

il file di nome *Clienti* assume il nuovo nome *Cliold.*
Il comando può essere usato anche nella forma abbreviata **REN**.

• MOVE
Sposta un file o una directory all'interno dell'albero delle directory. Questa operazione corrisponde al cambiamento del *pathname* di un file o di una directory, quindi il comando **MOVE** può essere usato anche per rinominare un file o una directory.
Il primo argomento del comando indica l'origine, il secondo la destinazione dello spostamento.

Esempi:

```
MOVE C:\Esercizi\Ese1 C:\Corretti\Problema1
```

sposta il file *Ese1* dalla directory *Esercizi* nella directory *Corretti* cambiando il nome in *Problema1*.

```
MOVE C:\Esercizi\Ese1 C:\Corretti
```

sposta il file *Ese1* dalla directory *Esercizi* nella directory *Corretti* mantenendone il nome.

```
MOVE C:\Esercizi C:\Esercizisvolti
```

cambia il nome della directory *Esercizi* con il nuovo nome *Esercizisvolti*.

• REPLACE
Sostituisce un file con un altro file. Il primo argomento del comando indica il percorso del file origine, il secondo la directory di destinazione della sostituzione.
Esempio:

```
REPLACE C:\Esercizi\Problema1 C:\Corretti
```

Il file *Problema1* che si trova nella directory *Esercizi* sostituisce il file con lo stesso nome che si trova nella directory *Corretti*.

• MORE
visualizza il contenuto di un file 24 righe per volta.
Viene usato per visualizzare i file che richiedono più videate. La sintassi generale del comando è la seguente:

```
MORE nomefile
```

Il comando manda al video le righe del file, facendo una pausa quando il video è stato riempito, e indicando nella parte bassa del video la parola *More* (in italiano *ancora*); per continuare con la videata successiva si deve premere la **barra spaziatrice**, per continuare riga per riga si usa invece il tasto **Invio**; per uscire senza vedere il resto si preme il tasto **Q**.

• SORT
ordina i dati contenuti in un file.
Normalmente il comando ordina alfabeticamente il contenuto di un file a partire dal primo carattere a sinistra di ogni riga. Il seguente comando

```
SORT Nomi
```

mette in ordine alfabetico le righe contenute nel file *Nomi* e visualizza il risultato.

• Ricerca di stringhe nei file
Ci sono due comandi per ricercare una o più stringhe in uno o più file, restituendo le linee nelle quali è stata trovata.
La differenza tra i due comandi consiste nella disponibilità di opzioni per raffinare la ricerca.
Il comando più semplice è **FIND** che ha la seguente sintassi generale:

```
FIND opzioni "stringa" nomefile
```

dove *stringa* indica la sequenza di caratteri da ricercare e *nomefile* può essere semplicemente il nome di un file nella directory corrente oppure il percorso di un file che si trova su un'altra unità o in un'altra directory.
Esempio:

```
FIND "Rossi" Rubrica.txt
```

visualizza le righe del file *Rubrica.txt* che contengono il cognome *Rossi*.
La stringa da cercare deve essere racchiusa tra virgolette.

Il comando **FINDSTR** svolge la medesima funzione, ma possiede molte più opzioni che possono essere visualizzate in modo completo con l'help in linea:

```
FINDSTR /?
```

Inoltre questo secondo comando può effettuare ricerche multiple con più stringhe.
La sintassi generale è analoga a quella del comando FIND:

```
FINDSTR opzioni stringhe nomefile
```

Per specificare più file su cui effettuare la ricerca si possono usare i caratteri jolly ? e *, oppure si può indicare il nome della directory che li contiene.
Per esempio il comando

```
FINDSTR "importo" *.dat
```

trova le righe che contengono la stringa *importo* nei file con estensione *dat*.

MATERIALI ON LINE

2. Attributi sui file

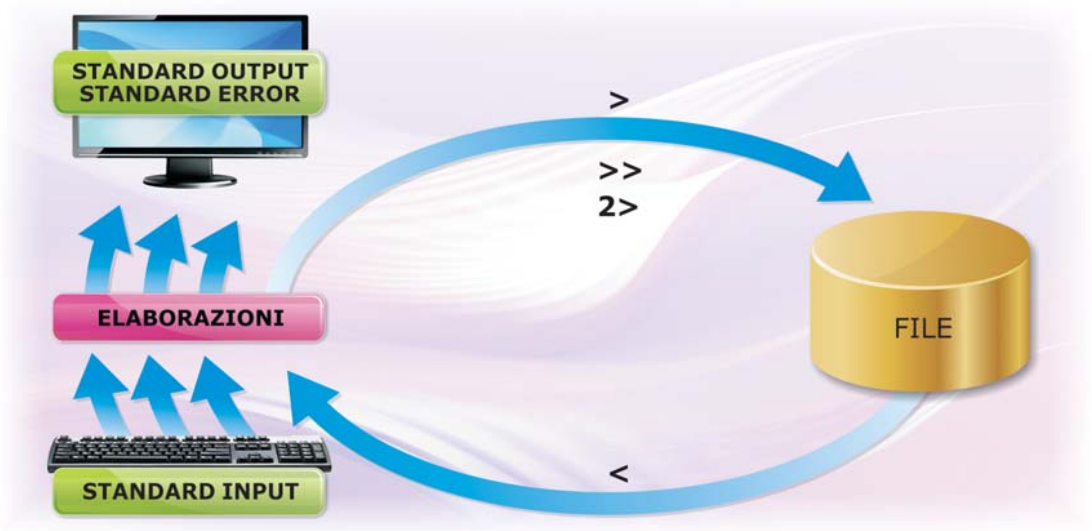
5 La ridirezione di input e output

Un comando della *shell* opera normalmente leggendo i dati dalla tastiera, denominata come **standard input**, cioè l'unità standard dalla quale vengono acquisiti i dati, e scrivendo i risultati sul video, indicato come **standard output**, cioè l'unità del sistema alla quale vengono inviati i risultati dell'elaborazione. Il video serve anche per produrre verso l'esterno i messaggi di diagnostica del sistema in caso di errore e per questo motivo viene indicato come l'unità che funziona da **standard error**.

Se si vogliono modificare queste assegnazioni standard per l'input, l'output e l'error occorre indicare il nome del file alternativo, poiché dal punto di vista del sistema i tre standard sono considerati file. Questa operazione si chiama **ridirezione**.

Essa viene rappresentata con i seguenti caratteri speciali (metacaratteri):

< considera il file indicato come lo standard input
> invia lo standard output al file indicato
>> appende l'output in coda al file indicato
2> ridirige lo standard error nel file indicato (il numero 2 è il tipo di descrittore esplicito per lo standard error, mentre 0 indica lo standard input e 1 lo standard output).



Esempi:

```
DIR *.txt >Testi
```

registra l'elenco dei file con estensione *txt* nel file *Testi*, anziché visualizzarlo sullo schermo.

```
DIR *.doc >>Testi
```

aggiunge l'elenco dei file con estensione *doc* in coda al file *Testi* senza sovrascrivere le righe già registrate.

```
SORT Nomi >Nomord
```

ordina alfabeticamente il file *Nomi* e registra l'elenco ordinato nel file *Nomord*.

Se il file *Prova.dat* non esiste nella directory corrente, il seguente comando:

```
DIR Prova.dat
```

produce sul video il nome della directory e poi il seguente messaggio di errore:

```
File non trovato
```

Con il comando:

```
DIR Prova.dat 2>errori
```

il messaggio di errore non compare sul video perché viene registrato nel file *errori,* dove potrà essere letto in un momento successivo (si può verificare il contenuto del file *errori* con il comando *TYPE errori*).

Volendo eliminare anche la visualizzazione del nome della directory si può usare il comando con la ridirezione dello standard output e dello standard error:

```
DIR Prova.dat >messaggi 2>errori
```

In questo modo sul video non compare alcun output del comando.

Output ed errori possono essere ridiretti anche su un file fittizio, avente il nome predefinito **NUL**, quando si vogliono fare ridirezioni senza registrare file nel disco; per esempio:

```
DIR Prova.dat 2>NUL
```

elimina la visualizzazione dei messaggi di errore e non li registra su un file fisico.

La ridirezione dell'input consente di fornire a un comando i dati di input acquisendoli da un file invece che dalla tastiera come avviene normalmente.

Si consideri il comando SORT senza argomenti:

```
SORT
```

non avendo specificato alcun file, la shell attende che l'utente inserisca i dati dallo *standard input* (tastiera); immettendo riga per riga, con il tasto *Invio* alla fine di ciascuna, i dati vengono acquisiti dal comando SORT; per interrompere l'inserimento si deve premere la combinazione di tasti **Ctrl + Z** e poi *Invio*; a questo punto il risultato dell'ordinamento viene mandato allo *standard output* (video).

Quindi il comando

```
SORT <Nomi
```

è una ridirezione dello standard input sul file *Nomi*, cioè i dati da ordinare vengono acquisiti dal file *Nomi*, anziché dalla tastiera.
Questa operazione è equivalente al comando visto nel paragrafo precedente per l'ordinamento di file

```
SORT Nomi
```

La seconda forma inoltre rende più rapida l'elaborazione.

In modo analogo il comando

```
SORT <Nomi >Nomord
```

ridirige sia lo standard input che lo standard output.

Un secondo esempio di ridirezione dell'input consiste nell'uso di un programma eseguibile *Prog1.exe* (realizzato compilando un sorgente scritto con un linguaggio di programmazione) che normalmente acquisisce i dati da tastiera: in modo diverso si può creare con l'editor un file di testo di nome *Dati.dat* contenente per ogni riga i dati da elaborare. In questo caso l'esecuzione si ottiene con la linea di comando:

```
Prog1.exe < Dati.txt
```

Questo modalità non richiede l'interazione con l'utente per fornire i dati al momento dell'esecuzione.

6 La pipeline

La **pipeline** è il modo più conveniente per utilizzare l'output di un comando come input di un altro comando senza creare file intermedi. Si chiama così perché letteralmente in inglese il termine *pipeline* significa tubatura oppure oleodotto: con questa metafora si vuole rappresentare una linea di comandi separati dal segno **|** (barra verticale, detta **pipe**, che si trova sulla tastiera a sinistra in alto sotto il tasto *Esc*) tale che l'output di un comando diventa l'input del comando successivo.

comando | comando | comando

Per esempio, se si vuole ottenere la lista dei file presenti nella propria directory e creati il 14 dicembre, si possono usare i due comandi:

```
DIR >Lista
FINDSTR "14/12/" Lista
```

Usando una pipeline si ha il vantaggio di ottenere lo stesso risultato con una sola riga di comandi e senza la necessità di creare il file temporaneo *Lista*:

```
DIR | FINDSTR "14/12/"
```

L'output del comando DIR diventa l'input per il comando FINDSTR.
L'effetto della precedente riga di comando è il seguente: DIR genera l'elenco dei nomi dei file, uno per riga, ciascuno descritto anche da dimensione, data e ora. Anziché visualizzare la lista sullo schermo, la shell la manda in ingresso al comando successivo, cioè FINDSTR, che provvede ad estrarre le linee contenenti la sequenza di caratteri specificata ed a visualizzarle. Se la lista è molto lunga, si può usare un'altra pipeline per visualizzare una pagina per volta, attraverso il comando MORE, nel modo seguente:

```
DIR | FINDSTR "14/12/" | MORE
```

L'output di FINDSTR viene così ulteriormente elaborato nel modo desiderato.
Per mezzo delle pipeline è possibile quindi effettuare operazioni molto complesse con una singola riga di comandi formata anche da più di due comandi successivi separati dai segni di *pipe*.
Esempi:

```
DIR | SORT /+13
```

lista i file della directory corrente in ordine crescente di ora di creazione.

```
DIR C:\Esercizi /S /B | FINDSTR "arch" | MORE
```

produce l'elenco senza intestazione (/B) dei nomi di file che si trovano nella directory *Esercizi* o nelle sue sottodirectory (/S) e che contengono nel nome la stringa "*arch*"; la visualizzazione avviene per pagine (24 righe per volta).

```
TREE C:\ | MORE
```

visualizza la struttura dell'albero delle directory del disco C: per pagine di video.

```
FINDSTR "Rossi" Fatture.txt | SORT
```

produce l'elenco ordinato delle righe del file *Fatture.txt* che si riferiscono al cliente *Rossi*.

```
FINDSTR /? | MORE
```

visualizza l'help in linea del comando FINDSTR per pagine di video.

```
HELP | MORE
```

visualizza l'elenco dei comandi, con una breve descrizione, per pagine di video.
Una pipeline può anche essere utilizzata con una ridirezione dell'output per costruire comandi complessi; si osservi che in questo caso l'operazione di ridirezione deve essere in coda alla riga di comando. Per esempio:

```
DIR /B | FINDSTR "arch" > ListaArch.txt
```

GESTIONE DEL SISTEMA E DELLE PERIFERICHE

In questo paragrafo vengono illustrati alcuni tra i comandi del sistema operativo Windows che riguardano la gestione del sistema di elaborazione, delle sue periferiche e il controllo dello stato dei processi. Anche se richiamano operazioni che sono più propriamente di competenza dell'*Amministratore del sistema*, essi risultano spesso utili anche per l'utente generico. Si deve osservare, comunque, che alcuni di questi comandi richiedono esplicitamente i privilegi di *Amministratore*.

- **DATE**
consente di visualizzare o modificare la data del computer (giorno, mese, anno).

- **TIME**
consente di visualizzare o modificare l'ora del computer (ore, minuti, secondi e centesimi di secondo).

- **FORMAT**
serve a preparare un supporto di memoria di massa per essere usato sul computer.
Esempio:

```
FORMAT F:
```

viene formattato il disco che si trova nell'unità F:

- **VOL**
visualizza l'etichetta di volume e il numero seriale di un disco (fisso o esterno).
Esempio:

```
VOL C:
```

visualizza le informazioni del disco C:

- **DISKCOPY**
copia il contenuto di un disco su un altro disco.
Esempio:

```
DISKCOPY A: B:
```

copia il disco che si trova nell'unità A: (originale) sul disco che si trova nell'unità B: (disco destinazione).

• **CHKDSK**
controlla un disco e visualizza il relativo rapporto sullo stato dei file e delle cartelle, oltre ad informazioni su capacità del disco, spazio occupato, numero di file e cartelle.
Esempio:

```
CHKDSK E:
```

effettua un controllo della memoria esterna USB identificata con la lettera E:
Le informazioni sulla **configurazione del sistema** di elaborazione possono essere visualizzate con i seguenti comandi:

• **SYSTEMINFO**
visualizza la configurazione e le proprietà delle componenti del sistema di elaborazione in uso.

• **VER**
visualizza la versione del sistema operativo in uso.

• **HOSTNAME**
visualizza il nome del computer in uso. È un comando senza argomenti.

• **IPCONFIG**
visualizza l'indirizzo IP del computer in uso.

```
IPCONFIG
```

visualizza, per ogni scheda presente nel computer e associata al protocollo TCP/IP, l'*indirizzo IP*, la *subnet mask* e il *gateway* predefinito.

```
IPCONFIG /ALL
```

visualizza le informazioni in modo completo, in particolare per ogni scheda di rete è possibile ottenere l'indirizzo fisico (*MAC address*).

• **PING**
verifica la connessione ad un altro computer della rete di cui si specifica l'indirizzo:

```
PING 192.168.1.2
```

Con questo comando viene inviato un pacchetto di dati di prova al computer avente l'indirizzo IP indicato come parametro del comando: se non si ottiene una risposta entro pochi secondi, significa che l'altro computer non è connesso alla rete. Ovviamente l'esecuzione senza successo di questo comando potrebbe anche dipendere dal fatto che il proprio computer non è correttamente connesso alla rete.
Il comando PING seguito da un indirizzo Internet consente di ottenere l'indirizzo IP del server Internet associato nel DNS (*Domain Name System*); per esempio:

```
PING www.google.it
```

verifica la possibilità di connessione al server *www.google.it* e visualizza il suo indirizzo IP.

MATERIALI ON LINE

3. File batch

PROBLEMI

Comandi

1 Ritornare alla directory root trovandosi nella sottodirectory CLIENTI.

2 Elencare i file che stanno sul disco F: nella directory BETA e che hanno il nome di 4 caratteri ed estensione qualsiasi.

3 Scrivere i comandi per visualizzare sul video:
- l'elenco dei file il cui nome inizia con la lettera A
- l'elenco dei file che hanno estensione TXT
- l'elenco dei file che hanno il carattere 1 al terzo posto nel nome.

4 Controllare se il file CONTI è contenuto nella sottodirectory AZ00 del disco C:, indicando il suo pathname assoluto.

5 Creare una directory NUOVA nella root trovandosi nella sottodirectory ANAGRA.

6 Cancellare dalla directory COPIA del disco C: i file che iniziano per PRO e hanno l'estensione che inizia con P.

7 Cancellare tutti i file che iniziano con P3 e hanno estensione XLS.

8 Spostarsi dalla root del disco C: alla sottodirectory ARCHIVI della directory CONTAB; indicare poi due modi possibili per tornare alla root.

9 Cambiare il nome al file ARTICOLI.DAT nel nuovo nome PRODOTTI.OLD dopo averlo copiato dalla directory MAGAZ del disco C: sull'unità E:.

10 Copiare il file PROVA1 della directory COPIA del disco C: sulla chiavetta USB, denominata con F:, nella directory TRASF.

11 Copiare tutti i file con estensione DOC, che stanno nella directory TESTI del disco C:, sul disco D: nella directory T01.

12 Scrivere il comando per copiare nella directory PROVA2 tutti i documenti Word presenti nella directory \DOCUMENTI del CD identificato come unità D:.

13 Controllare la data e l'ora del computer, e correggerle nel caso siano errate.

14 Cancellare la directory DOCUMENTI del disco F:.

15 Con l'opportuno comando, attivare l'aiuto in linea.

16 Scegliere un file di testo dal disco fisso del computer e visualizzare il suo contenuto sul video.

17 Elencare i file che hanno nel nome in seconda posizione la lettera A e nell'estensione in terza posizione la lettera M.

18 Cambiare il nome al file INTESTA della directory DISCO nel nuovo nome PRIMA.

19 Nell'albero delle directory di un disco G: ci sono le directory \ALFA e \ALFA\BETA. Quale comando occorre dare per cancellare la directory ALFA dal disco G:?

20 Creare una nuova directory sull'unità F: di nome PROVA2 e dentro di essa la directory ARCHIV2.

21 Cancellare i file VIDEO2 e STUD2, che si trovano rispettivamente nelle sottodirectory TERZAA e TERZAB, usando i pathname assoluti.

AUTOVERIFICA

22 Verificare che i file del problema precedente siano stati cancellati e controllare lo spazio disponibile su disco dopo questa operazione.

23 Scegliere un file dal disco fisso del computer e copiarlo nella directory principale di una chiavetta USB attribuendogli lo stesso nome. Con l'opportuno comando, cambiare poi il nome al file in PROVA.DOC.

24 Scelta una directory del disco fisso, con un solo comando e facendo uso dei caratteri jolly, copiare su un'unità esterna G: i file presenti nella directory e che hanno estensione EXE.

25 Scrivere il comando per ottenere sul video il contenuto del file CLIENTI.TXT, 24 righe per volta.

Ridirezioni e pipeline

26 Elencare i file con estensione .DOC e con la seconda lettera del nome uguale a R; si vuole ottenere l'output del comando nel file DocumentiR.

27 Produrre in output l'elenco dei file creati o modificati nell'anno scorso.

28 Elencare i file modificati o creati nel mese di febbraio, visualizzandoli 24 righe per volta.

29 Un file di nome STUD2 nella directory ARCHIV2 contiene un elenco di nomi di studenti con le rispettive età: verificare se nel file ci sono studenti di 26 anni e, se ci sono, elencarli in ordine alfabetico.

30 Scrivere la linea di comandi che consente di visualizzare i nomi dei file della directory corrente che sono stati creati nell'anno corrente e, all'interno di questi, quelli con estensione SYS.

31 Scrivere il comando per memorizzare in un file l'elenco alfabetico dei file creati nel giorno 2 febbraio contenuti nella directory \LAVORI.

32 Ridirigere la visualizzazione della data di oggi su un file di nome GIORNO.

33 Visualizzare i file del disco C: che sono stati creati o modificati due anni fa.

34 Un file CLIENTI contiene un elenco di cognomi e nomi: ordinare alfabeticamente il file creando un nuovo file ordinato di nome CLIORD.

35 Controllare se tra i nomi dei clienti del problema precedente esiste il cliente di Cognome = ROSSI.

36 Visualizzare la lista ordinata dei file presenti nella directory UTILITY.

37 Scrivere una pipeline che consenta di visualizzare soltanto il numero dei byte ancora disponibili su un disco.

38 Visualizzare l'elenco ordinato dei soli nomi dei file contenuti nella directory F:\LAVORI.

39 Scrivere il comando che consente di ottenere l'elenco alfabetico dei file con estensione .TXT creati o modificati il 4 maggio.

40 Scrivere su un nuovo file l'elenco dei file che hanno una B nella quarta posizione del nome e hanno estensione EXE.

41 Produrre la lista ordinata dei file che iniziano con R ed estensione qualsiasi creati nello scorso anno (utilizzare l'opzione */O* del comando *DIR*).

42 Comunicare il contenuto del file NOMI.DAT su standard output.

43 Cercare in un file le righe che contengono la parola NAPOLI visualizzandole 24 righe per volta.

parte seconda

La programmazione

2 Linguaggi e programmi

OBIETTIVI DI APPRENDIMENTO

In questo capitolo conoscerai il concetto di algoritmo e saprai riconoscere le caratteristiche fondamentali delle istruzioni che compongono un algoritmo.
Imparerai a costruire algoritmi ben ordinati attraverso le strutture di controllo.
Conoscerai i diversi paradigmi di programmazione e gli aspetti evolutivi dei linguaggi di programmazione.

1 Modello del problema

Il termine **modello** indica la rappresentazione schematica di un particolare aspetto della realtà. In particolare un modello permette, attraverso meccanismi di formalizzazione, di individuare gli elementi principali della realtà osservata e di trattare tali elementi come entità astratte.

Queste entità devono avere due caratteristiche fondamentali:

- devono essere registrabili in una memoria
- devono essere messe in relazione tra loro.

Il trasferimento di un problema sul sistema di elaborazione viene indicato nel linguaggio informatico con il termine **implementazione**.

Si parla di implementazione di un problema quando la sua soluzione viene affrontata attraverso la costruzione di un modello formalizzato, in modo tale che il problema possa essere rappresentato e gestito con un sistema di elaborazione automatica.

L'automazione di un'attività implica la necessità di studiare un sistema, rappresentare la realtà osservata in modo semplificato attraverso un modello, individuare, classificare e selezionare i dati. Le previsioni del tempo atmosferico richiedono, per esempio, la disponibilità dei dati riguardanti la temperatura e la pressione, rilevate non solo in un determinato luogo, ma in molte località geograficamente distanti.
All'interno del problema considerato vengono quindi individuate le **entità** che sono importanti: ciascuna di esse può essere presente nella realtà osservata con diversi *esemplari*, ai quali associamo i diversi valori che possono essere assunti dall'entità.

Per esempio, nello studio delle previsioni del tempo atmosferico entrano diverse entità:

- il luogo al quale si riferiscono le previsioni
- le grandezze fisiche che vengono misurate
- la misurazione.

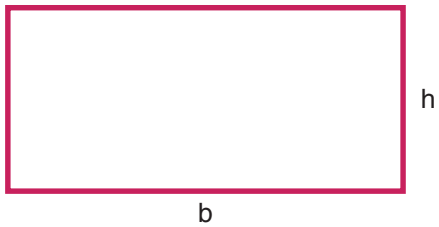
Gli esemplari di queste entità sono le diverse misurazioni che un istituto metereologico esegue in date diverse, le varie grandezze fisiche che l'istituto misura in ogni studio e i luoghi che fanno parte della zona competente a quell'istituto.
Per fornire una rappresentazione più precisa, ma soprattutto più formalizzata, nel modello vengono introdotte le proprietà delle entità.
Per esempio sono **proprietà** del luogo: il nome, la latitudine, la longitudine, l'altezza sul livello del mare; per le grandezze misurate: la descrizione (come la temperatura, la pressione, l'umidità, la direzione e la velocità del vento, le precipitazioni), l'unità di misura; per la misurazione: la data, l'ora del rilevamento, i valori di ciascuna grandezza misurata.
Queste proprietà possono assumere valori diversi per misurazioni diverse: quindi diciamo che sono *variabili*, cioè stanno al posto dei valori specifici che verranno considerati in relazione a una determinata misurazione.

Le **variabili** sono proprietà delle entità che possono assumere valori diversi per diversi esemplari della stessa entità.

Nelle formule della matematica l'uso delle variabili è una prassi comune: per calcolare l'area di un rettangolo si utilizza la formula

$$A = b \times h$$

dove *A*, *b* e *h* sono nomi che stanno al posto dei valori rispettivamente dell'area, della base e dell'altezza del rettangolo. Questi nomi saranno poi sostituiti dai valori effettivi delle misure riferite al particolare esemplare di rettangolo del quale si vuole calcolare l'area.



$A = b \times h$

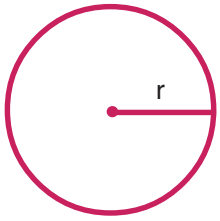
Considerando poi problemi dello stesso tipo, è possibile spesso identificare delle proprietà che assumono sempre gli stessi valori, o comunque mantengono lo stesso valore per un ragionevole periodo di tempo.

Queste proprietà vengono definite **costanti**, cioè proprietà che non cambiano considerando problemi con variabili diverse.

Esempi di *costanti in senso stretto* sono sicuramente il valore di π (pi-greco), che rappresenta il rapporto tra la circonferenza e il diametro di un cerchio, e la costante *g* nella formula del moto uniformemente accelerato:

$$s = \frac{1}{2} g t^2.$$

Quindi nel problema del calcolo dell'area del cerchio, la misura del raggio e dell'area sono variabili, pi-greco è una costante.



$A = r^2 \times \pi$

In ambito scientifico esempi di *costanti in senso lato* possono essere i valori del coefficiente d'attrito utilizzato per calcolare la forza d'attrito tra un oggetto e la superficie su cui si trova: questi valori possono cambiare in base al tipo di attrito (radente, volvente, statico, dinamico), ma si mantengono costanti all'interno dell'insieme di problemi dello stesso tipo. Nel problema del calcolo della forza d'attrito statico, sono variabili la massa dell'oggetto, l'inclinazione della superficie su cui si trova e l'entità della forza d'attrito calcolata ed è costante il valore del coefficiente d'attrito.

I nomi che diamo alle variabili o alle costanti, per distinguerle all'interno del modello considerato, si chiamano, in modo generico, **identificatori**.

2 Dati e azioni

La soluzione di un problema richiede da una parte la disponibilità dei *dati* necessari per effettuare i calcoli e dall'altra un insieme di *azioni* da svolgere per ottenere il risultato finale.

I **dati** sono i valori assunti dalle proprietà degli elementi che caratterizzano il problema, rappresentate con variabili o costanti.
Le **azioni** sono le attività che, mettendo i dati in relazione tra loro, consentono di ottenere i risultati desiderati.

PROGETTO 1

Dopo aver acquisito il costo del carburante al litro, dato un elenco di automobili con targa e litri di carburante consumati, contare quante auto hanno avuto un costo totale superiore a 200 euro e fornire alla fine il risultato del conteggio.

Targa	Litri
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....

Una buona regola pratica, per individuare quali sono i dati e quali le azioni presenti nel problema, consiste nel trattare i sostantivi come dati e i verbi come azioni.

Nell'esempio, i dati sono:
- costo di un litro di carburante
- targa dell'auto
- litri consumati
- costo dell'auto
- costo minimo (200 euro)
- risultato del conteggio.

In un problema i **dati** possono essere di tipo diverso:
- *numerici*, come nel caso di stipendi dei dipendenti, età delle persone, importi da pagare, dimensioni di un oggetto;
- *alfabetici*, come il nome di una persona, il titolo di un libro, la descrizione della ragione sociale di un'azienda, il nome di un prodotto commerciale o di un elemento chimico;
- *alfanumerici*, detti comunemente anche **stringhe**, cioè dati che sono rappresentati con cifre e lettere, quali il codice fiscale di un cittadino, la targa di un'automobile, la formula di un composto chimico oppure una formula matematica.

Una regola di carattere generale è la seguente: se un dato viene utilizzato in procedimenti di calcolo, allora è un dato di tipo numerico, altrimenti è alfabetico o alfanumerico.
Per esempio la partita IVA di un'azienda o il numero di telefono di un abbonato sono dati rappresentati con cifre numeriche, ma non vengono mai usati per fare addizioni o prodotti, e pertanto possono sicuramente essere considerati dati alfanumerici.

L'elenco di automobili, pur essendo un sostantivo, è un dato di tipo *non elementare* che comprende un insieme di dati elementari, che sono la targa e i litri di carburante di ciascuna auto. Il problema richiede di operare sui dati *elementari*, nel senso che le automobili devono essere esaminate una ad una per controllare il costo totale di carburante.
Si deve anche notare che il problema richiede di controllare il costo dell'automobile, che non è un dato immediatamente disponibile, in quanto nell'elenco compare la quantità di litri e non il costo: quindi abbiamo bisogno di un ulteriore dato, il costo del carburante al litro, che può essere considerato, in questo contesto, come una *costante*, in quanto non varia esaminando automobili diverse.

Identificatore	Variabile o costante	Descrizione	Tipo
Costo unitario	costante	Costo di un litro di carburante	numerico
Targa	variabile	Targa dell'automobile	alfanumerico
Litri	variabile	Litri di carburante consumati	numerico
Costo totale	variabile	Costo per un'auto	numerico
Costo minimo	costante	Costo minimo per il controllo	numerico
Contatore	variabile	Risultato del conteggio	numerico

Rileggendo il testo del problema si possono identificare i verbi che corrispondono alle azioni da svolgere per trovare i risultati:

- esaminare l'elenco;
- contare le automobili;
- fornire il risultato.

A queste azioni va aggiunto il calcolo del costo per ciascuna auto, conoscendo il numero di litri consumati e il costo unitario del carburante, attraverso la formula:

Costo totale = Costo Unitario x Litri

Si noti come, in questa azione di calcolo, più dati, variabili e costanti, vengono combinati tra loro per formare un'espressione matematica.

Le **azioni** possono essere riconducibili ad operazioni:

- di tipo *aritmetico*, quali i calcoli;
- di tipo *logico*, quali il confronto tra due valori o il controllo del valore di verità di un enunciato.

Per esempio:

- il calcolo del 3% di un totale è un'operazione di tipo aritmetico;
- la somma di N numeri è un'operazione di tipo aritmetico;
- il confronto A > B è un'operazione di tipo logico;
- il controllo per stabilire se è stata raggiunta la fine dell'elenco nella lettura di una lista di nomi è un'operazione di tipo logico;
- il controllo per stabilire se il numero di telefono di una persona è stato trovato in una rubrica è un'operazione di tipo logico.

L'azione *Esaminare* indicata precedentemente è un'azione complessa che corrisponde all'insieme di tante piccole azioni, tutte dello stesso tipo, che controllano i dati delle persone dell'elenco, una ad una.

Quindi, in generale, l'azione di *Esaminare un elenco* significa in pratica, e anche nella formalizzazione dei problemi, prendere in considerazione i dati scritti nelle singole righe che compongono l'elenco, una per volta dalla prima riga fino alla fine dell'elenco.

3 L'algoritmo

Indipendentemente dalla dimensione e dall'importanza che assume il problema, il processo di risoluzione può essere diviso in almeno due passi fondamentali che possono essere descritti nel seguente modo.

Passo 1: Descrizione del problema

In questa prima fase occorre definire con precisione: i dati che abbiamo a disposizione, i confini delle soluzioni adottate, i risultati attesi e le risorse da utilizzare per elaborare i dati iniziali e per ottenere la soluzione del problema.

Schematicamente gli elementi che caratterizzano un problema sono:

- i dati che servono ovvero i dati iniziali sui quali basare la soluzione del problema da affrontare. I dati iniziali vengono anche detti dati di ingresso, o **dati di input**. Tali dati non devono essere né sovrabbondanti né incompleti.
- i risultati che si vogliono ottenere, detti anche dati di uscita o **dati di output**.
- le risorse a disposizione sia dal punto di vista logico (tabelle, criteri di calcolo, schemi logici, leggi e formule, ecc.) sia dal punto di vista fisico o hardware (strumenti di calcolo quali macchine calcolatrici, personal computer o calcolatori di grosse dimensioni).
- le soluzioni adottate, ovvero il procedimento e il percorso che permettono di passare dai dati iniziali ai risultati attesi.

L'individuazione di questi elementi costituisce l'**analisi del problema**.

Passo 2: Stesura dell'algoritmo

La soluzione del problema, così come è stata descritta nella prima fase, deve essere ora organizzata e sviluppata in una serie di operazioni da attuare secondo un ordine ben definito, che permette di giungere ai risultati attesi a partire dai dati iniziali. La scomposizione del procedimento risolutivo di un problema, in una sequenza di operazioni elementari da seguire per ottenere i risultati attesi, costituisce la stesura dell'algoritmo risolutivo.

Nell'accezione più diffusa il termine **algoritmo** indica un procedimento, in generale per il calcolo di grandezze o di quantità, basato sull'applicazione di un numero finito di operazioni, che specificano in modo rigido e meccanico ogni singolo passo del procedimento risolutivo.

Il termine algoritmo proviene dalla matematica e sta a indicare le regole, o in generale le operazioni da compiere, per fare un calcolo o risolvere un problema. Deriva dal nome di un algebrista arabo del IX secolo di nome **Al-Khuwarizmi**.

L'**algoritmo** è la descrizione di un insieme finito di istruzioni, che devono essere eseguite per portare a termine un dato compito e per raggiungere un risultato definito in precedenza.

Esempi di algoritmi possono essere: la ricetta di cucina, le istruzioni per l'utilizzo di un elettrodomestico, le regole per eseguire la divisione tra due numeri, le indicazioni per la consultazione di un orario ferroviario, ecc.

Va notato, da subito, che nella definizione è implicita la presenza di un **esecutore**, in altre parole di un oggetto, macchina, ente o persona alla quale affidare l'interpretazione e l'esecuzione delle istruzioni contenute nell'algoritmo.

PROGETTO 2

L'addetto alla Biblioteca comunale, disponendo dell'elenco degli utenti che hanno preso in prestito i libri, con cognome, nome e data prestito, deve mandare un avviso agli utenti per i quali sono trascorsi più di 30 giorni dal prestito (si suppone che l'esecutore sia in grado di calcolare i giorni intercorrenti tra due date facendone la differenza).

Il bibliotecario deve procurarsi la lista delle persone che hanno effettuato prestiti di libri. Scorrendo l'elenco, per ciascun utente deve controllare se sono trascorsi più di 30 giorni dal prestito. In caso affermativo deve annotare il numero telefonico accanto al cognome, altrimenti deve mettere solo un segno di spunta per ricordare di aver controllato l'utente.

Dati di input (*dati che servono*):
l'elenco delle persone con cognome, nome, data del prestito
numeri telefonici degli utenti.

Dati di output (*risultati che si vogliono ottenere*)**:**
l'elenco degli utenti con l'indicazione per ciascuno dell'esito del controllo.

Algoritmo

Per ciascun utente dell'elenco:
    leggi cognome, nome e data prestito
    calcola il numero di giorni trascorsi dal prestito
    a seconda della situazione che si presenta:
        scrivi sull'elenco il numero telefonico oppure un segno di spunta.
Ripeti le operazioni precedenti mentre l'elenco non è finito.

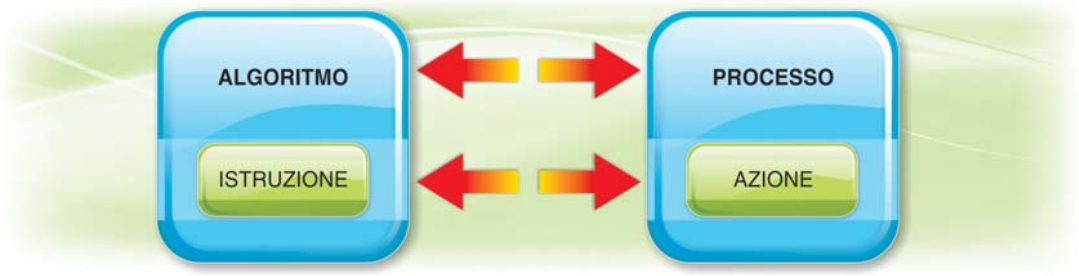
L'algoritmo è presentato secondo un linguaggio volutamente privo di qualsiasi formalismo. Più avanti saranno presentate le regole fondamentali e gli strumenti di base che servono per l'organizzazione di un algoritmo ben strutturato.

MATERIALI ON LINE

1. Caratteristiche dell'algoritmo

4 Algoritmo ed esecutore

Un'**azione** è un evento che si compie in un intervallo di tempo finito e che produce un risultato, un effetto, previsto e ben determinato. Ogni azione modifica lo stato di qualche oggetto, e il suo effetto può essere riconosciuto dal cambiamento di stato dell'oggetto in questione. Per descrivere le azioni è necessario disporre di un **linguaggio** o di una notazione formale: le descrizioni delle azioni sono dette **istruzioni**.
Un'istruzione elementare è un'istruzione che non può essere scomposta in istruzioni più semplici. Un'istruzione non elementare è detta **algoritmo**.
Un'istruzione composta descrive un'azione composta, vale a dire un'azione che può essere frammentata in azioni più semplici. Un'azione composta è chiamata **esecuzione** (o **processo**).
Un *algoritmo* è dunque composto da un certo numero di *istruzioni*, ad esso corrisponde un *processo* composto da un certo numero di *azioni*; in generale, non vi è corrispondenza tra la successione temporale delle azioni e la disposizione delle istruzioni all'interno dell'algoritmo.

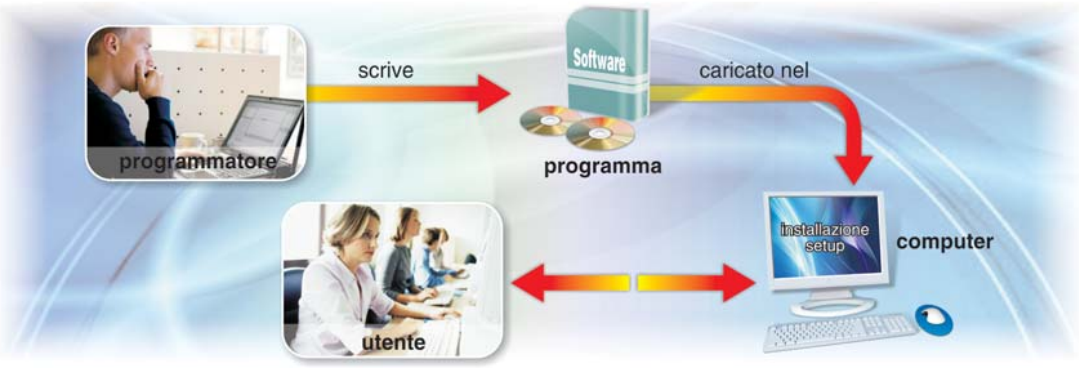


L'**esecutore** (o **processore**) è l'ente che esegue le azioni secondo le istruzioni di un algoritmo.

Esecutore è un nome generico, che non indica se si tratta di un esecutore umano oppure automatico. In pratica l'esecutore è da intendersi come l'ente che procede a compiere l'esecuzione del processo risolutivo.
Inoltre, per una descrizione generale dell'ambiente, in cui si opera con l'informatica, vanno definite due figure fondamentali:

- il **programmatore** ovvero colui che organizza, prepara e scrive l'algoritmo,
- l'**utente** ovvero la persona che attiva l'esecuzione dell'algoritmo e che interagisce con l'esecutore per fornire ad esso i dati iniziali del problema e per utilizzare i risultati finali.

L'esecutore può essere una persona, una macchina o un insieme di persone e macchine.
Nel seguito, per esecutore si intenderà principalmente un computer. In tal caso l'algoritmo dovrà essere scritto secondo un linguaggio definito, con regole rigorose, in modo da essere comprensibile da parte del computer.
In questo caso l'algoritmo diventa un **programma**.



5 Acquisire e comunicare i dati

La risoluzione di un problema comporta l'individuazione di tre elementi ben definiti e indispensabili per una corretta definizione dell'algoritmo risolutivo: si tratta dei dati iniziali, della soluzione adottata e dei risultati finali. Le relazioni che intercorrono tra questi tre elementi possono essere schematizzate nel seguente modo:

Ne consegue che un esecutore che provvede a compiere le azioni richieste dalle istruzioni di un algoritmo deve essere in grado di acquisire i dati iniziali, su cui attivare il processo di elaborazione, e deve essere in grado di comunicare all'esterno i risultati finali ottenuti: in altre parole deve essere in grado di leggere dall'esterno i dati in ingresso (o *dati di input*) e deve essere in grado di scrivere verso l'esterno i dati di uscita (o *dati di output*).

I **dati di input** sono quelli che vengono forniti dall'esterno per poter risolvere il problema; i **dati di output** sono quelli che vengono comunicati all'esterno, come risultato della soluzione del problema.

Ci possono essere poi altre variabili, che non sono né di input né di output, ma che sono comunque necessarie all'attività di elaborazione per ottenere risultati parziali, e che vengono chiamate **variabili di lavoro,** temporanee o di calcolo.

PROGETTO 3

Date le misure dei due cateti di un triangolo rettangolo, calcolare la misura del perimetro del triangolo.

Le misure dei due cateti devono essere fornite dall'esterno perché sono necessarie per il calcolo richiesto; l'ipotenusa è una variabile di lavoro, perché non viene fornita dall'esterno, ma per calcolare il perimetro serve la sua misura; il perimetro è il valore da comunicare come risultato finale, ed è perciò un dato di output.

Il tutto può essere schematizzato in questo modo:

Dati	Input	Output	Lavoro
Cateto1	✔		
Cateto2	✔		
Ipotenusa			✔
Perimetro		✔	

PROGETTO 4

Calcolo del numero di molecole contenute in una massa di acqua.

Dall'esterno devono essere acquisiti i valori della massa d'acqua, della massa molare dell'acqua e della costante di Avogadro, che sono dati di input, il numero di moli è una variabile di lavoro, e il numero di molecole è il risultato da comunicare al termine dell'elaborazione, e quindi è il dato di output.

Dati	Input	Output	Lavoro
Massa di acqua	✔		
Massa molare	✔		
Costante di Avogadro	✔		
Numero moli			✔
Numero molecole		✔	

Per fare in modo che l'esecutore acquisisca i dati e comunichi i risultati, nell'algoritmo che richiede tali azioni, per le operazioni di *input* vengono indicate istruzioni del tipo:

- immetti
- leggi
- acquisisci
- accetta
- read
- accept

ecc.

e per le operazioni di *output*:

- scrivi
- comunica
- mostra
- write
- display

ecc.

6 Gli operatori

Un'altra azione fondamentale che risulta comunque presente in un processo eseguito da un calcolatore è l'**assegnamento di un valore a una variabile**. Possiamo paragonare una variabile a una porzione di una lavagna: è sempre possibile leggervi quanto scritto ed è possibile cancellare e riscrivervi dell'altro.

Per esempio, l'assegnamento del valore 9 a una variabile *v* può essere descritto in questo modo:

assegna v = 9

come dire: "v prende il valore 9" oppure "9 viene assegnato a v", cioè l'assegnazione è verso sinistra.

Ad una variabile può essere assegnato un dato costante, come nell'esempio precedente, oppure il risultato di un espressione:

calcola v = E

Tale istruzione indica che viene calcolato il valore dell'espressione E e che il risultato viene assegnato a v. Un'espressione è una formula (o regola di calcolo) che specifica sempre un valore (o risultato). Ogni espressione è composta da operatori e operandi. Gli operandi possono essere costanti (per esempio numeri), espressioni o variabili.

Gli operatori possono essere di tre tipi: aritmetici, di relazione e logici.

Gli **operatori aritmetici** sono:

+ per l'addizione
– per la sottrazione
***** per la moltiplicazione
/ per la divisione
^ per l'elevamento a potenza

Gli **operatori di relazione** (o *di confronto*) sono utilizzati per confrontare il contenuto di due variabili e sono indicati con i simboli:

= uguale
< minore di
<= minore o uguale di
> maggiore di
>= maggiore o uguale di
<> diverso

Gli **operatori logici** sono:

AND per il prodotto logico (congiunzione)
OR per la somma logica (disgiunzione)
NOT per la negazione
XOR per la disgiunzione esclusiva

7 Strumenti per la stesura di un algoritmo

Scrivere o comprendere algoritmi è un'operazione non semplice che comporta la conoscenza di regole ben precise e di linguaggi specifici. Nei progetti precedenti è stato usato un metodo descrittivo ricorrendo in larga misura al linguaggio comune in quanto si trattava di risolvere problemi abbastanza semplici. Occorre però fornire strumenti di lavoro validi per risolvere situazioni complesse e variamente articolate.
La conoscenza di linguaggi utili per descrivere un algoritmo è, pertanto, il primo e fondamentale compito di chi vuole conoscere l'arte di inventare e comunicare algoritmi.
Vengono descritti i due linguaggi più comunemente utilizzati: la pseudocodifica e i diagrammi a blocchi.
Come in tutti i linguaggi occorre definire l'alfabeto (i simboli ammessi dal linguaggio), le parole (le combinazioni dei simboli) e la sintassi (le regole che permettono di associare tra loro le parole in modo coerente).

Il linguaggio di pseudocodifica

Si intende per **pseudocodifica** (o *pseudolinguaggio* o *linguaggio di progetto*) la descrizione di un algoritmo ottenuta utilizzando termini e parole del linguaggio comune, ma applicando una serie di regole che permettono di organizzare un tipo di testo formalmente rigoroso e strettamente orientato alla stesura degli algoritmi; non si tratta di un vero linguaggio di programmazione, ma solo di un linguaggio sintetico composto di un vocabolario e di una sintassi molto ristretti.
Utilizzando la pseudocodifica, il programmatore esprime le proprie idee in una forma naturale e si avvale di frasi ed espressioni elementari della lingua italiana; ciò permette di concentrarsi sulla logica della soluzione del problema, senza vincolare il lavoro al particolare tipo di linguaggio di programmazione effettivamente utilizzato, e senza essere distratto dai vincoli formali richiesti nel lavoro di stesura del programma.

Di seguito vengono presentate le regole principali.

- Le parole chiave che aprono e chiudono il testo di un algoritmo sono

inizio fine

Ogni istruzione è indicata con una frase del linguaggio corrente e può contenere un'espressione di tipo aritmetico o logico. Le espressioni possibili sono quelle eseguibili da un elaboratore come indicato nelle pagine precedenti.

- Le istruzioni

immetti variabile

scrivi variabile
scrivi messaggio

vengono utilizzate per descrivere rispettivamente le operazioni di input (*immetti*) e output (*scrivi*) dei dati.

- Le istruzioni

assegna variabile = valore

calcola variabile = espressione

indicano operazioni di assegnazione a una variabile di un valore o del risultato di un'espressione.

- Le variabili, le costanti e, in generale, le risorse utilizzate nell'algoritmo vengono indicate da parole dette **identificatori**.

In pseudocodifica vengono utilizzate altre **parole chiave** che permettono di strutturare la logica della soluzione adottata e che corrispondono ai passaggi fondamentali dell'algoritmo. Le parole chiave utilizzate in pseudocodifica sono:

se, allora, altrimenti, fine se, esegui, finché, mentre, ripeti

e verranno illustrate nelle pagine successive.

Diagramma a blocchi

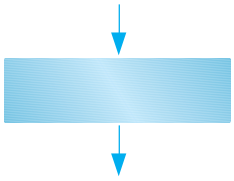
Il metodo del diagramma a blocchi consiste in una descrizione grafica dell'algoritmo; esso permette una visione immediata dell'intero procedimento e dell'ordine di esecuzione delle varie istruzioni.
I diagrammi a blocchi sono formati da simboli di forma diversa, ciascuna con un proprio significato; all'interno di ogni simbolo è presente un breve testo sintetico ma non ambiguo.

Linee orientate con frecce, che uniscono tra loro i vari simboli, indicano il flusso delle operazioni. I diagrammi vengono comunemente chiamati **diagrammi a blocchi** (o *diagrammi logici* o *flow-chart* o *diagrammi di flusso*).

I simboli

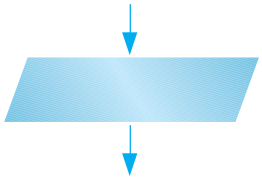
indicano il punto di partenza e quello di terminazione dell'algoritmo.
Da *Inizio* parte una sola freccia che raggiunge la prima istruzione e verso *Inizio* non arriva alcuna freccia. Verso *Fine* arrivano una o più frecce, ma da esso non ne parte nessuna.

Il simbolo



è detto di **elaborazione** e contiene al suo interno l'istruzione da eseguire (solitamente un'assegnazione di un valore o di un'espressione ad una variabile); può avere una o più frecce in entrata, ma una sola in uscita.

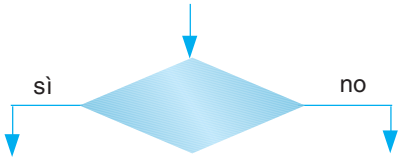
Il simbolo



viene utilizzato per rappresentare operazioni di immissione di dati (**input**) o di emissione di dati (**output**).

In mancanza di indicazioni specifiche si assume che l'immissione dei dati avvenga da tastiera (**standard input**) e che i dati di output siano presentati sullo schermo (**standard output**). Il simbolo è associato a una o più frecce di entrata e ad una sola freccia di uscita.

Il simbolo



viene detto **simbolo di decisione** e, nella maggior parte dei casi, serve per rappresentare un'operazione di confronto tra due dati. Il simbolo di decisione è usato per stabilire se una proposizione è vera o falsa e in corrispondenza delle frecce in uscita si trovano indicazioni del tipo sì/no, vero/falso, V/F.
Viene così definito il valore di un'espressione di tipo logico, cioè di una variabile a due valori a uno dei quali corrisponde il significato di condizione verificata, vero, e all'altro quella di condizione non verificata, falso.
Il rombo ha un punto di entrata e due punti di uscita.

Infine, è possibile, all'interno di un diagramma a blocchi, inserire alcuni **commenti** illustrativi che servono per dare maggiore chiarezza allo schema. Allo scopo si usa il simbolo

PROGETTO 5

Calcolo dell'area di un triangolo.

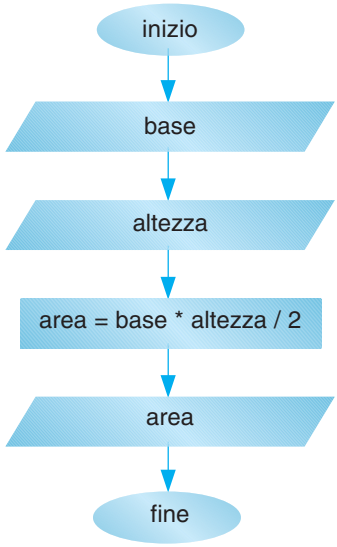
Per calcolare l'area del triangolo occorre conoscere le misure della base e dell'altezza; l'esecutore deve quindi chiedere all'utente tali misure, attendere che l'utente comunichi tali misure e leggere i valori assegnandoli ad altrettante variabili. Dopo aver calcolato l'area, l'esecutore deve provvedere a scrivere (comunicare all'utente) il risultato.

Dati di input: base e altezza del triangolo.
Dati di output: area del triangolo.

Algoritmo in pseudocodifica

```
inizio
    immetti base
    immetti altezza
    calcola area = base * altezza / 2
    scrivi area
fine
```

Diagramma a blocchi



Per meglio comprendere il senso di quanto scritto nell'algoritmo si osservi che il programmatore scrive una sequenza di comandi che sono espressi da verbi in modo imperativo (per esempio: *immetti*, *assegna*, *scrivi*) e che sono rivolti all'esecutore (computer).

L'esecutore, oltre a procedere nello svolgimento dei calcoli (per esempio: area = base * altezza / 2), interagisce con l'utente in tre modi:

- riceve dei dati (i dati di input),
- comunica dei risultati (i dati di output).

MATERIALI ON LINE

2. Diagrammi a blocchi con Word

LA MACCHINA DI TURING

Come visto nel capitolo precedente, il termine **automa** indica un dispositivo che sia in grado di eseguire da solo, cioè in modo automatico, senza l'intervento di una persona, una sequenza di azioni stabilite in precedenza, dotato di meccanismi per acquisire elementi dall'esterno e produrre elementi verso l'esterno: durante il suo funzionamento inoltre, all'interno, può assumere stati diversi tra loro. Dopo aver descritto i concetti di *automa* e di *algoritmo*, si presentano ora due aspetti da risolvere:

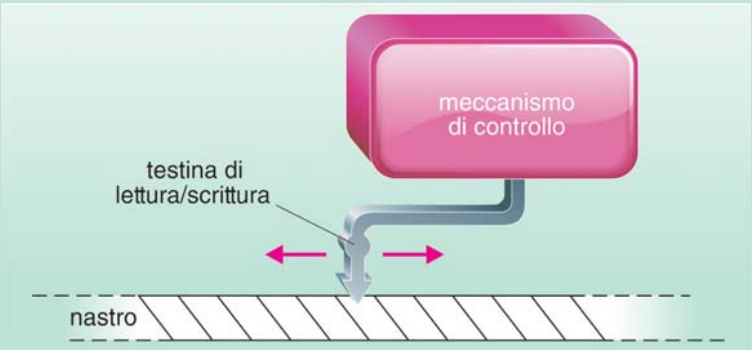
a) stabilire se esiste sempre un procedimento risolutivo da formalizzare con un algoritmo in corrispondenza di un determinato problema;

b) stabilire le caratteristiche dell'automa in grado di eseguire l'algoritmo.

La risposta al primo aspetto verrà data di seguito con la *Tesi di Church-Turing*.

Alla seconda esigenza risponde il modello di macchina astratta proposto nel 1936 dal matematico inglese **Alan M. Turing** (1912-1954), che prende perciò il nome di **Macchina di Turing** (MdT).
Il particolare automa ideato da Turing fa riferimento alla comune attività mentale quando è impegnata nella risoluzione di algoritmi di calcolo: di solito si usano un foglio e una penna con la quale segnare sul foglio i dati e i simboli delle operazioni; il lavoro è controllato dalla mente umana, per la quale la carta costituisce il supporto di memoria (il foglio per i calcoli, oppure il foglio del libro che contiene le regole di calcolo).

La **Macchina di Turing**, in modo analogo, è costituita da un meccanismo di controllo, da un nastro di lunghezza infinita nei due sensi e da una testina di lettura/scrittura che può effettuare due tipi di movimento a sinistra o a destra.



Il nastro è diviso in celle, in ciascuna delle quali può essere letto o scritto un simbolo appartenente all'alfabeto dei simboli trattati dalla macchina.

Durante il suo funzionamento la macchina evolve da una configurazione all'altra, cioè da uno **stato** all'altro: in corrispondenza del simbolo letto sul nastro e dello stato in cui si trova, viene determinato il simbolo che viene scritto sul nastro, lo stato successivo della macchina e il movimento della testina.

All'inizio del processo di evoluzione, sul nastro si trova la sequenza dei simboli di input e al verificarsi della terminazione si trova l'output del procedimento eseguito.

La testina di lettura/scrittura può accedere ad una cella per volta, e si può spostare a destra o a sinistra.

Da un punto di vista formale la MdT viene definita come una sestupla

$$MdT = (A, I, S, s_o, F, d)$$

dove:

- A è l'**alfabeto** dei simboli utilizzati dalla macchina; l'alfabeto comprende anche il simbolo *blank* che corrisponde alla cella vuota, cioè alla parte di nastro non utilizzato, e serve per indicare l'assenza di simboli.
- I è l'insieme dei **simboli di input** (sottoinsieme di A) che possono essere letti dal nastro.
- S è l'insieme degli **stati** della macchina.
- s_0 è lo **stato iniziale** della macchina.
- F è l'insieme degli **stati finali** della macchina (sottoinsieme di S).
- d indica la **funzione di transizione**, che associa ad ogni coppia (simbolo di input, stato) una terna (stato, simbolo di output, movimento della testina):

$$(i_t, s_{t-1}) \rightarrow (s_t, u_t, m_t)$$

e sta quindi ad indicare il simbolo che viene scritto sul nastro, lo stato successivo e il movimento della testina, quando la macchina, trovandosi in un determinato stato, legge un determinato simbolo di input dal nastro.

Nella **matrice di transizione**, che schematizza con una tabella la *funzione di transizione*, sono indicati sulle righe i simboli che vengono letti dal nastro (sottoinsieme dell'alfabeto A) e sulle colonne gli stati della macchina: all'incrocio tra una riga e una colonna vengono specificati lo stato successivo, il simbolo che viene scritto in output sul nastro e il movimento della testina (separati da una virgola):

stati / input	s_1	s_2	s_3
i_1	s_{11},u_{11},m_{11}	s_{12},u_{12},m_{12}	s_{13},u_{13},m_{13}
i_2	s_{21},u_{21},m_{21}	s_{22},u_{22},m_{22}	s_{23},u_{23},m_{23}
i_3	s_{31},u_{31},m_{31}	s_{32},u_{32},m_{32}	s_{33},u_{33},m_{33}
i_4	s_{41},u_{41},m_{41}	s_{42},u_{42},m_{42}	s_{43},u_{43},m_{43}

In ogni istante la MdT assume una configurazione che può essere definita con la quintupla:

- simbolo letto
- stato
- nuovo stato
- simbolo scritto
- movimento della testina.

Il passaggio da una configurazione a quella dell'istante successivo si chiama **mossa** della MdT.
Vediamo ora l'esempio di un problema risolto mediante una MdT.

PROGETTO 6

Calcolare il quoziente e il resto della divisione di un numero intero per 2.

L'alfabeto dei simboli è composto dalle 10 cifre decimali con l'aggiunta del simbolo *b*, che indica il *blank*, e il simbolo # che indica la situazione di fine input sul nastro:

A = (b, #, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

L'insieme dei simboli di input è costituito dalle 10 cifre decimali e dal simbolo #.
L'insieme degli stati è composto da:

S = (R0, R1, FINE)

Lo stato iniziale è R0, lo stato finale è FINE.

All'inizio sul nastro si trovano scritte le cifre che compongono il dividendo; alla fine dell'elaborazione si possono trovare le cifre del quoziente e, immediatamente alla loro destra, quella del resto (che può essere ovviamente 0 oppure 1).

La funzione di transizione opera in questo modo:

(R0, cifra pari) → (R0, cifra pari:2, DESTRA)
(R1, cifra pari) → (R0, (cifra pari+10):2, DESTRA)

(R0, cifra dispari) → (R1, cifra dispari:2, DESTRA)
(R1, cifra dispari) → (R1, (cifra dispari+10):2, DESTRA)

(R1, #) → (FINE, 1, DESTRA)
(R0, #) → (FINE, 0, DESTRA).

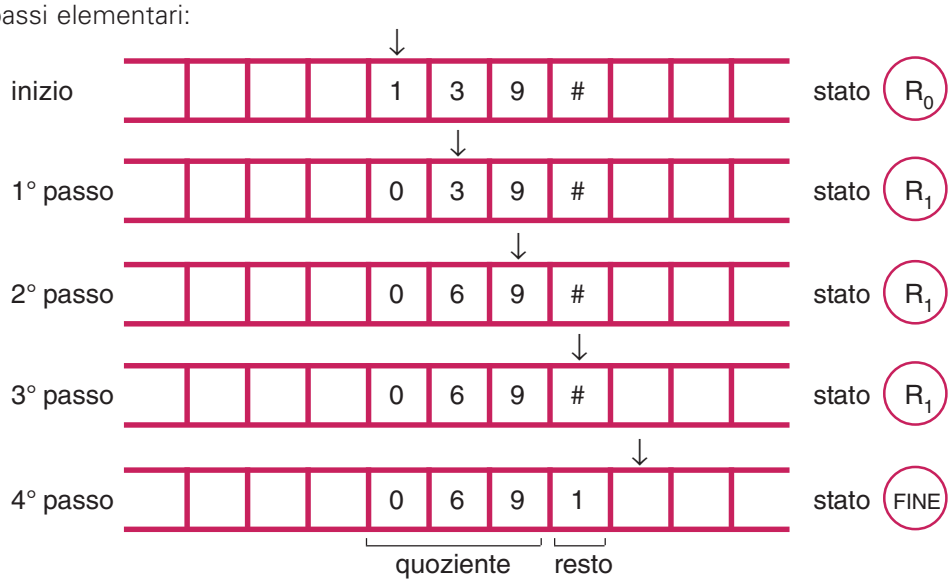
La MdT opera scandendo le cifre del dividendo a partire da quella più significativa (più a sinistra) e scrivendo al posto di ogni cifra il quoziente della cifra diviso 2, tenendo conto dell'eventuale resto precedente. Gli stati R0 e R1 servono a ricordare rispettivamente le situazioni con resto uguale a 0 oppure a 1.

La funzione di transizione può essere descritta con la seguente tabella:

	R0	R1
#	FINE,0,D	FINE,1,D
0	R0,0,D	R0,5,D
1	R1,0,D	R1,5,D
2	R0,1,D	R0,6,D
3	R1,1,D	R1,6,D
4	R0,2,D	R0,7,D
5	R1,2,D	R1,7,D
6	R0,3,D	R0,8,D
7	R1,3,D	R1,8,D
8	R0,4,D	R0,9,D
9	R1,4,D	R1,9,D

La lettera *D* indica il movimento della testina verso destra e il simbolo # sta ad indicare la fine dell'input sul nastro.

Il funzionamento della MdT così definita può essere descritto secondo questa sequenza di passi elementari:



L'uso della Macchina di Turing introduce in modo intuitivo l'idea che un problema sia risolubile attraverso un procedimento quando sia possibile costruire una MdT in grado di eseguire un algoritmo che descrive, attraverso passi elementari, il procedimento risolutivo. D'altra parte appare intuitiva anche una coincidenza tra il concetto di algoritmo e il concetto di MdT, nel senso che l'algoritmo è il procedimento risolto dalla MdT astratta.
La risposta all'esigenza presentata (e rimasta aperta) all'inizio, di stabilire quando un problema è risolubile attraverso un procedimento, viene fornita attraverso la sistemazione teorica delle idee intuitive sopra descritte.
Nel seguito si userà il termine **computabile** come sinonimo di calcolabile o risolubile. Il termine **funzione computabile** indica un procedimento che trasforma i dati iniziali in dati finali.
La **Tesi di Church-Turing**, elaborata da Turing e dallo scienziato americano **Alonzo Church** (1903-1995), risolve a livello teorico l'esigenza posta e afferma che:

> La classe delle funzioni computabili, secondo il concetto intuitivo di algoritmo, coincide con la classe delle funzioni Turing-computabili, cioè computabili con una MdT.

Sulla base della Tesi di Church-Turing possiamo quindi individuare all'interno dell'insieme di tutti i possibili problemi il sottoinsieme dei problemi che sono computabili: sono quelli ai quali può essere associata una MdT che risolve il problema.
Inoltre l'algoritmo risolutivo di un problema computabile può essere identificato con la sequenza delle azioni che vengono eseguite da una MdT.

AUTOVERIFICA

Domande da 1 a 5 pag. 102
Problemi da 1 a 7 pag. 104

MATERIALI ON LINE

3. Esempi di Macchina di Turing
4. Teoria della complessità algoritmica

8 Le strutture di controllo

Dal punto di vista informatico la stesura di un algoritmo è uno dei passi necessari all'interno di un lavoro più complesso che costituisce la realizzazione di un programma.
La programmazione è infatti un'attività che può essere suddivisa in quattro fasi:

- definizione del problema, dei dati di input e di output;
- organizzazione dell'algoritmo risolutivo;
- stesura del programma, cioè la traduzione dell'algoritmo nel linguaggio di programmazione;
- prove di esecuzione del programma.

Poiché la stesura dell'algoritmo risolutivo è una delle fasi fondamentali del lavoro di programmazione, si tratta di definire un insieme di regole che devono essere seguite per una corretta organizzazione del lavoro e per il raggiungimento di un buon livello di qualità dei programmi.

Per individuare i modelli organizzativi con cui si possono strutturare gli algoritmi, per mettere a fuoco le strutture fondamentali che compongono un programma e per mostrare gli aspetti principali dei formalismi usati per la descrizione di un algoritmo, viene riproposto, a titolo di esempio, il problema del lavoro di controllo del bibliotecario, presentato in precedenza nel Progetto 2, scrivendo l'algoritmo risolutivo con le tecniche della pseudocodifica e dei diagrammi a blocchi.

PROGETTO 7

L'addetto alla Biblioteca comunale, disponendo dell'elenco degli utenti che hanno preso in prestito i libri, con cognome, nome e data prestito, deve mandare un avviso agli utenti per i quali sono trascorsi più di 30 giorni dal prestito (si suppone che l'esecutore sia in grado di calcolare i giorni intercorrenti tra due date facendone la differenza).

Il bibliotecario deve procurarsi la lista delle persone che hanno effettuato prestiti di libri. Scorrendo l'elenco, per ciascun utente deve controllare se sono trascorsi più di 30 giorni dal prestito. In caso affermativo deve annotare il numero telefonico accanto al cognome, altrimenti deve mettere solo un segno di spunta per ricordare di aver controllato l'utente.

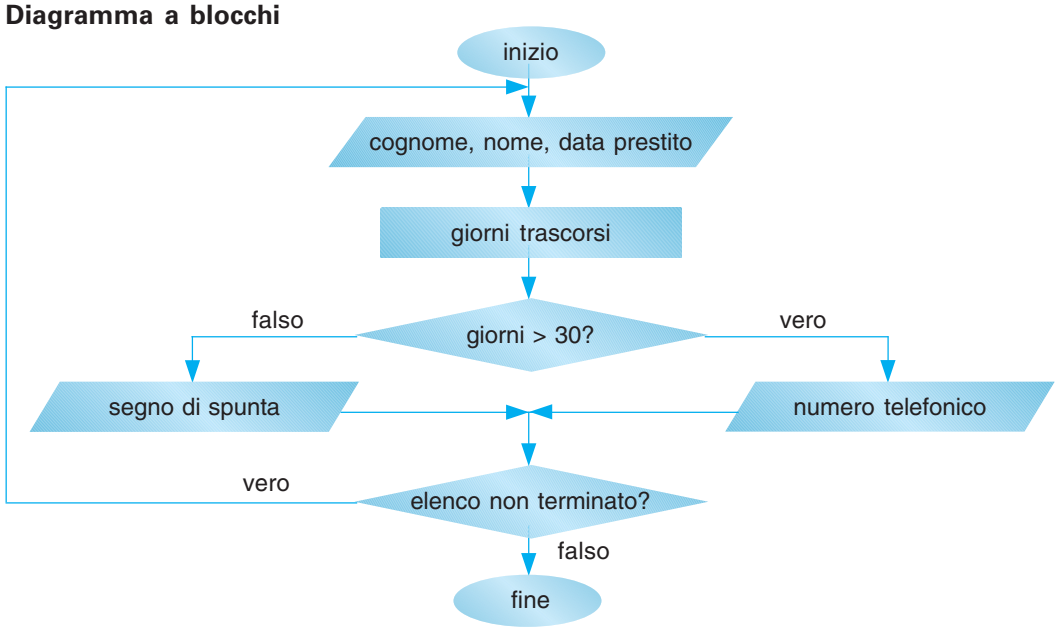
Dati di input: l'elenco delle persone con cognome, nome, data del prestito, numeri telefonici degli utenti.

Dati di output: l'elenco degli utenti con l'indicazione per ciascuno dell'esito del controllo.

Algoritmo in pseudocodifica

```
inizio
    esegui
        leggi cognome, nome, data del prestito
        calcola il numero di giorni trascorsi
        se numero è superiore a 30
        allora
            scrivi il numero telefonico accanto al cognome
        altrimenti
            metti un segno di spunta accanto al cognome
        fine se
    ripeti mentre l'elenco non è terminato
fine
```

Diagramma a blocchi



Osservando con attenzione l'algoritmo precedente, si può notare che le istruzioni sono organizzate secondo schemi classificabili nel modo seguente.

1) Istruzioni organizzate in **sequenza**:

. . . . . .
    leggi cognome, nome, data del prestito
    calcola il numero di giorni trascorsi
. . . . . .

vale a dire istruzioni che devono essere eseguite una dopo l'altra secondo l'ordine con cui sono state scritte.

2) Istruzioni che vengono eseguite in **alternativa** con altre:

    se numero è superiore a 30
    allora
        scrivi il numero telefonico accanto al cognome
    altrimenti
        metti un segno di spunta accanto al cognome
    fine se

l'esecutore deve fare una scelta tra un certo gruppo di istruzioni e un altro gruppo a seconda di quello che succede in quel momento durante l'elaborazione.

3) Istruzioni che devono essere eseguite in **ripetizione**:

esegui
. . . . . .
. . . . . .
ripeti mentre l'elenco non è terminato

Le istruzioni comprese tra *esegui* e *ripeti mentre* devono essere eseguite più volte, tante quante sono gli utenti dell'elenco da controllare.
Per quanto visto sopra, si può dire che è possibile individuare all'interno di un algoritmo alcune strutture tipiche.

Nel progetto precedente sono state messe in risalto tre strutture, vale a dire sono stati riconosciuti degli schemi particolari secondo cui sono organizzate le istruzioni all'interno dell'algoritmo. Si tratta ora di stabilire se le strutture riconosciute nell'esempio sono un caso particolare strettamente legato al problema esaminato, oppure se quanto è stato enucleato trova una sua giustificazione e una conferma nella teoria generale della programmazione.
Questi ragionamenti sono stati affrontati da numerosi studi teorici che permettono di rispondere alle domande emerse nelle righe precedenti: qui viene semplicemente illustrato l'enunciato fondamentale, che sta alla base di un corretto approccio al lavoro di creazione degli algoritmi.

Qualsiasi algoritmo appropriato può essere scritto utilizzando soltanto tre strutture di base: **sequenza, alternativa, ripetizione.** Questi tre modelli organizzativi di base si chiamano **strutture di controllo**, perché servono a controllare il percorso all'interno del procedimento risolutivo per ottenere i risultati desiderati.

La struttura alternativa si chiama anche **selezione** o **struttura condizionale**, e la struttura di ripetizione viene indicata anche con il termine **iterazione**.
Riassumendo, se si utilizza la pseudocodifica, le tre strutture di controllo fondamentali sono rappresentate nel seguente modo:

Sequenza

```
. . . . . . . . .
istruzione-1
istruzione-2
istruzione-3
. . . . . . . . .
```

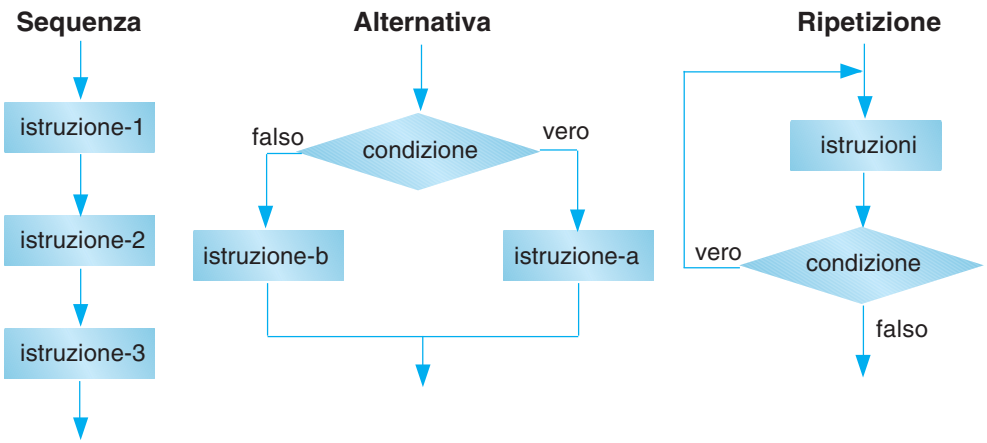
Alternativa

```
se condizione
    allora
        istruzione-a
    altrimenti
        istruzione-b
fine se
```

Ripetizione

```
esegui
        istruzioni
ripeti mentre condizione
```

Se si utilizzano i diagrammi a blocchi le strutture sono rappresentate dai seguenti schemi:



Per quanto riguarda l'uso delle strutture di controllo per la stesura degli algoritmi, il riferimento classico è il **Teorema di Böhm-Jacopini** (1966) che può essere così formulato:

Un qualsiasi algoritmo può essere espresso usando esclusivamente le strutture di sequenza, di selezione e di iterazione.

MATERIALI ON LINE

5. Approfondimento sul teorema di Böhm-Jacopini

9 La struttura di alternativa

La struttura alternativa o **selezione** viene rappresentata secondo lo schema:

se condizione
allora
    istruzione-a
altrimenti
    istruzione-b
fine se

Se la *condizione* è vera, viene eseguita l'*istruzione-a*, altrimenti viene eseguita l'*istruzione-b*. *Istruzione-a* e *istruzione-b* possono indicare, come accade nella maggior parte dei casi, non una sola istruzione, ma un gruppo di istruzioni.
La condizione è un'espressione booleana di cui viene valutata la verità: vengono quindi utilizzati i segni del confronto: <, >, =, >=, <=, <>, e gli operatori booleani AND, NOT, OR, XOR per costruire espressioni logiche combinando tra loro più condizioni.
Il progetto che segue mostra un algoritmo risolutivo con la struttura di alternativa.

PROGETTO 8

Dati due numeri disporli in ordine crescente.

Dati di input: due numeri, primo e secondo.

Dati di output: i due numeri ordinati, minore e maggiore.

Algoritmo in pseudocodifica

```
inizio
        immetti primo, secondo
        se primo <= secondo
        allora
                assegna minore = primo
                assegna maggiore = secondo
        altrimenti
                assegna minore = secondo
                assegna maggiore = primo
        fine se
        scrivi minore, maggiore
fine
```

Nella struttura di alternativa viene controllata la condizione indicata dopo la parola *se*: se la condizione è vera viene eseguita l'istruzione o il gruppo di istruzioni scritte sotto la parola *allora*; se la condizione è falsa vengono eseguite le istruzioni sotto la parola *altrimenti*.
Si noti che le istruzioni da eseguire quando la condizione è vera sono disposte in modo rientrato (si dice anche *indentato*) rispetto ad *allora*; inoltre, l'insieme di operazioni che devono essere eseguite, se si prende la strada di *allora*, termina quando termina l'indentazione; altrettanto vale per le istruzioni sotto *altrimenti*.
L'uso di una corretta indentazione, quando si descrivono gli algoritmi con la pseudocodifica, consente di facilitare la lettura e la comprensione del procedimento risolutivo.

LA STRUTTURA DI SCELTA MULTIPLA

L'alternativa a due vie non sempre risponde alle necessità di risolvere situazioni più complesse: è stato quindi introdotto lo schema della **scelta multipla** (o *struttura di selezione multipla*).

In pseudocodifica tale struttura può essere rappresentata nel modo seguente:

```
caso di variabile =
lista valori-1
        istruzioni-1
lista valori-2
        istruzioni-2
. . .
lista valori-n
        istruzioni-n
altrimenti
        istruzioni
fine caso
```

La presenza della struttura di selezione multipla all'interno di un algoritmo determina al momento dell'esecuzione le seguenti azioni: se il valore della *variabile* è presente in una delle liste di valori, allora viene eseguito il blocco di istruzioni corrispondente; successivamente si procede con l'esecuzione della prima istruzione che segue la struttura *caso di*; in caso contrario viene eseguito il blocco di istruzioni indicato in modo indentato sotto ad *altrimenti*, e poi si passa all'istruzione successiva.
La variabile da controllare si chiama anche **selettore**.

La struttura di selezione multipla è una struttura derivata dalla struttura fondamentale di selezione binaria, infatti può essere rappresentata utilizzando una successione di controlli nidificati del tipo *se.... allora ... altrimenti ... fine se*.

```
se variabile = valore-1
allora
        istruzioni-1
altrimenti      se variabile = valore-2
                allora
                        istruzioni-2
                altrimenti      ...............................
                                se variabile = valore-n
                                allora
                                        istruzioni-n
                                altrimenti
                                        istruzioni
                                fine se
                fine se
fine se
```

Le due forme sono funzionalmente equivalenti, cioè producono gli stessi output con gli stessi input, ma la prima forma, con *caso di*, risulta più compatta e rende il testo dell'algoritmo più leggibile.

PROGETTO 9

Un volo aereo low-cost per sola andata con destinazione Londra ha prezzi diversi a seconda dei mesi dell'anno e precisamente:

da gennaio a giugno	**euro 26**
luglio e agosto	**euro 54**
da settembre a novembre	**euro 21**
dicembre	**euro 48.**

Fornita la data di partenza, si vuole ottenere il costo del volo.

La data inserita è nella forma gg/mm/aaaa: da essa si può estrarre il valore numerico del mese. Con una selezione multipla, si esaminano i diversi casi: a seconda del numero del mese si scrive il prezzo del volo, secondo la tabella contenuta nel testo del problema.

Dati di input:

data della partenza.

Dati di output:

costo del volo.

Algoritmo in pseudocodifica

```
inizio
        immetti dataPartenza
        estrai mese
        caso di mese =
        1 .. 6
                assegna costoVolo = 26
                scrivi costoVolo
        7, 8
                assegna costoVolo = 54
                scrivi costoVolo
        9 .. 11
                assegna costoVolo = 21
                scrivi costoVolo
        12
                assegna costoVolo = 48
                scrivi costoVolo
        altrimenti
                scrivi "data errata"
        fine caso
fine
```

Si noti che la lista di valori con cui viene valutata la variabile *mese* può essere espressa in modi diversi: indicando un singolo valore, oppure valori separati con una virgola (per esempio: 7, 8), oppure un intervallo di valori possibili (per esempio, 1 .. 6).

10 Logica iterativa

La ripetizione

La **ripetizione**, come terza struttura di controllo fondamentale della programmazione, si rappresenta con la schema:

esegui
    istruzioni
ripeti mentre condizione

La *condizione* deve essere un'espressione che rappresenta un valore *Vero* o *Falso*.
Le istruzioni comprese tra *esegui* e *ripeti* vengono eseguite una prima volta, dopo di che viene verificata la condizione scritta dopo *mentre*: se la condizione risulta falsa si prosegue con l'istruzione successiva, altrimenti si ripete l'esecuzione delle istruzioni a partire dalla prima istruzione dopo *esegui*.

PROGETTO 10

Dato un elenco di prodotti, con l'indicazione per ciascuno di essi del nome e del prezzo, scrivere il nome dei prodotti che hanno un prezzo superiore a un valore prefissato e fornito all'inizio del procedimento.

Per ogni prodotto dell'elenco si devono richiedere il nome e il prezzo; si deve controllare se il prezzo è superiore al valore fornito e, in questo caso, deve essere scritto il nome del prodotto. Il controllo viene ripetuto mentre l'elenco non è finito.

Dati di input:
valore minimo prefissato per i prezzi
nome del prodotto
prezzo del prodotto.

Dati di output:
nome dei prodotti con prezzo superiore.

Algoritmo in pseudocodifica

```
inizio
    immetti valoreMinimo
    esegui
        immetti nome, prezzo
        se prezzo > valoreMinimo
        allora
            scrivi nome
        fine se
    ripeti mentre l'elenco non è finito
fine
```

Ripetizione precondizionale

È possibile definire anche **ripetizioni precondizionali**, cioè strutture che prima verificano se le istruzioni si devono eseguire e, solo in caso affermativo, procedono all'iterazione delle istruzioni stesse.

In pseudocodifica la ripetizione precondizionale si rappresenta con:

esegui mentre condizione
    istruzioni
ripeti

Il segmento di algoritmo precedente significa che si deve prima controllare la condizione: se questa è verificata si procede all'esecuzione delle istruzioni comprese tra *esegui mentre* e *ripeti*. Le istruzioni vengono ripetute quando la condizione è verificata, la ripetizione si arresta quando la condizione diventa falsa.
Se la condizione risulta subito falsa, le istruzioni non vengono eseguite nemmeno una volta.

PROGETTO 11

Calcolo del prodotto tra interi utilizzando la sola operazione di somma.

Presi in considerazione due numeri interi, si tratta di sommare il primo numero con se stesso per un numero di volte pari al secondo numero. Per sapere quante volte deve essere eseguita l'operazione di somma e per sapere quando fermarsi, ogni volta che viene fatta la somma si decrementa di 1 il valore del secondo numero e si interrompe la ripetizione della somma quando questo diventa 0.

Dati di input:
i due numeri da moltiplicare, primo e secondo fattore.
Dati di output:
il prodotto dei due numeri.

Algoritmo in pseudocodifica

```
inizio
    immetti primo, secondo
    assegna prodotto = 0
    esegui mentre secondo <> 0
        calcola prodotto = prodotto + primo
        calcola secondo = secondo – 1
    ripeti
    scrivi prodotto
fine
```

La ripetizione con contatore

La struttura di **ripetizione con contatore** (o **enumerativa**) è un'altra struttura derivata dalla ripetizione. Questa nuova struttura permette di ripetere un certo gruppo di istruzioni, non in base al valore di verità di una condizione, ma in base a un numero di volte prefissato.

Se il numero di ripetizioni che si vogliono eseguire è noto a priori, è possibile organizzare una ripetizione precondizionale dove la condizione di arresto è indicata dal valore massimo che può raggiungere una variabile utilizzata come contatore.

La rappresentazione della ripetizione enumerativa avviene in pseudocodifica nel modo seguente:

per variabile-contatore **da** iniziale **a** finale
    istruzioni
ripeti

Quando, all'interno di un algoritmo, viene incontrata tale struttura, la *variabile-contatore*, che serve da controllo per la ripetizione, assume il valore indicato con *iniziale*. Si procede poi all'esecuzione del gruppo di istruzioni indicate sotto la parola *per* ed indentate rispetto ad essa. L'esecuzione di queste istruzioni viene ripetuta e, ad ogni ripetizione, viene incrementato il valore della *variabile-contatore*. Mentre tale valore è compreso nell'intervallo che va dal valore *iniziale* al valore *finale*, le istruzioni vengono rieseguite; la ripetizione si arresta quando la *variabile-contatore* diventa maggiore di *finale*.

Una variante rispetto alla versione precedente consiste nell'inserimento dell'incremento (passo) che deve subire la *variabile-contatore* ad ogni ripetizione del gruppo di istruzioni.

Tale variante in pseudocodifica si indica con:

per variabile-contatore **da** iniziale **a** finale **passo** incremento
    istruzioni
ripeti

In questo caso, ad ogni esecuzione delle istruzioni la *variabile-contatore* viene aumentata di un valore pari ad *incremento*.
In mancanza di indicazioni sull'incremento (per *default*, come si dice con linguaggio informatico) si sottintende che l'incremento sia unitario.

La struttura di ripetizione con contatore è derivata dalla struttura fondamentale di ripetizione, nel senso che il controllo della ripetizione delle istruzioni mediante un contatore può essere fatto in modo equivalente usando una struttura *esegui mentre... ripeti.*

Le due strutture riportate sotto sono funzionalmente equivalenti:

ripetizione con contatore

per i **da** 1 **a** n
    istruzioni
ripeti

ripetizione precondizionale

assegna i = 1
esegui mentre i <= n
    istruzioni
    calcola i = i + 1
ripeti

La struttura di ripetizione con contatore, nel caso di ripetizione con un numero prefissato di volte, risulta più compatta e più semplice da rappresentare, in quanto dentro la frase *per* sono rappresentati l'assegnazione del valore iniziale del contatore e l'incremento del contatore stesso.

PROGETTO 12

Dati gli importi delle spese domestiche per ciascun mese dell'anno, calcolare la spesa totale annuale.

Il numero degli importi è prefissato a 12, quanti sono i mesi dell'anno. All'inizio viene azzerato il totale dell'anno; con una ripetizione con contatore si acquisiscono i valori delle spese di ciascun mese e si sommano al totale. Alla fine della ripetizione viene scritto il totale come risultato dell'elaborazione.

Dati di input:
spesa di ciascun mese dell'anno.

Dati di output:
totale delle spese nell'anno.

Algoritmo in pseudocodifica

```
inizio
        assegna totale = 0
        per mese da 1 a 12
                immetti spesaMese
                calcola totale = totale + spesaMese
        ripeti
        scrivi totale
fine
```

AUTOVERIFICA

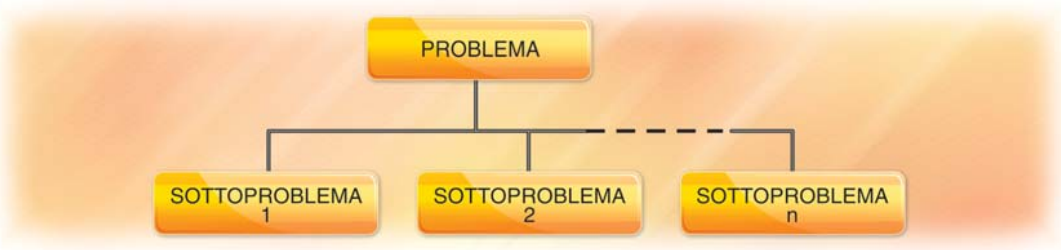
Domande da 6 a 8 pag. 102-103
Problemi da 8 a 15 pag. 104-105

MATERIALI ON LINE

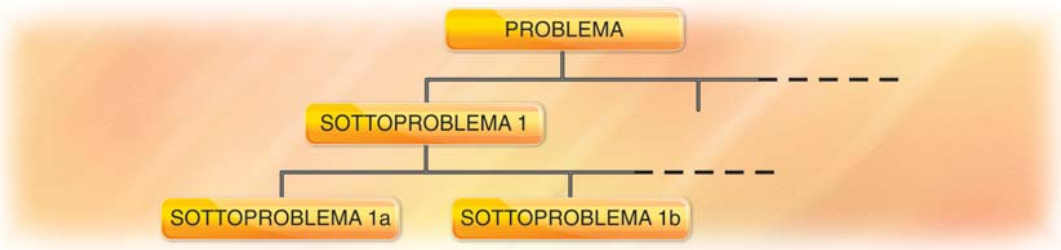
6. Diagrammi a blocchi dei progetti
7. Osservazioni sulle strutture di alternativa e di ripetizione
8. Ripetizione precondizionale per il controllo di fine input
9. Algoritmi diversi per la soluzione dello stesso problema

11 Sviluppo top-down

Nei paragrafi precedenti sono stati presentati problemi relativamente semplici per l'inserimento di valori o la visualizzazione di alcuni dati.
Quando il livello di difficoltà dei problemi aumenta, ci troviamo di fronte a situazioni in cui è difficile controllare, a un livello di dettaglio basso, i vari aspetti del problema. Si ricorre allora al cosiddetto metodo dei **raffinamenti successivi**.
Il problema principale, che deve essere risolto tramite un programma, viene suddiviso in sottoproblemi più semplici di dimensioni più ridotte che avranno minore complessità.
Si realizza così uno schema logico del tipo indicato nella figura seguente.



Ciascun sottoproblema viene affrontato separatamente e, se presenta ancora un grado di complessità notevole, si procede, come prima, attraverso la suddivisione del problema in sottoproblemi per diminuire ulteriormente il livello di astrazione.



Tale modo di procedere viene detto **sviluppo top-down** in quanto, nel realizzare gli schemi visti prima, si procede dall'alto verso il basso: ad ogni passaggio al livello inferiore si ottiene una suddivisione in sottoproblemi più semplici.

Si tratta in sostanza di definire quali sono le parti fondamentali di cui si compone l'applicazione software (*top*), e procedere poi al dettaglio di ogni singola parte (*down*). Ogni parte corrisponde ad un modulo che svolge una specifica funzione all'interno del problema, cioè l'applicazione deve essere scomposta in **moduli funzionalmente indipendenti**. Questo modo di operare consente di costruire applicazioni software come insiemi di moduli, con il vantaggio di avere sempre sotto controllo il funzionamento del programma, e nello stesso tempo poter avere in ogni istante una visione dettagliata di ogni singolo modulo.

Viene facilitato il lavoro di **manutenzione e aggiornamento del software**, perché si può intervenire con modifiche o correzioni su un solo modulo, avendo nel contempo cognizione di quello che fa l'intero programma.
Inoltre alcuni moduli, essendo funzionalmente indipendenti, possono essere riutilizzati, senza bisogno di molte modifiche, all'interno di altre applicazioni.

Un'altra esigenza che viene risolta dalla metodologia di sviluppo top-down è rappresentata dalla possibilità di riutilizzare sottoprogrammi provenienti da altri progetti: in questi casi potrebbe accadere che le variabili del programma principale non abbiano lo stesso nome delle variabili che devono corrispondere nel sottoprogramma.
Questo non è importante, in quanto al momento della chiamata di un sottoprogramma, si possono passare i **parametri**, che sono indicati nell'intestazione del sottoprogramma.
L'operazione, con la quale il programma chiamante manda i valori al sottoprogramma, assegnandoli ai parametri, si chiama **passaggio di parametri**.

L'uso dei parametri risponde quindi all'esigenza di costruire sottoprogrammi che possono essere eseguiti più volte all'interno dello stesso progetto con dati diversi; inoltre essi diventano moduli software che possono essere utilizzati in altri programmi.
Le variabili indicate nell'intestazione del sottoprogramma si chiamano **parametri formali**; le variabili che forniscono i valori ai parametri si chiamano **parametri attuali**.

PROGETTO 13

Ordinamento crescente di tre numeri.

L'utente inserisce tre valori numerici. L'algoritmo deve ordinarli dal più piccolo al più grande.

Dati di input:
i tre numeri.

Dati di output:
i tre numeri ordinati.

Analisi del problema

L'algoritmo risolutivo confronta il primo numero con il secondo, scambiandoli tra loro se non sono in ordine crescente; successivamente confronta il primo con il terzo, scambiandoli tra loro se non sono in ordine crescente, e infine il secondo con il terzo, con eventuale scambio.

Algoritmo in pseudocodifica

```
inizio
    immetti a, b, c
    se a > b
    allora                  scambia a con b
        assegna temp = a
        assegna a = b
        assegna b = temp
    fine se
    se a > c
    allora                  scambia a con c
        assegna temp = a
        assegna a = c
        assegna c = temp
    fine se
    se b > c
    allora                  scambia b con c
        assegna temp = b
        assegna b = c
        assegna c = temp
    fine se
    scrivi a, b, c
fine
```

Al termine dell'algoritmo la variabile *a* contiene il numero più piccolo e la variabile *c* il numero più grande. Si può notare che l'operazione di controllo e scambio del contenuto di due variabili viene eseguita più volte nel corso del programma su variabili diverse (prima *a* e *b*, poi *a* e *c* infine *b* con *c*), ma utilizzando istruzioni identiche.
È conveniente raggruppare quindi queste istruzioni, che vengono usate tre volte, in una singola procedura che verrà chiamata tre volte.

Sottoprogramma Ordina (x,y)

```
inizio
    se x > y
    allora
        assegna temp = x
        assegna x = y
        assegna y = temp
    fine se
fine
```

Algoritmo principale

```
inizio
    Ordina (a,b)
    Ordina (a,c)
    Ordina (b,c)
fine
```

L'algoritmo chiama tre volte il sottoprogramma *Ordina*, chiedendo di operare su tre differenti coppie di dati. Ogni chiamata causa l'esecuzione del sottoprogramma, al termine della quale la coppia di dati forniti risulta ordinata.

Lo sviluppo top-down e l'uso delle strutture di controllo (sequenza, selezione e iterazione) sono alla base della metodologia di sviluppo del software detta **programmazione strutturata**.

Questa metodologia permette di risolvere i problemi scomponendoli in sottoproblemi che si ritiene siano più facili da risolvere. Si riesce a organizzare e semplificare in maniera migliore lo sviluppo e la creazione del programma, perché viene ridotta la complessità del problema cercando di risolverlo modulo per modulo.

La programmazione strutturata indica che bisogna procedere attraverso astrazioni, individuando cioè le parti principali che compongono un problema e analizzandole singolarmente.

Programmare usando le tecniche introdotte dalla programmazione strutturata significa avere a disposizione uno strumento molto efficace per risolvere i problemi. I programmi scomposti in moduli, usando la programmazione strutturata, risultano essere più leggibili, più facili da comprendere e di conseguenza rendono più agevole il successivo lavoro di modifica o miglioramento.

12 Funzioni

In matematica una **funzione** è una relazione tra un insieme detto degli **argomenti** e un altro detto dei **valori**. Analogamente nella progettazione degli algoritmi il termine **funzione** sta ad indicare un sottoprogramma che riceve dal programma chiamante i valori assegnati ai parametri formali e restituisce un valore.
Le funzioni vengono quindi usate per rappresentare un procedimento di calcolo o, in generale, un'elaborazione che deve fornire un valore al programma chiamante.

La struttura generale di una funzione è la seguente:

```
NomeFunzione (parametri)
inizio
…
…
fine
```

Per esempio, la funzione per calcolare il risultato dell'addizione di due numeri interi può essere rappresentata in questo modo:

```
Funzione somma (a, b)
inizio
        somma = a + b
fine
```

La funzione riceve dalla procedura chiamante due valori assegnati ai parametri *a* e *b*, e restituisce in *somma* il risultato.

In alternativa è possibile utilizzare l'istruzione **ritorna** per indicare il valore restituito dalla funzione:

```
Funzione somma (a, b)
inizio
        ritorna a + b
fine
```

PROGETTO 14

Calcolo del reciproco di un numero.

Dati di input: un numero.
Dati di output: il reciproco del numero.

Analisi del problema
La funzione riceve come parametro un numero e calcola il reciproco del numero. Controlla anche se il valore del numero è zero: in questo caso restituisce il messaggio "*impossibile*".

Algoritmo in pseudocodifica

```
Funzione reciproco (x)
inizio
    se x=0
    allora
        reciproco = "impossibile"
    altrimenti
        reciproco = 1/x
    fine se
fine
```

13 Logica ricorsiva

Con il termine **ricorsione** (o ricorsività) si indica la possibilità che le funzioni hanno di chiamare se stesse, cioè la chiamata della funzione è contenuta all'interno della funzione stessa.
In matematica la quasi totalità delle funzioni possono essere definite attraverso procedimenti di calcolo ricorsivo.

PROGETTO 15

Calcolo del fattoriale di un numero n (n!).

Dati di input: un numero intero.
Dati di output: il fattoriale del numero.

Analisi del problema
La notazione **n!** si legge fattoriale di *n* e sta ad indicare la moltiplicazione di tutti gli interi minori o uguali ad n:
$n! = n*(n-1)*(n-2)* \ldots 3*2*1$

Per esempio:
$5! = 5*4*3*2*1 = 120$

Inoltre, per definizione, $0! = 1$ e $1! = 1$.

È possibile dare per il calcolo del fattoriale anche una definizione ricorsiva:
per $n = 0$, $n! = 1$
per $n > 0$, $n! = n*(n-1)!$

Esattamente questa definizione viene utilizzata per realizzare un sottoprogramma che calcoli il fattoriale di *n*.
Nella funzione *fattoriale* è contenuta la chiamata della funzione stessa, passando come parametro il valore di x–1.
La ricorsione è possibile perché ad ogni chiamata della funzione, viene generata in memoria centrale una copia della funzione, come se fosse una funzione diversa.

Funzione fattoriale (x)

```
inizio
    se x = 0
    allora
        fattoriale = 1
    altrimenti
        fattoriale = x * fattoriale(x - 1)
    fine se
fine
```

Algoritmo principale

```
inizio
    immetti n
    scrivi fattoriale(n)
fine
```

MATERIALI ON LINE

10. Esempi di funzioni ricorsive

14 Paradigmi di programmazione

I computer vengono utilizzati in diversi ambiti per offrire servizi, per controllare strumentazioni, per scopi educativi e ludici e per comunicare. Questa diversità di utilizzi è resa possibile dal software, cioè dai programmi che i computer sono in grado di eseguire.
I programmi eseguibili dai computer sono realizzati utilizzando i **linguaggi di programmazione** che, tramite uno specifico insieme di termini, permettono di descrivere come il computer si deve comportare.
Da quando sono stati inventati i primi elaboratori, si sono sviluppati diversi linguaggi di programmazione per la creazione di programmi. I primi linguaggi di programmazione erano nati per creare programmi che risolvessero problemi in particolari settori, per esempio i problemi matematici, logici o economici. Successivamente, l'evoluzione dei linguaggi di programmazione si è rivolta verso linguaggi per utilizzi generali, in grado di risolvere una vasta gamma di problemi. L'obiettivo comune a tutti i linguaggi è sempre stato quello di semplificare l'attività di programmazione e aumentare la produttività rendendo più rapido lo sviluppo di programmi e più facile la loro manutenzione.
I linguaggi di programmazione possono essere raggruppati e classificati in base al modo con cui permettono di risolvere i problemi e costruire i programmi. Si dice che un problema viene affrontato e risolto attraverso un determinato paradigma.
Il termine paradigma deriva dal greco *paradeigma*, cioè modello.

Nell'ambito della programmazione, con il termine **paradigma** si intende l'insieme di idee a cui ci si ispira per modellare e per risolvere i problemi.

I paradigmi sono importanti nello studio dei linguaggi di programmazione: ogni linguaggio mette a disposizione istruzioni e costrutti logici che facilitano l'utilizzo di un paradigma per risolvere i problemi. Quindi, per sfruttare pienamente le potenzialità di un particolare linguaggio di programmazione, è necessario conoscere il paradigma sul quale si basa.

I principali paradigmi di programmazione sono:
- imperativo
- orientato agli oggetti
- logico
- funzionale.

È importante osservare che qualsiasi problema, risolvibile con un certo paradigma, lo è anche con un altro. Comunque certi tipi di problemi sono naturalmente adatti per essere risolti con specifici paradigmi. I quattro paradigmi di programmazione elencati in precedenza non devono essere pensati come modalità completamente diverse per modellare i problemi. I paradigmi, e quindi i linguaggi di programmazione che ne derivano, si differenziano per alcuni elementi di base. Questo non vieta che possano anche avere dei punti in comune perché, alla fine, con tutti i linguaggi si produce un programma eseguibile da un computer.

• Paradigma imperativo

Il paradigma di programmazione imperativo assume che la computazione sia portata avanti attraverso una **sequenza ordinata di passi**. Alla base di tutto c'è l'istruzione di **assegnamento** che serve per cambiare il valore delle variabili. L'ordine di esecuzione è fondamentale perché ogni singolo passo può portare a conseguenze diverse a seconda del valore assunto dalle variabili in quel momento.
I linguaggi di programmazione imperativi sono numerosi perché il paradigma imperativo è molto vicino al modo con cui funziona l'elaboratore, cioè attraverso l'esecuzione sequenziale e l'assegnamento. Questi linguaggi di programmazione riservano molta importanza al concetto di **algoritmo**, alla metodologia di **sviluppo top-down** e all'uso delle strutture di controllo. Sono quindi i linguaggi che traducono in pratica le tecniche della **programmazione strutturata**.
Il paradigma imperativo è molto efficiente per un uso generale e non è riservato a specifici problemi.

• Paradigma orientato agli oggetti

Il paradigma orientato agli oggetti analizza il problema cercando di individuare gli **oggetti** che lo compongono e le **correlazioni** esistenti tra gli stessi. Questi oggetti possono essere la rappresentazione di oggetti reali oppure di concetti. Considerando il singolo oggetto, lo si può vedere come composto da un insieme di stati e di operazioni che indicano rispettivamente le proprietà e le funzionalità caratterizzanti l'oggetto. Più oggetti comunicano tra di loro scambiandosi dei messaggi.
Questo paradigma di programmazione sta alla base dei linguaggi di programmazione orientati agli oggetti (per esempio *Java*). Questi linguaggi pongono particolare attenzione al concetto di **sistema** e alla metodologia di programmazione orientata agli oggetti (**OOP**, *Object Oriented Programming*). Nei capitoli successivi saranno presentate le caratteristiche della OOP e i concetti chiave di *classe, incapsulamento, ereditarietà* e *polimorfismo*. L'ereditarietà è il beneficio principale che distingue il paradigma orientato agli oggetti dagli altri paradigmi, perché permette il riutilizzo del codice esistente e la sua estensione.

MATERIALI ON LINE

11. Paradigma logico e funzionale

15 Linguaggi di programmazione

Dal 1950 fino ad oggi sono stati creati moltissimi linguaggi di programmazione. Vediamo nel seguente schema i principali linguaggi con l'anno di creazione e le caratteristiche distintive.

Linguaggio	Anno	Utilizzo	Paradigma
Fortran	1954-1958	Calcolo numerico	Imperativo
Cobol	1959-1961	Applicazioni gestionali	Imperativo
Pascal	1971	Usi generali	Imperativo
Prolog	1972	Intelligenza artificiale	Logico
C	1974	Usi generali, programmazione di sistema	Imperativo
Smalltalk	1971-1980	Applicazioni personali	Ad oggetti
C++	1979	Usi generali	Ad oggetti
Java	1995	Usi generali	Ad oggetti

Negli anni più recenti, dal 1995 in poi, con il progressivo sviluppo delle reti e di Internet, hanno avuto una grande diffusione i linguaggi dedicati alla creazione delle pagine Web e delle applicazioni per Internet:

- i linguaggi **HTML** e **CSS** non sono linguaggi di programmazione, ma linguaggi per l'organizzazione dei contenuti Web e il layout delle pagine da visualizzare con il browser;
- **JavaScript**, **Php**, **ASP.NET** sono invece linguaggi di programmazione per lo sviluppo di applicazioni Web dal lato del client e dal lato del server; tutti sono basati sul *paradigma ad oggetti*.

Inoltre versioni dei linguaggi C++ e Java vengono ampiamente utilizzate per la realizzazione di applicazioni per l'informatica **mobile** (*smartphone* e *tablet*).

I linguaggi di programmazione possono essere classificati in diversi modi. Un primo modo è rappresentato dal loro utilizzo, cioè l'ambito applicativo per cui sono stati principalmente progettati. Si individuano linguaggi più rivolti al calcolo numerico, ad applicazioni gestionali, ad usi nell'intelligenza artificiale, alle reti e Internet oppure ad usi generali. Questi ambiti di utilizzo indicano che i linguaggi di programmazione includono dei costrutti che sono stati pensati in riferimento al tipo di problemi che dovevano risolvere. Per esempio il Cobol, utilizzato per applicazioni gestionali, è indirizzato alla gestione di record e file, in cui i calcoli da eseguire sono semplici ma si basano su grandi quantità di dati. La struttura di dati principale del Cobol è il record mentre le operazioni principali sono legate alla gestione dei file.

Un'altra possibile classificazione è in base al paradigma di programmazione che supportano. Tra i linguaggi imperativi possiamo includere il Pascal e il C, tra quelli logici il Prolog e tra quelli orientati agli oggetti, C++ e Java.

Un ultimo criterio di classificazione è in base al tipo di istruzioni offerte da questi linguaggi, in particolare si possono distinguere per il livello di astrazione in linguaggi di basso livello e linguaggi di alto livello. I primi si basano su istruzioni vicine al funzionamento dell'hardware, come il linguaggio Assembler. I linguaggi di alto livello (o evoluti) come il Pascal, C, C++ e Java consentono di programmare utilizzando istruzioni e costrutti più sofisticati, demandando agli strumenti di compilazione il compito di tradurre i programmi in un formato eseguibile dal computer.

• Linguaggi di basso livello

I primi programmatori avevano come unico strumento per programmare il **linguaggio macchina**. Potevano comporre istruzioni per il calcolatore producendo sequenze di zero (0) e di uno (1). Era un'attività complessa, che portava a compiere diversi errori, molto spesso difficili da individuare. L'attività di programmazione richiedeva molta esperienza e abilità.
Questo modo di programmare è situato a un livello molto vicino all'elaboratore, in quanto si devono elencare passo per passo le istruzioni elementari che la macchina deve eseguire.
Dati i problemi legati all'uso del linguaggio macchina, il livello di astrazione è stato elevato con l'uso del **linguaggio Assembler**, nel quale le istruzioni vengono indicate con un nome simbolico, utile per migliorare la comprensione e la leggibilità dei programmi.
I nomi simbolici rappresentano un primo passo di astrazione rispetto alla macchina sottostante. I programmi scritti in questo modo hanno bisogno di un ulteriore programma, detto **assemblatore**, in grado di tradurre il codice scritto in linguaggio Assembler in codice macchina, cioè sequenze di zero e uno. Il linguaggio Assembler richiede programmatori esperti, ma ha il vantaggio di produrre programmi molto efficienti. La stesura di un programma di grandi dimensioni richiede molto tempo, non tanto per la fase di analisi, quanto, soprattutto, per il trattamento dei registri e la gestione dei tipi di dati non elementari.

Le istruzioni messe a disposizione dai linguaggi Assembler traducono le seguenti operazioni elementari del processore:

- caricamento di valori nei registri o in celle di memoria
- operazioni aritmetiche su numeri
- confronti tra due valori
- salto ad una particolare istruzione.

I linguaggi Assembler sono strettamente legati al microprocessore a cui fanno riferimento; dipendono dall'insieme di istruzioni messe a disposizione e dalle caratteristiche hardware. Ogni computer ha quindi il suo linguaggio Assembler. Un programma, scritto con un certo linguaggio Assembler, può funzionare solo con uno specifico microprocessore: per poter eseguire il programma con calcolatori diversi deve essere completamente riscritto.

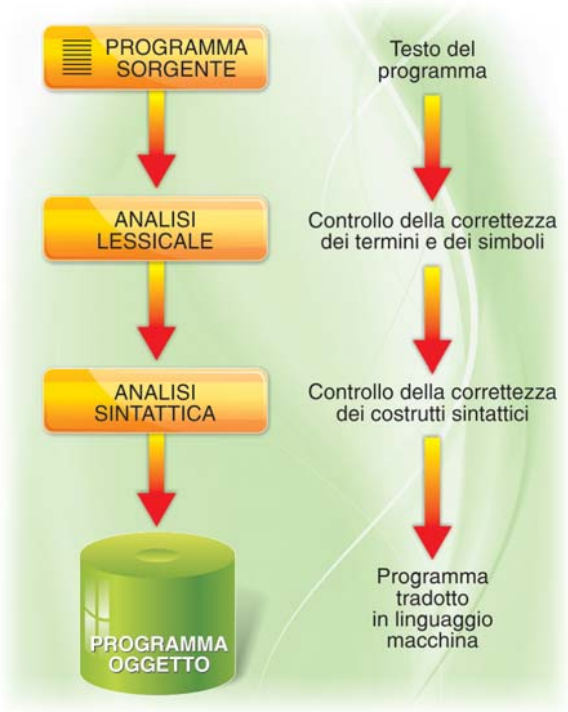
• Linguaggi di alto livello

I linguaggi di alto livello hanno subìto un'ulteriore evoluzione cercando di non essere troppo vincolati al modo con cui l'elaboratore sottostante esegue i comandi. L'obiettivo è stato quello di semplificare l'attività di programmazione e mascherare la complessità dei computer.
Per quanto riguarda il modo di riferirsi alle istruzioni, sono state introdotte parole inglesi che facilitano il lavoro di programmazione rendendo più comprensibile il codice sorgente. I programmi risultano più leggibili e di conseguenza si semplifica anche l'intervento di persone diverse per modificarli. Le istruzioni non si riferiscono direttamente alle operazioni che esegue il calcolatore, ma sono più orientate al problema.
Ne sono un esempio le istruzioni per leggere un valore inserito dall'utente o scrivere un risultato sul video. Questo consente al programmatore di non preoccuparsi di come sono realizzate in pratica queste istruzioni in termini di modifiche dei registri e degli indirizzi di memoria.
I vantaggi introdotti dai linguaggi di alto livello si evidenziano nell'aumento di produttività dei programmatori. L'apprendimento di un nuovo linguaggio è più immediato e lo sviluppo di applicazioni risulta più rapido, perché il lavoro di traduzione è riversato sui compilatori, e si può dedicare più tempo all'analisi del problema. Di contro, nell'uso di questi linguaggi, il codice può risultare meno efficiente e occupare più memoria.
Quest'ultimo svantaggio viene minimizzato dal continuo aumento della capacità elaborativa, in termini sia di velocità che di memoria.

Il testo di un programma scritto usando un linguaggio di programmazione, detto programma **sorgente** (in inglese *source*), deve essere tradotto in **linguaggio macchina**, praticamente nel linguaggio formato dalle cifre binarie 0 e 1, per poter essere eseguito dal computer.

Questa traduzione viene effettuata da un programma traduttore che produce una sequenza di istruzioni eseguibili dall'elaboratore.
Due sono le modalità di base per eseguire questa traduzione:

- **interpretazione,** il programma che prende il nome di *interprete* considera il testo sorgente istruzione per istruzione e lo traduce mentre lo esegue; su questo principio lavorano linguaggi quali il Basic o alcuni linguaggi per la gestione di basi di dati e per le applicazioni Web;
- **compilazione**, il programma traduttore si chiama *compilatore* e trasforma l'intero programma sorgente in linguaggio macchina, memorizzando in un file il risultato del proprio lavoro. Così un programma compilato una sola volta può essere eseguito, senza bisogno di altri interventi, quante volte si vuole. Il risultato della compilazione si chiama programma **oggetto** (in inglese *object*).



La compilazione consente di avere programmi detti **eseguibili**, che possono essere utilizzati senza possedere il compilatore sul sistema dove il programma deve funzionare, mentre con l'interpretazione serve l'insieme di *routine* che consentono di tradurre le righe del programma sorgente (di norma fornite in uno o più file). L'interpretazione non richiede tempi di compilazione, ma, quando il programma è eseguito, ha tempi di traduzione che lo rendono più lento di un programma eseguibile. Per questi motivi l'interpretazione è comoda quando il programma è in fase di progettazione, mentre la compilazione è assolutamente preferibile per un programma applicativo destinato all'utente finale.
La compilazione deve essere seguita da un'ulteriore operazione detta **linking** (collegamento), che viene svolta da un programma apposito chiamato **linker** (collegatore). Tale operazione consiste nell'aggiungere al programma compilato i moduli del compilatore che realizzano le funzioni richieste dai vari comandi, e contemporaneamente di risolvere i riferimenti a celle di memoria o a variabili.
Alla fine di questo lavoro si ottiene il programma **eseguibile**.
Il compilatore genera il proprio output a condizione che l'input, cioè il programma sorgente, sia formalmente corretto, nel senso che deve rispettare le regole del linguaggio scelto. Ogni volta che questo non si verifica, il compilatore emette un messaggio che segnala l'errore.

Gli errori possono riguardare l'uso di termini non appartenenti al linguaggio (**errori di tipo lessicale**) oppure la costruzione di frasi non corrette dal punto di vista delle regole grammaticali del linguaggio (**errori di tipo sintattico**).
Naturalmente il compilatore non è in grado di rilevare **errori logici**, cioè riguardanti la correttezza dell'algoritmo: se un programma traduce un algoritmo sbagliato, ma è scritto nel rispetto delle regole del linguaggio, il compilatore non può accorgersi dell'errore.

Così come con la compilazione non si possono rilevare situazioni di errore che possono verificarsi durante l'esecuzione del programma (**runtime errors**), sulla base di particolari valori assunti dai dati durante l'elaborazione: si pensi per esempio alla divisione per un numero che assume durante l'esecuzione il valore 0.

Gli interpreti trasformano il programma sorgente in linguaggio macchina un'istruzione per volta facendola eseguire subito dopo.

Eventuali errori formali (lessicali o sintattici) vengono rilevati e segnalati solo quando l'istruzione errata viene tradotta e causano l'interruzione dell'esecuzione.

Un particolare programma di utilità durante il lavoro di programmazione è il **debugger** (letteralmente significa spulciatore), che serve per individuare eventuali errori logici nel programma.

Esso può per esempio consentire di:

- eseguire il programma da correggere un'istruzione per volta, così da verificarne la corretta evoluzione;
- controllare i valori assunti dalle variabili in certi punti dell'esecuzione.

16 La produzione del software

Di pari passo con l'evoluzione dei linguaggi di programmazione, si è assistito all'evoluzione della modalità di **produzione del software**. Questa modalità va intesa come tutto l'insieme di attività che intercorrono tra la nascita e la realizzazione di un programma.

Un programma (*software*) è una sequenza di operazioni che il computer deve eseguire per ottenere i risultati attesi. Solitamente la necessità di creare un programma nasce da esigenze del mondo reale: c'è un committente che ha un problema e ha bisogno di un'applicazione, eseguibile con un elaboratore, per risolverlo. Il programmatore, in seguito alla richiesta, si preoccupa di realizzare il programma software.

Il **programmatore** è colui che trasforma il problema, presente nel mondo reale, in modo da poter essere eseguito da un computer.

Il programma software si posiziona come l'intermediario tra due sistemi: da un lato c'è il sistema reale, dove i problemi vengono presentati utilizzando un linguaggio naturale, per esempio l'italiano; dall'altro lato c'è il sistema di calcolo, cioè il computer che esegue in sequenza le istruzioni.

Nel sistema reale il modo di descrivere i problemi è lontano dal modo con cui viene interpretato dall'elaboratore. Nel sistema di calcolo i problemi, per essere risolti, devono essere scritti in forma comprensibile all'elaboratore, cioè come sequenza di zeri e di uno. Questa sequenza è interpretata dal calcolatore come una successione di specifiche istruzioni aventi un preciso significato.

Ci sono quindi due interpretazioni, molto diverse tra loro, dello stesso problema. Il lavoro molto complesso di traduzione tra le due rappresentazioni è affidato al programmatore. Finché le due rappresentazioni restano lontane tra loro, il compito di realizzare un programma software risulta dispendioso.

La realizzazione delle applicazioni è stata semplificata avvicinando il software al sistema reale. Questo è stato reso possibile con:

- l'ingegneria del software, un'evoluzione nella modalità di produzione del software;
- l'evoluzione dei linguaggi di programmazione.

L'**ingegneria del software** si è sviluppata per facilitare la rappresentazione del mondo reale. Essa punta al miglioramento della produzione del software, incentivando l'uso di un insieme di regole e strumenti per creare e sviluppare il software secondo precise metodologie.
Un ruolo importante è svolto dall'utente di un'applicazione, il quale viene coinvolto direttamente nella fase di realizzazione del software. L'ingegneria del software si è occupata di definire un **ciclo di vita del software**, cioè un periodo di tempo durante il quale un programma software viene ideato, implementato ed aggiornato. Esso è costituito da diverse **fasi**:

- Definizione degli obiettivi
- Analisi
- Progettazione
- Produzione
- Prove e formazione degli utenti.

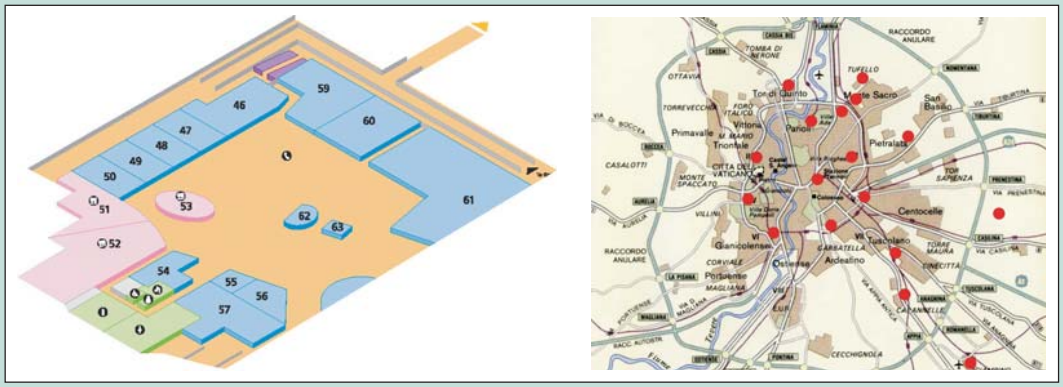
L'evoluzione nella produzione del software e nei linguaggi di programmazione può essere vista come un tentativo di ridurre la complessità dei sistemi di calcolo. Alla base di questa evoluzione c'è uno strumento molto usato per gestire la complessità, chiamato *astrazione*.
Per quanto riguarda i linguaggi di programmazione, sono stati abbandonati i linguaggi macchina, vicini al modo di operare del processore, a favore di linguaggi più vicini al linguaggio naturale usato dall'uomo. Questi linguaggi rappresentano un'astrazione rispetto al modo di operare del processore.

MATERIALI ON LINE

12. Le fasi dello sviluppo del software
13. Il software *open source*

L'ASTRAZIONE

Il mondo in cui viviamo è molto complesso. Pensando alle cose che ci circondano ci si accorge che ognuna ha un suo grado di complessità: si pensi per esempio alle schede e ai componenti hardware di cui è composto un computer, a un'automobile oppure a una grande città. È difficile per un individuo riferirsi a questi oggetti e comprenderne ogni dettaglio. Se si considera una grande città, risulta difficile ricordarsi come raggiungere tutte le vie di quella città.
È difficile gestire una realtà complessa, ma è un compito ancora più arduo la costruzione di oggetti complessi. Questa situazione è comune ai programmatori quando devono realizzare un'applicazione software di grandi dimensioni.
Gli uomini hanno difficoltà a gestire la complessità, ma hanno anche un modo per poterla controllare: questo modo è l'*astrazione*. Anche se non ce ne accorgiamo, usiamo moltissimo la nostra abilità di astrarre. Per esempio una piantina stradale rappresenta un'astrazione di una città. La piantina mostra tutte le strade tralasciando gli altri aspetti della città. Non vengono cioè disegnate le singole case, non si considerano i semafori e non si indicano le posizioni dei negozi. Questo perché la cosa che interessa vedere in una cartina sono le strade. Ma questo dipende dal livello di astrazione della cartina: infatti in una piantina, che è stata fatta per esigenze pubblicitarie, i dettagli aumentano quando vengono indicate, con particolari simboli, le posizioni dei negozi e la descrizione del tipo di servizi offerti.

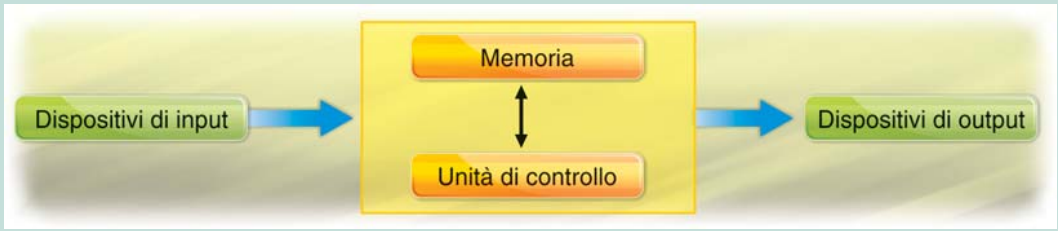


L'**astrazione** è un procedimento che consente di semplificare la realtà che vogliamo analizzare. La semplificazione avviene concentrando l'attenzione solo sugli elementi importanti dell'oggetto complesso che stiamo considerando.

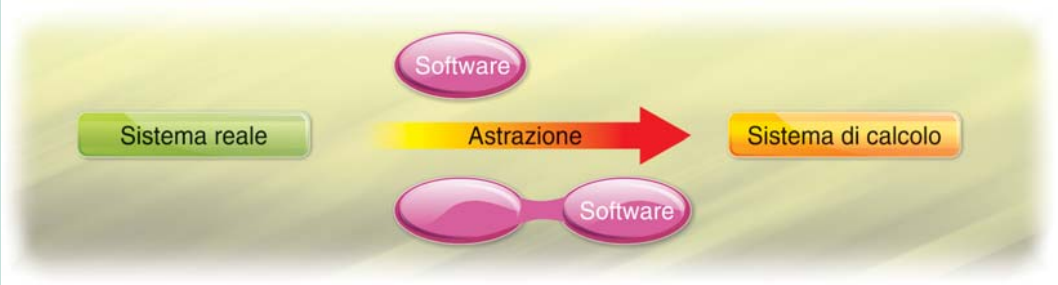
A seconda del tipo di astrazione che vogliamo fare, cambiano anche gli aspetti che vengono considerati importanti; quelli giudicati irrilevanti possono essere trascurati. Si prenda come esempio un computer: esso può essere considerato a diversi livelli di astrazione. Una segretaria che usa il computer per scrivere delle lettere considera il computer come un oggetto che le consente di scrivere un testo sul monitor, di modificarlo e infine di stamparlo. Per la segretaria ha poca importanza come è costruito il programma di elaborazione dei testi o come funziona la scheda video. L'astrazione rispetto al computer è limitata ai compiti che esso consente di svolgere.
Un altro livello di astrazione, invece, è quello del tecnico che ragiona in termini di circuiti, schede e altri dispositivi hardware. L'astrazione fatta dal tecnico sarà composta da un insieme di dispositivi collegati tra loro. Come si vede, allo stesso oggetto si possono associare diverse visioni che focalizzano l'attenzione su alcuni elementi piuttosto che su altri.

Il risultato dell'astrazione è la creazione di un **modello** che ha il vantaggio di ridurre la complessità della realtà ignorando i dettagli. L'astrazione, vista in questo modo, corrisponde al processo di modellazione.

L'astrazione risulta essere uno strumento molto utile. Il momento giusto di procedere con l'operazione di astrazione è quando ci si accorge che non si riesce più a gestire la complessità del fenomeno che si sta analizzando.
Se per esempio si deve spiegare il funzionamento di un elaboratore a un nuovo studente di informatica, non ha senso addentrarsi nei particolari di come il processore esegue una certa istruzione. Quello che si farà è un'astrazione sul funzionamento del computer. Le cose importanti da includere nella schematizzazione del funzionamento saranno la memoria, l'unità di controllo (CPU), i dispositivi di input e output.



L'astrazione riveste un ruolo importante nella produzione del software. Un'applicazione software solitamente rappresenta un'astrazione della realtà. Si pensi a un'azienda che vuole gestire un archivio: nelle fasi che portano alla realizzazione del programma si crea un modello di quello che si vuole gestire e si indicano le entità che devono essere inserite nell'archivio. Il programma dovrà poi simulare tutte le fasi che si riferiscono alla gestione dell'archivio: dovrà specificare un modo per inserire i valori nell'archivio come pure un modo per ritrovarli.



In questo caso l'astrazione corrisponde all'individuazione degli elementi da memorizzare e alla definizione delle operazioni che si possono effettuare sull'archivio. Il processo di astrazione assume un ruolo decisivo soprattutto nella fase di analisi del problema.

A livello di sistema di calcolo, il ruolo svolto dall'astrazione è più evidente. Per esempio, le variabili rappresentano un'astrazione dei registri della CPU e delle celle di memoria, mentre le istruzioni dei moderni linguaggi di programmazione rappresentano un'astrazione delle operazioni di base degli elaboratori.
In generale si può dire che i linguaggi di programmazione si sono evoluti sia nelle **strutture dati** che mettono a disposizione, sia nelle **strutture di controllo**. L'aumento di astrazione riferita alle strutture di controllo ha dato origine alla metodologia della programmazione strutturata. L'aumento di astrazione riferita alle strutture dati ha portato alla formalizzazione utilizzata nella programmazione orientata agli oggetti.

AUTOVERIFICA

Domande da 9 a 13 pag. 103
Problemi da 16 a 20 pag. 105

AUTOVERIFICA

DOMANDE

Progettazione degli algoritmi

1 Con quale termine viene indicato il trasferimento di un problema sul sistema di elaborazione?

a) Traduzione
b) Implementazione
c) Programmazione
d) Traslazione

2 Quali sono le attività che in un algoritmo consentono di ottenere i risultati desiderati, mettendo i dati in relazione tra loro?

a) Istruzioni
b) Controlli
c) Azioni
d) Trasformazioni

3 Completa le frasi seguenti scegliendo tra le parole elencate al termine della domanda.

a) Le ..... descrivono le azioni che deve compiere un esecutore.
b) Un ..... è formato da più azioni elementari.
c) Si dice ..... l'ente che esegue le azioni secondo le istruzioni di un algoritmo.

controllo, processo, modello, algoritmo, processore, istruzioni, variabili

4 Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.

a) Un ..... è un insieme di operazioni da eseguire per ottenere certi risultati.
b) Un ..... è la rappresentazione grafica di un algoritmo.
c) Il termine ..... indica la rappresentazione di un algoritmo tramite un linguaggio naturale.
d) Un ..... è la rappresentazione di un algoritmo in una forma comprensibile all'elaboratore.

programma software, oggetto, diagramma a blocchi, paradigma, pseudocodifica, algoritmo

5 Quale tra le seguenti definizioni descrive meglio la Macchina di Turing?

a) Un esempio di computer di piccole dimensioni.
b) Un automa in grado di eseguire solo le quattro operazioni aritmetiche.
c) Una macchina astratta in grado di eseguire un algoritmo.
d) Una macchina in grado di scrivere simboli su un nastro.

Strutture di controllo

6 Secondo il teorema di Böhm-Jacopini, quali strutture di controllo sono sufficienti per esprimere un algoritmo?

a) sequenza
b) selezione
c) interruzione
d) negazione
e) iterazione

7 Che cosa viene scritto in output dopo l'esecuzione delle seguenti istruzioni?

```
assegna a=5
assegna b=3
se a>b
allora
      calcola c = a - b
altrimenti
      calcola c = b - a
fine se
scrivi c
```

8 Che cosa viene scritto in output eseguendo gli algoritmi a) e b)?

```
a)                          b)
assegna x = 0               assegna x = 0
esegui                      esegui mentre x <= 0
      calcola x = x + 1           calcola x = x + 1
      scrivi x                    scrivi x
ripeti mentre  x <= 0       ripeti
```

Sviluppo del software

9 Quali delle seguenti istruzioni è un esempio corretto di chiamata ricorsiva della funzione *fattoriale*?

a) fattoriale=x*fattoriale
b) fattoriale(x-1)=x*fattoriale(x)
c) fattoriale=x*fattoriale(x-1)
d) fattoriale(x)=x*(x-1)

10 Quali delle seguenti affermazioni, riferite ai paradigmi di programmazione, sono vere (V) e quali false (F)?

a) Un paradigma è un insieme di idee a cui ci si ispira per risolvere i problemi V F
b) Il paradigma imperativo si basa sull'uso di funzioni e sulla loro composizione V F
c) Particolari problemi sono risolti più efficientemente con certi paradigmi V F
d) Java è un linguaggio che usa il paradigma orientato agli oggetti V F

11 Con quale strumento si traduce il codice sorgente in codice oggetto?

a) editor
b) compilatore
c) interprete
d) linker

12 Metti in ordine le fasi del ciclo di vita del software

a) Analisi.
b) Progettazione.
c) Produzione.
d) Definizione degli obiettivi.
e) Prove e formazione degli utenti.

13 Quale tra queste definizioni corrisponde al termine *astrazione*?

a) Procedimento che consente di semplificare la realtà che vogliamo analizzare.
b) Procedimento che complica la realtà che vogliamo analizzare.
c) Algoritmo che consente di semplificare la realtà.
d) Procedimento che trascura le qualità considerate rilevanti.

AUTOVERIFICA

PROBLEMI

Progettazione degli algoritmi

1 Per calcolare il volume di una sfera, conosciuto il raggio, si utilizza la formula $v = 4/3\ \pi r^3$. Distinguere in questa formula le variabili e le costanti.

2 Date le temperature registrate nei giorni del mese di luglio in una città, si vuole sapere in quanti giorni del mese si è avuta una temperatura superiore ai 30°. Produrre uno schema di documentazione delle variabili e costanti, secondo la traccia seguente:

Identificatore	Variabile o costante	Descrizione	Tipo
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....

3 Usando una pseudocodifica, descrivere un algoritmo che, acquisiti in input due numeri, restituisca il quoziente della divisione nel caso in cui il secondo numero sia diverso da zero, altrimenti mandi all'esterno un messaggio di errore.

4 Usando una pseudocodifica, descrivere l'algoritmo per calcolare il perimetro di un triangolo rettangolo, noti i cateti.

5 Usando un diagramma a blocchi, descrivere un algoritmo che, acquisiti in input tre numeri, ne calcoli la media aritmetica.

6 Usando un diagramma a blocchi, descrivere l'algoritmo per scambiare il contenuto di due variabili se non sono in ordine crescente (occorre usare una terza variabile temporanea).

7 Costruire una semplice Macchina di Turing per il calcolo del successivo di un numero intero. Normalmente la testina scrive la cifra successiva a quella letta nella casella del nastro; quando legge la cifra 9 deve scrivere 0 e passare a uno stato che serve a ricordare il riporto. All'inizio la testina è posizionata sull'ultima cifra a destra, e il movimento avviene verso sinistra.

Strutture di controllo

8 Di una città viene fornita la popolazione di due anni successivi. Calcolare l'incremento di popolazione.

9 Date le coordinate di due punti scrivere l'equazione della retta passante per essi (controllare che la retta non sia verticale).

10 Date le equazioni di due rette nella forma $y=mx+q$, trovare le coordinate del punto di intersezione (segnalare anche il caso di rette parallele e di rette coincidenti).

11 Data l'equazione di una retta nella forma $y=mx+q$ e le coordinate di un punto, dire se il punto appartiene alla retta.

12 Data una sequenza di numeri, contare quelli positivi e quelli negativi. Comunicare il numero dei positivi e dei negativi, e comunicare la percentuale dei positivi e dei negativi rispetto al totale dei numeri.

13 Si introduce da tastiera un elenco di libri formato da autore, titolo e prezzo. Calcolare il totale e applicare sul totale il 4% in più per le spese di spedizione.

14 Uno studente esamina le informazioni sulle verifiche che ha fatto durante l'anno scolastico, e vuole contare quante sono le prove sufficienti.

15 Dopo aver inserito un numero intero N, sommare i primi N numeri dispari e verificare che tale somma è uguale al quadrato di N.

Sviluppo del software

16 Scrivere una funzione che restituisca l'età di una persona, conoscendo il suo anno di nascita.

17 Scrivere una funzione per stabilire se due rette $a_1x+b_1y+c_1=0$ e $a_2x+b_2y+c_2=0$ sono perpendicolari.

18 Calcolare la somma dei primi *n* interi utilizzando la definizione ricorsiva:

$$\sum_{i=1}^{1} i = 1$$

$$\sum_{i=1}^{n} i = n + \sum_{i=1}^{n-1} i$$

19 Ricercare su Internet i nomi e le caratteristiche dei moderni linguaggi di programmazione di alto livello.

20 Fornire esempi di errori lessicali, sintattici e di run-time che si possono incontrare durante il lavoro di programmazione.

Focus notes

COMPUTER PROGRAMMING

An *algorithm* is a procedure for solving a problem. In computer science an algorithm is a finite, deterministic, and effective problem-solving method suitable for implementation as a computer program.

Implementation defines how something is done by a computer. In computer science terms, *implementation* is the code written in a programming language.

The three tasks of any computer program are: reading input data, processing it and displaying the results.

The data is read from the *standard input* device (keyboard).

The results are displayed on the *standard output* device (screen).

A data value is assigned to the variable in an *assignment* statement.

In a basic program, the instructions are executed sequentially in the order in which they appear in the program (*sequence*).

Decision-making statements alter the sequence of the instructions depending upon certain conditions and transfer the control from one part to other parts of the program (*selection*).

A loop is a group of statements, which are repeated for a certain number of times (*repetition*).

Loops are of two types:

- *counter-controlled loop*, the number of iterations to be performed is defined in advance in the program itself.
- *condition-controlled loop,* loop termination happens on the basis of certain conditions.

In *Bohm-Jacopini*'s theory (*theorem*) a well-structured program is built combining the instructions in only three specific control structures: sequence, selection and repetition.

Sequence means statements appearing one after another. Selection is implemented using *if . . . else* and *switch* statements. Repetition is implemented using *for* and *do . . . while* statements.

Focus notes

Glossary

algorithm
A process or a set of rules used for calculation or problem solving.
In computer programming the term is used referring to instructions given to a computer.

compiler
A computer program which translates a program, written in a *high level language* (programming language), into an equivalent *machine language* program.

data structure
A way of storing data in a computer so that it can be organised and retrieved efficiently.

debugging
A process of finding and removing the *bugs* in a computer program that produce an incorrect or unexpected result.

implementation
Converting an algorithm into a computer program using a particular programming language and an operating system.

iteration
Repetition of a process within a computer program (synonymous with repetition or loop).

machine language
The binary language in which the CPU and the hardware of a computer are to be programmed.

operand
An operand is a data item on which operators perform operations.

operator
An operator indicates an operation to be performed on data.

Acronyms

CPU	Central Processing Unit
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
I/O	Input/Output
OOP	Object Oriented Programming
TM	Turing Machine

SCHEDA DI AUTOVALUTAZIONE

CONOSCENZE

- ☐ Variabili e costanti, dati e azioni
- ☐ La metodologia di lavoro nella formalizzazione dei problemi
- ☐ Definizione e caratteristiche di algoritmo
- ☐ Operazioni di input e di output
- ☐ Gli operatori
- ☐ Strumenti per la stesura di un algoritmo
- ☐ L'individuazione dei dati di un problema
- ☐ La Macchina di Turing
- ☐ Le strutture di controllo
- ☐ Teorema di Böhm-Jacopini
- ☐ Lo sviluppo top-down e l'organizzazione dei programmi
- ☐ Passaggio di parametri
- ☐ Le funzioni
- ☐ Ricorsività
- ☐ Paradigmi della programmazione
- ☐ Linguaggi di programmazione
- ☐ Produzione del software
- ☐ Astrazione e modelli

ABILITÀ

- ☐ Saper distinguere all'interno di un problema tra variabili e costanti, tra dati ed azioni
- ☐ Utilizzare la pseudocodifica per rappresentare gli algoritmi
- ☐ Rappresentare graficamente gli algoritmi con i diagrammi a blocchi
- ☐ Definire una Macchina di Turing in modo formalizzato
- ☐ Costruire algoritmi strutturati
- ☐ Rappresentare le strutture di controllo
- ☐ Individuare le strutture di controllo più idonee per la soluzione di un problema
- ☐ Utilizzare il metodo dei raffinamenti successivi per la soluzione di problemi complessi
- ☐ Dichiarare e utilizzare una funzione
- ☐ Comprendere l'importanza di procedimenti ricorsivi
- ☐ Descrivere le caratteristiche generali dei paradigmi di programmazione
- ☐ Cogliere gli aspetti evolutivi dei linguaggi di programmazione
- ☐ Individuare le motivazioni che portano alla programmazione ad oggetti
- ☐ Individuare gli elementi comuni a tutti i linguaggi di programmazione

SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 539

parte seconda

La programmazione

3 Le basi del linguaggio Java

OBIETTIVI DI APPRENDIMENTO

In questo capitolo conoscerai gli aspetti caratteristici del linguaggio di programmazione.

Sarai in grado di progettare la struttura generale di un programma e di utilizzare correttamente la sintassi e i costrutti del linguaggio.

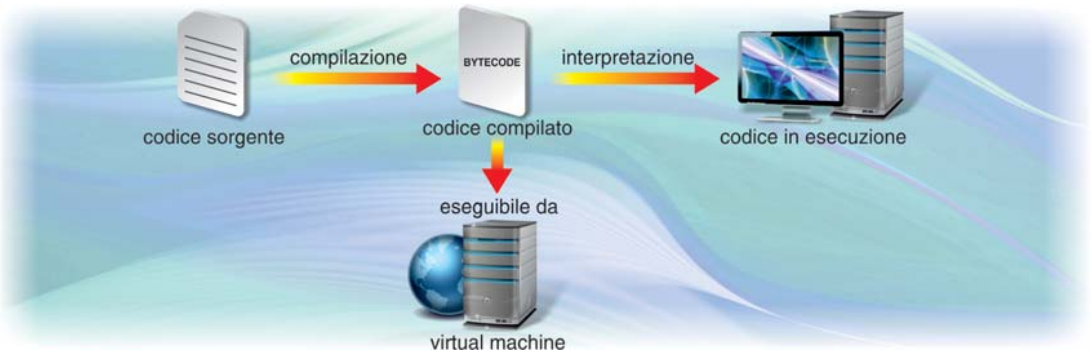
1 Caratteristiche generali

Java è un linguaggio di alto livello e orientato agli oggetti. Si può usare per scrivere applicazioni di diversa natura, ed è quindi considerato un linguaggio per usi generali. Java nasce all'interno di un progetto di ricerca molto ampio che aveva lo scopo di creare software avanzato per vari sistemi. Inoltre è stato pensato in riferimento all'importanza che hanno assunto le reti di computer negli ultimi anni ed ha avuto una forte integrazione con Internet, con la possibilità di scrivere piccole applicazioni per la rete (*applet*), eseguibili all'interno delle pagine Web visibili in Internet. Il primo programma consistente, scritto usando Java, è stato un browser Web.

Java è stato creato dalla *Sun Microsystem* e ha fatto la sua apparizione ufficiale nel 1995. Successivamente, nel 2010, la *Sun* è stata acquistata da *Oracle*.
Le caratteristiche del linguaggio di programmazione Java che lo rendono molto attraente sono diverse. Innanzitutto, Java è un linguaggio semplice e divertente da imparare. Inoltre è un linguaggio di programmazione *orientato agli oggetti*. Questo significa che vengono messi a disposizione dei costrutti per trattare i concetti fondamentali di *oggetto*, *classe* ed *ereditarietà*. Un'altra caratteristica importante è la capacità di costruire applicazioni che sono *portabili* su differenti piattaforme.

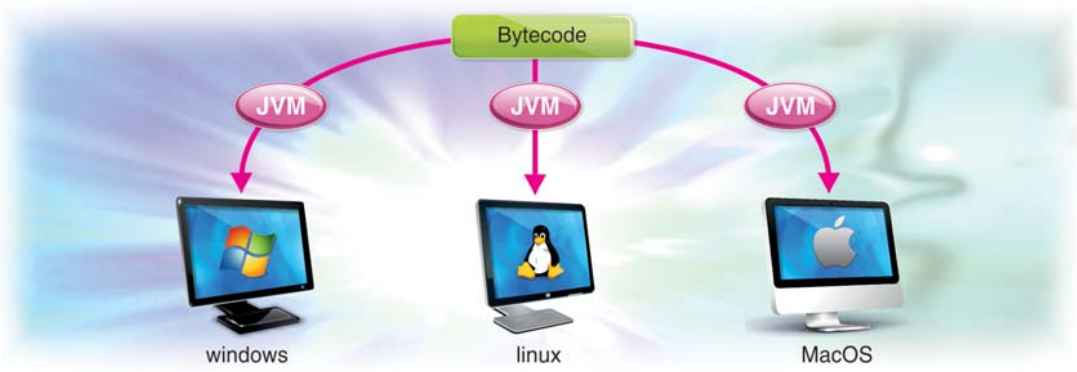
La **portabilità** è la capacità di un programma di essere eseguito su piattaforme diverse senza dover essere modificato e ricompilato.

Il termine **piattaforma** fa riferimento all'architettura hardware e software di un elaboratore, in particolare al processore e al sistema operativo installato. Le piattaforme sono diverse se sono basate su un diverso processore oppure su un diverso sistema operativo (per esempio: piattaforma *Windows*, piattaforma *Linux*, piattaforma *MacOS* per i computer *Apple*).
Se si esegue la compilazione di un programma C/C++ su una piattaforma Windows, quello che si ottiene è un codice eseguibile legato alla piattaforma su cui è stato compilato. Questo perché contiene istruzioni eseguibili solo da quel particolare processore e sfrutta le utilità di quello specifico sistema operativo. Un programma compilato in questo modo girerà sui sistemi Windows, ma non su piattaforme Linux.
Per far eseguire il programma anche sul sistema operativo Linux dobbiamo effettuare l'operazione di **porting**: dobbiamo cioè compilare il programma con un compilatore che sia in grado di generare un codice eseguibile Linux. Durante questa operazione possono essere necessarie alcune modifiche al codice sorgente. In conclusione, un programma che richiede tutte queste modifiche, per poter essere eseguito su una piattaforma diversa, non è portabile.
Al contrario un'applicazione Java, una volta scritta e compilata, non ha bisogno di subire le operazioni di *porting*. Tale applicazione può essere eseguita senza modifiche su diversi sistemi operativi ed elaboratori con architetture hardware diverse.
Vediamo come viene realizzata in pratica la portabilità con Java.



Quando si compila il codice sorgente scritto in Java, il compilatore genera il codice compilato, chiamato **bytecode**. Questo è un formato intermedio indipendente dall'architettura ed è usato per trasportare il codice in modo efficiente tra varie piattaforme hardware e software.
Questo codice non può essere eseguito da una macchina reale. È un codice generato per una macchina astratta, detta **Java Virtual Machine (JVM)**.

Per essere eseguito su una macchina reale, il codice compilato (*bytecode*) deve essere interpretato. Significa che ogni istruzione della *Virtual Machine* viene tradotta nella corrispettiva istruzione della macchina su cui si sta eseguendo l'applicazione in quel momento. A seconda dell'elaboratore su cui il programma viene eseguito, si ha un'interpretazione diversa. È questa la modalità attraverso la quale le applicazioni Java diventano portabili.



Per questi motivi si può dire che Java è un linguaggio **interpretato**, anche se la produzione del *bytecode* è effettuata con un'operazione di compilazione.

In questo capitolo saranno presentate le nozioni di base sul linguaggio Java, mentre nel capitolo successivo saranno spiegate le classi e gli oggetti e le modalità di implementazione della programmazione ad oggetti.

2 L'ambiente di programmazione

Esistono numerosi strumenti e ambienti di sviluppo per il linguaggio Java: il più famoso è il **JDK** (*Java Development Kit*), distribuito gratuitamente da *Oracle*, con il quale si possono scrivere e compilare *applet* e applicazioni Java.
È un ambiente senza interfacce grafiche, nel senso che il compilatore e l'interprete sono eseguiti da riga di comando. Nel caso di un sistema Windows significa che per compilare ed eseguire un'applicazione Java si deve aprire il *Prompt dei comandi*.

Per poter programmare in Java è necessario predisporre un ambiente di sviluppo. Gli strumenti fondamentali sono:

- un editor di testi (per esempio *Blocco note* di Windows)
- un compilatore e un interprete Java (compresi nel JDK distribuito da *Oracle*)
- un browser Web abilitato ad eseguire le *applet*.

Il JDK viene distribuito come un'applicazione autoinstallante; se si è scelto di installare il JDK, per esempio, nella directory *C:\Programmi\Java\jdk1.7.0*, il compilatore e l'interprete saranno posizionati nella directory *C:\Programmi\Java\jdk1.7.0\bin*.

Per evitare di fare sempre riferimento a questa directory ogni volta che si chiede l'esecuzione del compilatore, occorre modificare la **variabile di ambiente PATH** nelle *Variabili d'ambiente* in *Proprietà del sistema di Windows*, aggiungendo la directory *C:\Programmi\Java\jdk1.7.0\bin*: fare clic con il tasto destro del mouse su *Computer*, *Proprietà*, *Impostazioni di sistema avanzate*, *Avanzate*, *Variabili d'ambiente*.

Occorre osservare che la rete Internet offre la possibilità di trovare molti supporti per l'attività di programmazione: editor, compilatori, documentazione, annunci, tutoriali, esempi, demo e altre utili informazioni. Da Internet si possono ottenere anche, spesso in forma gratuita, software di tipo **IDE** (*Integrated Development Environment*), cioè ambienti che facilitano lo sviluppo dei programmi attraverso un'interfaccia grafica e una modalità di programmazione visuale. Alcuni di questi software sono presentati nell'inserto *Ambienti di sviluppo in Java* dopo questo capitolo.

L'ambiente di programmazione Java include un insieme di **librerie** contenenti *classi* e *metodi* di varia utilità per lo sviluppo di applicazioni. Le principali librerie sono:

- **java.lang**: collezione delle classi di base che è sempre inclusa in tutte le applicazioni.
- **java.io**: libreria per la gestione degli accessi ai file e ai flussi di input e output.
- **java.awt**: libreria contenente le classi per la gestione dei componenti grafici (colori, font, pulsanti, finestre).
- **java.net**: supporto per creare applicazioni che si scambiano dati attraverso la rete.
- **java.util**: classi di utilità varie (array dinamici, gestione della data, struttura di dati *stack*).

La disponibilità di un vasto insieme di librerie consente di raggiungere uno degli obiettivi della programmazione orientata agli oggetti: la **riusabilità del software**. Si può dire che le versioni più recenti di Java differiscono dalle precedenti solo per il numero maggiore di librerie incluse piuttosto che per i costrutti del linguaggio.
Le librerie di Java sono costituite da varie *classi* già sviluppate e raggruppate in base all'area di utilizzo, e sono molto utili per la creazione di nuovi programmi.

L'insieme delle librerie fornite dall'ambiente di programmazione viene anche indicato con il termine **API** (*Application Programming Interface*). Il JDK è fornito di una ricca documentazione in cui vengono descritte le specifiche e il modo d'uso di queste API.

Un'altra caratteristica importante di Java è la **gestione della memoria** effettuata automaticamente dal sistema di *run-time*. Questo sistema si occupa dell'allocazione della memoria e della successiva deallocazione. Il programmatore viene liberato dagli obblighi di gestione della memoria.
I programmi generati in questo modo risultano più affidabili e robusti, nel senso che vengono evitati gli errori molto frequenti nella programmazione in C/C++ legati all'uso dei *puntatori*.
Per ridurre questi problemi, gli inventori di Java hanno deciso di non inserirli nel linguaggio. Tutto quello che veniva fatto in altri linguaggi di programmazione usando i puntatori, può essere fatto in Java usando gli oggetti. Essi vengono allocati dinamicamente quando servono e viene liberata automaticamente la memoria quando non vengono più utilizzati. L'assenza di puntatori consente di scrivere programmi più affidabili e più leggibili.

MATERIALI ON LINE

1. Java Development Kit (JDK): installazione e modalità operative

3 La struttura dei programmi

Utilizzando il linguaggio di programmazione Java, si possono realizzare delle *applicazioni*, cioè dei programmi veri e propri che possono essere eseguiti autonomamente.

In Java, un'applicazione può essere costituita da una o più **classi**. Tra tutte le classi che compongono l'applicazione, una si differenzia dalle altre perché contiene il metodo **main().** Questo *metodo* è importante perché l'esecuzione di un'applicazione Java comincia eseguendo questo metodo.
Per iniziare, ci limiteremo alla costruzione di applicazioni con una sola classe. Successivamente saranno trattate applicazioni che usano più classi.

Un semplice programma Java, formato da una sola classe, assume la seguente struttura generale:

```
class <nome classe>
{
  public static void main(String args[])
  {
    // dichiarazioni di variabili
    . . .
    // istruzioni
    . . .
  }
}
```

La classe dichiarata in questo modo contiene il solo metodo *main*. Questo metodo raggruppa le dichiarazioni di variabili e le istruzioni che compongono l'applicazione Java.

Esso viene dichiarato usando le parole chiave *public*, *static* e *void* che specificano alcune proprietà del metodo:

- **public** indica che il metodo è pubblico ed è visibile;
- **void** indica che non ci sono valori di ritorno;
- **static** indica che il metodo è associato alla classe e non può essere richiamato dai singoli oggetti della classe (il suo significato sarà presentato più in dettaglio nel seguito).

Dopo il nome del metodo, tra parentesi, sono indicati i parametri. Il metodo *main* possiede come parametro un array di stringhe (indicato con **args[]**) che corrisponde ai parametri passati dalla riga di comando quando viene eseguita l'applicazione.

Le parentesi graffe **{...}** sono usate per individuare un **blocco**, che può essere: una classe, un metodo oppure un insieme di istruzioni. All'interno di un blocco possono essere presenti altri blocchi in modo annidato. Per poter meglio distinguere questi blocchi conviene seguire e usare una tecnica di **indentazione**. Anche se un programma Java potrebbe essere scritto usando una sola riga, è utile disporre ordinatamente le varie istruzioni e gli elementi sintattici per migliorare la leggibilità del codice.
Le righe che iniziano con **//** indicano **frasi di commento**.

Basandoci sulla struttura appena presentata, possiamo scrivere un semplice programma che stampa sul video un messaggio.

```
class Domanda
{
  public static void main(String args[])
  {
    System.out.println("Quanti anni hai?");
  }
}
```

Questo semplice esempio dichiara una classe chiamata *Domanda*, contenente il metodo *main* che è formato da un'unica istruzione, precisamente una chiamata del metodo **System.out.println().** In particolare l'oggetto *System.out* indica lo *standard output* (video). Il risultato dell'esecuzione di questo programma è la visualizzazione della scritta "*Quanti anni hai?*"; poi il programma termina.
System.out.println() è un metodo che scrive sullo *standard output* i suoi argomenti e torna a capo.

In Java ogni istruzione deve terminare con il **punto e virgola** (;). Questo terminatore va inserito alla fine di tutte le istruzioni e di tutte le dichiarazioni.

È importante sottolineare che Java è un linguaggio **case-sensitive**. Questo significa che vi è differenza nello scrivere una lettera maiuscola o minuscola. Per esempio l'identificatore *System* è diverso da *system*. Un errore nella digitazione comporta la segnalazione di un errore di compilazione.

Per realizzare un programma e renderlo eseguibile si devono seguire i seguenti passi:

1) editare il codice sorgente
2) compilare il codice sorgente
3) eseguire il codice compilato.

Per **editare** il codice Java è possibile usare il programma *Blocco note* di Windows o qualsiasi altro editor di testi.

Il codice sorgente deve essere salvato in un file con estensione **.java**. Il nome da dare al file corrisponde al nome della classe, cioè il nome inserito dopo la parola chiave *class*. Il programma dell'esempio precedente deve essere salvato in un file chiamato *Domanda.java*.
Se il programma è composto da più classi, conviene memorizzare ogni classe in un file diverso.

Per **compilare** il programma dobbiamo usare il *Prompt dei comandi*. Il compilatore Java viene richiamato usando il seguente comando:

```
javac <nome file>
```

Supponendo che il programma *Domanda.java* si trovi nella directory *c:\sorgenti*, prima occorre posizionarsi nella directory *c:\sorgenti* e poi eseguire la compilazione nel seguente modo:

```
C:\sorgenti>javac Domanda.java
```

La fase di compilazione genera un file compilato che ha estensione **.class** e rappresenta il *bytecode*. Il file generato assume il nome *Domanda.class*.

Per **eseguire** il programma bisogna attivare l'interprete Java. L'interprete prende il codice compilato (*bytecode*), lo traduce in codice macchina e lo esegue. L'interprete Java viene richiamato usando il comando:

```
java <nome file>
```

Il nome del file passato all'interprete deve essere indicato senza estensione e corrisponde alla classe che contiene il metodo *main*.
Eseguendo il programma, il risultato che si ottiene è il seguente:

```
C:\sorgenti>java Domanda
Quanti anni hai?
```

AUTOVERIFICA

Domande da 1 a 7 pag. 142
Problemi da 1 a 3 pag. 146

MATERIALI ON LINE

2. Editing, compilazione ed esecuzione di un programma Java in Windows

4 Gli identificatori e le parole chiave

Un programma scritto in Java si compone di parole che si possono classificare in due gruppi:

- identificatori
- parole chiave del linguaggio.

Gli **identificatori** sono i nomi che il programmatore assegna alle specifiche componenti del suo programma, per identificare le variabili, i metodi e le classi.
Un identificatore è composto da una sequenza di lettere e di numeri. Il primo carattere deve essere una lettera. È anche possibile usare il carattere di sottolineatura _.

Bisogna fare molta attenzione alla scrittura degli identificatori perché, come già detto, Java distingue le lettere maiuscole e minuscole.
Per esempio, i seguenti identificatori:

```
Nome
NOME
```

sono considerati diversi.

Gli identificatori assumono un ruolo fondamentale per la leggibilità e la comprensione di un programma. Utilizzare identificatori significativi da associare ad attributi, metodi e classi, rende il programma autodocumentato.

Solitamente si seguono le seguenti convenzioni:

- gli identificatori che si riferiscono al nome di attributi, di metodi e di oggetti cominciano con la prima lettera minuscola (per esempio, *nome*, *voto*, *aggiorna*, *auto*);
- gli identificatori che specificano il nome delle classi iniziano con la lettera maiuscola (per esempio, *Libro*, *Automobile*, *Rettangolo*);
- se gli identificatori sono formati da più parole, queste vengono unite e ogni parola successiva alla prima ha l'iniziale maiuscola (per esempio, *totFattura*, *contaSecondi*, *calcolaMedia*).

Le **parole chiave** sono un insieme di parole riservate di Java che non possono essere usate come identificatori.

Parole chiave del linguaggio Java

abstract	continue	goto	package	this
assert	default	if	private	throw
boolean	do	implements	protected	throws
break	double	import	public	transient
byte	else	instanceof	return	try
case	enum	int	short	void
catch	extends	interface	static	volatile
char	final	long	super	while
class	finally	native	switch	
const	for	new	synchronized	

I programmi Java devono essere scritti seguendo le regole dalla **sintassi** del linguaggio che specifica come combinare tra loro le parole chiave e gli identificatori in modo corretto.
Le regole della sintassi Java sono illustrate nei prossimi paragrafi.

5 Variabili e costanti

Le **variabili** sono i contenitori con i quali è possibile memorizzare i dati. Le variabili possono cambiare valore durante l'esecuzione del programma. Gli elementi che caratterizzano una variabile sono il *tipo*, il *nome* e la sua *visibilità*.

In Java la dichiarazione di una variabile assume la seguente forma:

<tipo> <nome variabile>;

Se, per esempio, si vuole dichiarare una variabile che deve contenere un valore numerico, si deve scrivere:

```
int prezzo;
```

Durante la dichiarazione delle variabili è possibile inserire il valore iniziale da assegnare ad esse.
L'**assegnamento** di un valore di inizializzazione a una variabile si ottiene usando il segno uguale (=).

<tipo> <nome variabile> = <valore iniziale>;

Per esempio:

```
double altezza = 1.83;
```

Una **costante**, a differenza delle variabili, può assumere un solo valore durante tutta l'esecuzione del programma.
Per dichiarare una costante, bisogna far precedere alla dichiarazione la parola chiave **final**.
Il valore di conversione tra la *yard* (0.914 metri) e il *metro* può essere espresso con la costante

```
final double YARD_METRO = 0.914;
```

Se si tenta di modificare il valore della costante, viene generato un errore di compilazione. Si noti che, per convenzione, il nome delle costanti viene indicato tutto in lettere maiuscole e se si compone di più parole queste vengono unite con un simbolo di sottolineatura.

Non esiste un punto preciso del programma dove dichiarare le variabili. Per comodità e leggibilità del codice conviene raggruppare le dichiarazioni di tutte le variabili nella parte iniziale del blocco di programma in cui verranno utilizzate. In linea di principio ogni variabile può essere dichiarata in qualsiasi punto, ma a seconda della posizione, cambia la sua **visibilità** (*scope*). Il campo di visibilità di una variabile è costituito dal blocco dentro al quale è stata dichiarata. All'esterno di questo blocco non viene riconosciuta. Inoltre, dopo la dichiarazione di una variabile, non è possibile dichiarare altre variabili con lo stesso nome, anche all'interno di blocchi annidati. In pratica *non* si possono verificare situazioni come la seguente:

```
{
  int a;
  . . .
  {
    int a;
  }
  . . .
}
```

6 Tipi di dato

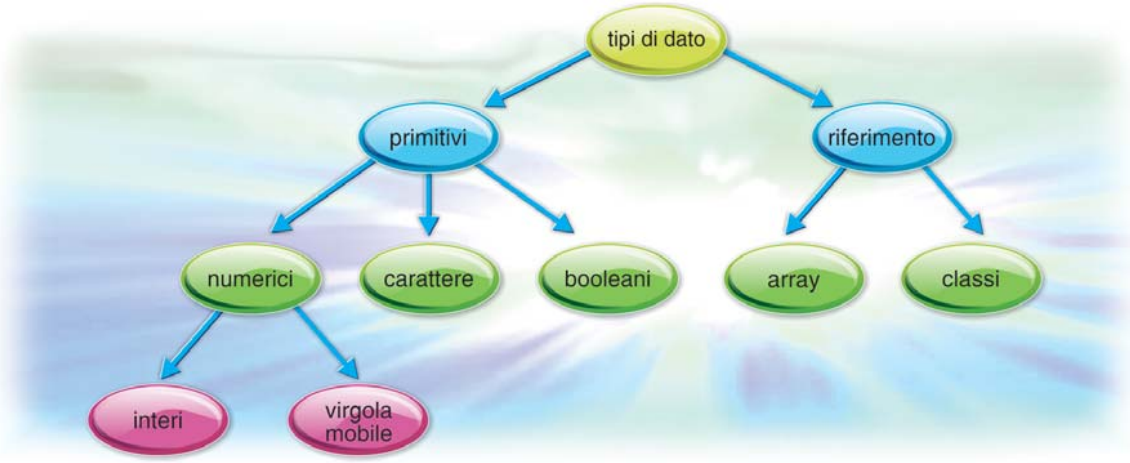
Il **tipo di dato** stabilisce qual è l'insieme di valori e operazioni accettabili da una variabile. In Java si distinguono due tipi di dato:

- tipi primitivi
- tipi riferimento.

I tipi di dato **primitivi** sono analoghi a quelli offerti da tutti i linguaggi di programmazione. Possono essere raggruppati in tre categorie: numerici, carattere e booleani.

I tipi di dato **riferimento** sono gli array e le classi, che verranno presentati più avanti.

Schematizzando la classificazione dei tipi di dato presenti in Java, si ottiene la seguente rappresentazione gerarchica.



I **tipi numerici** sono usati per dichiarare le variabili che possono contenere i numeri.
Questi tipi si possono distinguere a loro volta in tipi *interi* e tipi non interi in *virgola mobile* (o numeri in notazione esponenziale).

Tipi interi

Tipo	Dimensione	Valori
Byte	8 bit	da -128 a 127
Short	16 bit	da -32.768 a 32.767
Int	32 bit	da -2.147.483.648 a 2.147.483.647
Long	64 bit	da - 2^{63} a 2^{63}-1

I **tipi interi**, a seconda della loro occupazione di memoria (*dimensione*), identificano un intervallo di valori interi che una variabile può assumere. Per esempio:

```
byte numeroLotto;
```

I valori che può assumere la variabile *numeroLotto* sono compresi nell'intervallo -128..127. Questo perché il tipo *byte*, come è indicato nella tabella, richiede un'occupazione di memoria di 8 bit. Con 8 bit è possibile identificare 256 numeri differenti (2^8 = 256).
In generale l'intervallo di valori ammissibile per un tipo intero varia da -2^{n-1} a 2^{n-1}-1, avendo indicato con *n* il numero di bit occupati dal tipo.
In Java non esistono i tipi *unsigned*, cioè tipi che contengono solo valori positivi.

Tipi a virgola mobile

Tipo	Dimensione	Precisione
Float	32 bit	singola
Double	64 bit	doppia

Esistono due **tipi numerici a virgola mobile**: usando *float* si definiscono numeri a singola precisione (con 7 cifre decimali), usando *double* i numeri sono a doppia precisione (con 16 cifre decimali).

Per esempio:

```
double peso = 8.573;
```

Si noti che il numero è stato scritto usando anche il valore dei decimali. Indicare un numero in questo modo, fa capire al compilatore che si tratta di un numero a doppia precisione. Per indicare un numero a singola precisione si deve aggiungere, dopo il numero, la lettera **f**. Per esempio: 8.573f.

Il **tipo di dato carattere** è indicato con la parola chiave **char**. È definito usando 16 bit e la codifica *Unicode*. Java non utilizza la codifica ASCII per rappresentare i caratteri.

La codifica **Unicode** definisce un insieme di caratteri usando due byte.
I primi 128 caratteri dell'insieme *Unicode* sono equivalenti ai caratteri nel codice ASCII.

Per esempio:

```
char lettera = 'N';
```

Alcuni caratteri, che non possono essere digitati, vengono rappresentati attraverso le **sequenze di escape**. Queste sono formate da un *backslash* (****) seguito da una lettera o da un simbolo. Alcuni esempi di *sequenze di escape* sono:

\n ritorno a capo
\t tabulazione
**** *backslash*
\" doppio apice

Le **stringhe** sono un tipo di dato che raggruppa un insieme di caratteri. Le stringhe non sono un tipo predefinito, ma sono definite da una apposita classe (*String*), come si vedrà nel capitolo successivo.
Le *sequenze di escape* possono essere usate all'interno di una stringa per stampare dei caratteri speciali.
Per esempio:

```
System.out.println("prima riga \n\t seconda riga indentata");
```

Il **tipo di dato booleano** è indicato con la parola chiave **boolean**. Le variabili dichiarate in questo modo possono assumere due soli valori: *true* e *false*.
Per esempio:

```
boolean trovato = true;
```

Questo tipo di dato non può essere convertito in un tipo di dato numerico.

7 Il *casting* per la conversione di tipo

Il linguaggio Java richiede molta attenzione sull'uso dei tipi di dato. Il compilatore controlla che gli assegnamenti vengano effettuati tra variabili dello stesso tipo o di un tipo compatibile. Quasi tutti questi controlli di correttezza di tipo vengono eseguiti durante la fase di compilazione. Si dice anche che Java è fortemente *tipizzato*.
Se per esempio si vuole assegnare un valore intero a una variabile di tipo *float*, Java effettua automaticamente la conversione perché il tipo *float* è più grande del tipo intero. Questa conversione, chiamata anche **promozione**, è permessa perché solitamente non si ha perdita di dati.

Per esempio:

```
int intNum;
float floatNum;

intNum = 32;
floatNum = intNum;
```

La variabile *floatNum* può ricevere un valore *int* che viene automaticamente convertito in *float*. La conversione da *float* a intero, invece, non viene eseguita automaticamente.

Quando si tenta di effettuare assegnamenti tra variabili ed espressioni con tipi che sono in conflitto, viene generato un errore. Per esempio, usando in un programma le variabili precedenti:

```
floatNum = 13.278f;
intNum = floatNum;
```

il secondo assegnamento, durante la fase di compilazione, restituisce un errore. Non viene eseguita automaticamente la conversione tra *float* e *int*.
Se si vuole eseguire l'assegnamento, il programmatore deve indicare esplicitamente la sua volontà, forzando la conversione.

Il **casting** è il meccanismo che consente al programmatore di indicare la conversione da un tipo di dato a un altro tipo.

Il casting si effettua in pratica anteponendo al valore da convertire, tra parentesi tonde, il tipo di dato in cui sarà convertito.

Nell'esempio precedente, il casting viene effettuato aggiungendo **(int)** prima della variabile **floatNum**.

```
floatNum = 13.278f;
intNum = (int) floatNum;
```

In questo caso la conversione produce le perdita di tutte le cifre decimali. Nella variabile *intNum* resta memorizzato solo il valore intero 13.
Il *casting esplicito* è previsto per evitare la perdita di dati nella conversione tra tipi.
Java non consente la conversione automatica tra tipi che comportano una perdita di dati, ma lascia la possibilità di effettuare la conversione se viene esplicitamente richiesta.

Il casting è uno strumento utile anche quando all'interno di un'espressione si usano tipi di dato diversi.
Se si dividono due numeri interi, il risultato della divisione è un numero intero. Quindi eseguendo 20/6 il risultato è 3, che è un risultato non corretto.

```
int numeratore = 20;
int denominatore = 6;
double risultato;
risultato = numeratore / denominatore;
```

La variabile *risultato* contiene il valore 3.0.

Per ottenere il risultato corretto, con le cifre decimali, si deve effettuare un casting del numeratore prima di fare la divisione.

```
risultato = (double) numeratore / denominatore;
```

Con questa modifica, alla variabile *risultato* viene assegnato il valore 3.3333333333333335.

8 Operatori

Gli **operatori** sono i caratteri speciali che identificano particolari operazioni sui dati. Per esempio, si possono usare gli operatori matematici per le operazioni tra i numeri oppure l'operatore di concatenazione tra le stringhe.

Operatori del linguaggio Java

=	>	<	!	~	?:					
==	<=	>=	!=	&&	||	++	–	–		
+	-	*	/	&	|	^	%	<<	>>	>>>
+=	-=	*=	/=	&=	|=	^=	%=	<<=	>>=	>>>=

Le **operazioni aritmetiche** di base possono essere eseguite con i quattro operatori fondamentali: **+**, **-**, *****, **/**.
L'operatore **%** è usato per calcolare il **resto** della divisione tra due numeri.
Per esempio:

```
int num1 = 25;
int num2 = 7;
int modulo;

modulo = num1 % num2;
```

Alla variabile *modulo* viene assegnato il valore 4.

Se la formula da implementare si compone di più fattori è consigliabile l'utilizzo delle parentesi tonde per rendere chiaro e non ambiguo l'ordine di esecuzione dei vari operatori aritmetici.
Per esempio:

```
risultato = ((a + b) / (c - d)) * e;
```

Ci sono due operatori di **incremento** e di **decremento** che aggiungono o tolgono 1 al valore della variabile. Gli operatori sono **++** e **– –**.
L'istruzione

```
i++;
```

corrisponde all'istruzione

```
i = i + 1;
```

In Java l'assegnamento di un valore a una variabile viene eseguito usando l'**operatore di assegnamento =**.
Oltre a questo operatore esistono altri operatori di assegnamento. Sono tutti caratterizzati dal segno = preceduto da un altro operatore. Tra gli operatori composti di assegnamento ci sono: +=, –=, *=, /=, %=.
Questi operatori rappresentano la combinazione di un'operazione di calcolo e di un assegnamento.

Per esempio, l'istruzione

```
totale *= 10;
```

corrisponde all'abbreviazione della seguente:

```
totale = totale * 10;
```

Le stringhe possono essere concatenate tra loro usando l'**operatore di concatenazione +**. Esso consente anche di concatenare una stringa con un numero, eseguendo automaticamente la conversione del numero in stringa.
Questa possibilità è molto utile con il metodo *println* per l'output dei risultati, perché permette di stampare sia testo che numeri. Per esempio:

```
int sco = 20;
System.out.println("sconto = " + sco + "%");
```

NOTAZIONE PREFISSA E POSTFISSA

Gli operatori di incremento e di decremento possono essere posizionati dopo il nome della variabile (*postfissi*) oppure prima del nome della variabile (*prefissi*).
Il seguente esempio mostra il diverso comportamento: la posizione dell'operatore di incremento specifica se l'incremento di una variabile deve essere fatto prima o dopo il suo impiego nell'espressione di calcolo.

```
int num;
int risultato;

num = 10;
risultato = (num++) - 3;
```

La variabile *risultato* assume il valore 7, *num* diventa 11.
L'ultima istruzione è equivalente alle seguenti istruzioni:

```
risultato = num - 3;
num = num + 1;
```

Di seguito viene mostrato l'esempio con l'operatore prefisso.

```
num = 10;
risultato = (++num) - 3
```

In questo caso la variabile *risultato* assume il valore 8, *num* il valore 11.
L'ultima istruzione è equivalente alle seguenti istruzioni:

```
num = num + 1;
risultato = num - 3;
```

9 Commenti e documentazione

All'interno dei programmi Java possono essere inserite frasi contenenti **commenti** o annotazioni del programmatore, con le quali è possibile documentare il significato delle variabili o della costanti utilizzate, oppure la funzione svolta da una parte del programma.

In Java ci sono tre forme per rappresentare i commenti.

- La prima forma (commento di riga) utilizza i caratteri **//**. Tutto quello che viene scritto dopo questi caratteri e sulla stessa riga è considerato un commento. Questa modalità è usata per i commenti brevi. Per esempio:

```
int eta; // dichiarazione di una variabile intera
```

- La seconda forma di commento è la seguente **/* ... */**. Tutto il testo compreso tra i caratteri di inizio /* e quelli di termine */ viene considerato un commento. Questa forma consente di dividere il commento su più righe.

  Per esempio:

```
/* commento che occupa
più righe */
```

- La terza forma è **/** ... */**, dove al posto dei puntini va inserito un commento.

  Funziona come il commento su più righe. Rappresenta un commento di documentazione e può essere usato prima di dichiarare una classe o un metodo. Questa particolare forma di commento è importante perché viene riconosciuta dagli strumenti per la generazione automatica della documentazione dei programmi. Per esempio:

```
/** Funzione per calcolare il MCD tra due numeri */
public void calcolaMCD(int a, int b)
{
  // istruzioni
}
```

L'uso frequente delle frasi di commento deve diventare un'abitudine del programmatore fin dai primi semplici esercizi: in questo modo possono essere costruiti programmi che si autodocumentano, e che possono essere compresi e letti facilmente a distanza di tempo, anche da persone diverse dal programmatore che ha scritto il programma.

AUTOVERIFICA

Domande da 8 a 16 pag. 143-144
Problemi da 4 a 7 pag. 146

MATERIALI ON LINE

3. Operatori logici per le operazioni sui dati bit a bit
4. Strumenti per la generazione automatica della documentazione dei programmi

10 La gestione dell'input/output

In questo paragrafo vengono presentati gli elementi per le operazioni di input e output utilizzando un'interfaccia a caratteri.
Nei capitoli successivi vedremo l'uso dell'interfaccia grafica con Java, per costruire applicazioni che usano finestre e pulsanti.
In alcuni esempi precedenti è già stata presentata la modalità per scrivere sul video con istruzioni come la seguente:

```
System.out.println("messaggio da visualizzare");
```

System.out rappresenta un oggetto associato allo *standard output*. Esiste anche un oggetto associato allo *standard error* che è **System.err**. Entrambi fanno riferimento, in assenza di una diversa specifica (per *default*), allo schermo.
Println è un metodo. Sull'oggetto *System.out* viene invocato il metodo *println()*, con un parametro che rappresenta il valore da visualizzare. Il parametro può essere una stringa, un numero intero o reale, un carattere o un booleano. Quando il metodo *println()* viene richiamato, stampa sul video il parametro e inserisce un ritorno a capo.
Con l'oggetto *System.out* è possibile utilizzare anche il metodo **print()**: esso stampa il parametro passato, ma non stampa il ritorno a capo.
I metodi *print()* e *println()* ricevono un unico parametro, ma utilizzando l'operatore **+** è possibile concatenare stringhe e numeri.

PROGETTO 1

Scrivere un programma che consenta di stampare il valore di 3 variabili: un numero intero, un numero reale e un booleano.

Il programma dichiara 3 variabili di tipo diverso, alle quali vengono assegnati 3 valori. L'operazione di stampa visualizza i valori assegnati.

PROGRAMMA JAVA (*Stampa.java*)

```
/**
 * Programma per la stampa di tre variabili.
 */
class Stampa
{
  public static void main (String args[])
  {
    // dichiarazione delle variabili
    int intero = 5;
    float reale = 25.68f;
    boolean trovato = false;

    // operazioni di stampa
    System.out.println("Numero intero = " + intero);
    System.out.println("Numero reale = " + reale);
    System.out.println("Trovato = " + trovato);
  }
}
```

Il risultato dell'esecuzione di questo programma è:

```
Numero intero = 5
Numero reale = 25.68
Trovato = false
```

Nella dichiarazione del numero reale è stata aggiunta la lettera *f* dopo il numero: infatti il solo numero sarebbe interpretato come un numero di tipo *double* che non può essere assegnato a una variabile di tipo *float*. L'aggiunta della lettera *f* equivale ad effettuare un casting. In alternativa si può effettuare la dichiarazione anche in questo modo:

```
float reale = (float) 25.68;
```

Per le operazioni di **input** esiste un oggetto analogo **System.in** che gestisce il flusso di dati inseriti da tastiera. L'oggetto *System.in*, in pratica, viene mascherato con altri oggetti più potenti che forniscono maggiori funzionalità.

Si utilizza la classe **BufferedReader** nel seguente modo:

```
InputStreamReader input = new InputStreamReader(System.in);
BufferedReader tastiera = new BufferedReader(input);
```

Con queste dichiarazioni viene definito un oggetto *tastiera*, di classe *BufferedReader*, usando l'operatore **new** che crea un nuovo oggetto, come verrà spiegato più in dettaglio nel prossimo capitolo.
La classe *BufferedReader* mette a disposizione il metodo **readLine()** che consente di leggere, in questo caso da *standard input*, una riga per volta.
Una linea viene considerata terminata quando viene premuto il tasto di *Invio*.
Questo metodo restituisce solo stringhe, quindi se si vogliono acquisire valori numerici, si deve effettuare la conversione tra stringhe e numeri.

Per effettuare la **lettura di una stringa** dobbiamo dichiarare una variabile stringa e poi usare il metodo *readLine()* con l'oggetto *tastiera* che abbiamo definito in precedenza.

```
String nome;
nome = tastiera.readLine();
```

L'operazione deve essere completata considerando le eccezioni che possono essere generate dal metodo *readLine()*. L'**eccezione** segnala una situazione anomala. Quando si verifica un'eccezione bisogna prevedere un blocco di istruzioni per gestire questa situazione. Il blocco dell'eccezione è realizzato con il costrutto **try { } catch { }**.
Quindi la lettura di una stringa si effettua correttamente nel seguente modo:

```
String nome;
try
{
  nome = tastiera.readLine();
}
catch(Exception e) {}
```

Il segmento di codice precedente stabilisce che, al verificarsi dell'eccezione, non deve essere effettuata alcuna operazione.
Per effettuare la lettura di numeri, occorre convertire la stringa ottenuta dall'operazione di input nel seguente modo:

```
String leggiNumero;
int num;

try
{
  leggiNumero = tastiera.readLine();
  num = Integer.valueOf(leggiNumero).intValue();
}
catch(Exception e)
{
  System.out.println("\nNumero non corretto!");
  return;
}
```

L'istruzione aggiuntiva consente di convertire una stringa in un numero intero, attraverso due metodi della classe **Integer**: uno (*valueOf*) converte il parametro stringa in un oggetto di classe *Integer*, l'altro metodo (*intValue*) restituisce un valore intero.
In questo caso, l'eccezione che può essere generata si verifica quando i caratteri inseriti non corrispondono a cifre numeriche. Se per esempio si inseriscono i valori 34f5 oppure 45.45, essi non vengono riconosciuti come interi e quindi provocano un'eccezione che causa la terminazione del programma.

Per leggere un numero a virgola mobile, occorre cambiare il modo di conversione.
Per esempio, se si vuole ottenere un valore *float*, si deve scrivere:

```
numFloat = Float.valueOf(leggiNumero).floatValue();
```

In modo del tutto analogo, per ottenere un valore *double*:

```
numDouble = Double.valueOf(leggiNumero).doubleValue();
```

PROGETTO 2

Leggere da tastiera l'età di tre persone e calcolare l'età media.

Il programma acquisisce da tastiera le età con tre operazioni di input. Viene poi calcolata la media sommando i 3 numeri e dividendo per 3. Alla fine si visualizza il risultato del calcolo.

Algoritmo in pseudocodifica

```
inizio
  immetti eta1
  immetti eta2
  immetti eta3
  calcola media = (eta1 + eta2 + eta3) /3
  scrivi media
fine
```

PROGRAMMA JAVA (*Media.java*)

```
import java.io.*;

class Media
{
  public static void main(String argv[])
  {
    // impostazione dello standard input
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);

    // dichiarazione delle variabili
    int eta1, eta2, eta3;
    int media;

    System.out.println("Persona 1 ***************");
    System.out.print("eta': ");
    try
    {
      String numeroLetto = tastiera.readLine();
      eta1 = Integer.valueOf(numeroLetto).intValue();
    }
    catch(Exception e)
    {
      System.out.println("\nNumero non corretto!");
      return;
    }

    System.out.println("Persona 2 ***************");
    System.out.print("eta': ");
    try
    {
      String numeroLetto = tastiera.readLine();
      eta2 = Integer.valueOf(numeroLetto).intValue();
    }
    catch(Exception e)
    {
      System.out.println("\nNumero non corretto!");
      return;
    }

    System.out.println("Persona 3 ***************");
    System.out.print("eta': ");
    try
    {
      String numeroLetto = tastiera.readLine();
      eta3 = Integer.valueOf(numeroLetto).intValue();
    }
    catch(Exception e)
    {
      System.out.println("\nNumero non corretto!");
      return;
    }

    media = (eta1 + eta2 + eta3) / 3;
    System.out.println("\nEta' media: " + media);
  }
}
```

La prima riga del programma (*import java.io.*;*) serve per richiamare la libreria **java.io**, che contiene le classi *InputStreamReader* e *BufferedReader* usate per leggere dallo standard input.
Quando si verifica un'eccezione durante la lettura, viene mostrato un messaggio di errore e l'istruzione *return* interrompe l'esecuzione del metodo *main*, causando la fine del programma.

AUTOVERIFICA

Problemi da 8 a 13 pag. 147

11 Le strutture di controllo: sequenza

La struttura di **sequenza** viene realizzata posizionando le istruzioni una di seguito all'altra e separandole con il punto e virgola. Ogni dichiarazione e istruzione deve terminare con il punto e virgola.

PROGETTO 3

Costruire un programma che, dopo aver acquisito l'ora del giorno espressa come numero di secondi, converta il valore dai secondi al formato ore:minuti:secondi.

Un'ora si compone di 60 minuti e ogni minuto è composto da 60 secondi.
Un'ora contiene quindi 3600 secondi.
Le ore 8:00 possono essere espresse in secondi con il valore 28800, ottenuto moltiplicando 8 x 60 x 60.
Avendo il totale dei secondi, per ottenere le ore si procede per divisioni successive, sfruttando l'operatore / per la divisione intera e l'operatore % per calcolare il resto della divisione.

Algoritmo in pseudocodifica

```
inizio
    immetti orario
    calcola ore = orario / 3600
    calcola minuti = (orario % 3600) / 60
    calcola secondi = orario % 60
    scrivi orario HMS
fine
```

PROGRAMMA JAVA (*Orario.java*)

```
class Orario
{
  public static void main(String args[])
  {
    // dati di input
    int orario = 41345;

    // dati di output
    int ore, minuti, secondi;
```

```
    // calcolo sequenziale
    ore     = orario / 3600;
    minuti  = (orario % 3600) / 60;
    secondi = orario % 60;

    // stampa orario
    System.out.println("Orario: " + ore + ":"
                                  + minuti + ":"
                                  + secondi);
  }
}
```

12 Le strutture di controllo: selezione

La struttura di **selezione** consente di far eseguire particolari istruzioni in alternativa ad altre sulla base del risultato di una condizione. Se la condizione è vera viene eseguito un blocco di istruzioni, altrimenti ne viene eseguito un altro. I **blocchi** sono indicati con le parentesi graffe.

In Java la selezione è realizzata con il seguente costrutto:

```
if (condizione)
{
  // istruzioni eseguite se la condizione è vera
}
else
{
  // istruzioni eseguite se la condizione è falsa
}
```

Il blocco indicato da *else* può non essere presente.
Se il blocco è formato da una sola istruzione, si può evitare di mettere le parentesi graffe. Comunque l'uso delle parentesi, anche in presenza di una sola istruzione, serve per ottenere una maggiore leggibilità ed evitare possibili errori di interpretazione.
La condizione che segue l'*if* può essere una variabile di tipo *boolean* oppure un'espressione che contiene gli operatori di confronto oppure gli operatori booleani.

• Operatori di confronto
L'operatore di uguaglianza è rappresentato da due simboli di uguale (**==**). Si faccia attenzione alla differenza rispetto all'operatore di assegnamento **=**, che usa un solo segno di uguale.
Per esempio, con la seguente struttura di selezione, sul video viene scritto il messaggio *sufficiente* se il voto è uguale a 6.

```
if (voto == 6)
{
  System.out.println("sufficiente");
}
```

La disuguaglianza (*diverso da*) è espressa usando l'operatore **!=**.
Gli altri operatori di confronto sono **<**, **<=**, **>**, e **>=**.

• **Operatori booleani**
Gli operatori booleani vengono applicati a due valori booleani e restituiscono il risultato in base alle regole dell'algebra booleana.
L'operatore **&&** (oppure &) indica l'operazione di AND.
L'operatore **||** (oppure |) indica l'operazione di OR.
La negazione NOT viene espressa con l'operatore **!**.

I due operatori di AND e i due di OR forniscono gli stessi risultati. La differenza si nota in termini di efficienza.
L'espressione (*cond1 && cond2*) è equivalente a (*cond1 & cond2*): sono vere solo se entrambe *cond1* e *cond2* sono vere, mentre basta che una condizione sia falsa per rendere falsa l'espressione.
L'uso di && è più efficiente perché, durante la valutazione, se *cond1* è falsa, *cond2* non viene valutata, perché comunque l'espressione è sicuramente falsa. Al contrario l'operatore & valuta sempre entrambe le condizioni. È quindi utile usare come operatori booleani && e ||.

Per esempio:

```
if ((ora >= 12) && (ora <= 17))
{
  System.out.println("Buon pomeriggio");
}
else
{
  System.out.println("Buona giornata");
}
```

La struttura di **selezione multipla** consente di guidare l'esecuzione del programma scegliendo tra diverse alternative, a seconda del valore di una certa espressione.
L'espressione deve restituire un tipo intero oppure un carattere.
La struttura generale della selezione multipla in Java è la seguente:

```
switch (espressione)
{
  case valore1: // istruzioni
                break;
  case valore2: // istruzioni
                break;
  . . . .
  default: // istruzioni
           break;
}
```

L'espressione dopo la parola chiave **switch** solitamente è una variabile intera. All'interno del blocco si elencano i possibili valori che può assumere l'espressione, usando per ogni valore un blocco **case**.
Durante l'esecuzione, a seconda del valore dell'espressione, viene eseguito il blocco *case* associato. Se non viene trovato alcun valore nell'elenco dei *case*, si eseguono le istruzioni inserite dopo **default**. La parola chiave **break** serve per indicare la fine del blocco di istruzioni e fa terminare la selezione multipla.

Nel seguente frammento di programma viene mostrato come funziona il costrutto *switch* e come sia anche possibile unire tra loro più casi, associando ad essi un unico blocco di istruzioni.

```
int numero = (int) (Math.random() * 10);

System.out.println("E' stato estratto ");
switch (numero)
{
  case 1: System.out.println("il numero 1");
          break;
  case 0:
  case 2:
  case 4:
  case 6:
  case 8: System.out.println("un numero pari");
          break;

  default: System.out.println("un numero dispari diverso da 1");
           break;
}
```

Nell'esempio precedente è stata usata l'istruzione **Math.random()**, precisamente è stato invocato il metodo **random()** della classe **Math**. Questa classe contiene diversi metodi per eseguire le operazioni matematiche. Il metodo *random()* restituisce un numero casuale di tipo *double* compreso tra 0.0 e 1.0.
Moltiplicando il numero restituito per 10 si ottiene un numero compreso tra 0.0 e 10.0. Inoltre l'operazione di *casting* serve per convertire il numero casuale in un intero, troncando la parte decimale.

13 Le strutture di controllo: ripetizione

La struttura di **ripetizione** (o *ciclo* oppure *iterazione*, in inglese *loop*) consente di eseguire un blocco di istruzioni più volte. In Java ci sono tre modalità per rappresentare questa struttura di controllo.

- **while**

La prima struttura di controllo iterativa assume la seguente forma:

```
while (condizione)
{
  // istruzioni
}
```

Le istruzioni contenute nel blocco vengono eseguite mentre la condizione si mantiene vera. La condizione viene controllata prima di entrare nel ciclo, quindi se è falsa, le istruzioni non vengono eseguite nemmeno una volta.
Nel seguente esempio viene incrementata la variabile *a* mentre si mantiene minore di un certo valore: il programma restituisce tutte le potenze del numero *a* (2, 4, 8, 16, ...) inferiori a 1000.

```
int a=2;
while (a <= 1024)
{
  System.out.println(a);
  a *= 2;
}
```

• do while
La seconda struttura di controllo iterativa consente di effettuare il test sulla condizione alla fine del ciclo. In questo modo il blocco di istruzioni viene eseguito almeno una volta.
La struttura è la seguente:

```
do
{
  // istruzioni
}
while (condizione);
```

Le istruzioni vengono eseguite mentre la condizione si mantiene vera. L'ultima riga deve terminare con un punto e virgola.

• for
La terza struttura di controllo iterativa consente di eseguire la ripetizione con un contatore. Inoltre raggruppa all'inizio del ciclo le fasi di inizializzazione, aggiornamento e controllo della condizione.
La struttura generale è la seguente:

```
for (inizializzazione; condizione; aggiornamento)
{
  // istruzioni
}
```

Dopo la parola chiave *for* vanno inseriti, tra parentesi tonde, alcuni elementi separati dal punto e virgola.
La parte *inizializzazione* contiene il codice che viene eseguito una sola volta prima di entrare nel ciclo *for*. Solitamente contiene la dichiarazione di una variabile locale e la rispettiva inizializzazione.
La parte *condizione* contiene un'espressione booleana che viene valutata prima di eseguire il blocco di istruzioni. Se è falsa, non verranno mai eseguite le istruzioni presenti nel blocco. Se è vera, viene eseguito il ciclo.
Al termine del ciclo, vengono eseguite le istruzioni presenti nella parte *aggiornamento* e successivamente viene rivalutata la condizione per stabilire se la ripetizione deve continuare.
Per esempio, la seguente struttura *for* visualizza i primi 10 numeri naturali:

```
for (int i=1; i<=10; i++)
{
  System.out.println(i);
}
```

La variabile *i*, essendo stata dichiarata all'interno del *for*, è visibile solo nel blocco.

PROGETTO 4

Dati come input dieci voti di uno studente espressi con numeri, segnalare con un giudizio non numerico se il voto è negativo, sufficiente o positivo.

Algoritmo in pseudocodifica

```
inizio
  per i da 0 a numeroVoti–1
      immetti voto
      se voto < 6
      allora
            assegna testo = "negativo"
      altrimenti
            se voto == 6
            allora
                  assegna testo = "sufficiente"
            altrimenti
                  assegna testo = "positivo"
            fine se
      fine se
      scrivi testo
  ripeti
fine
```

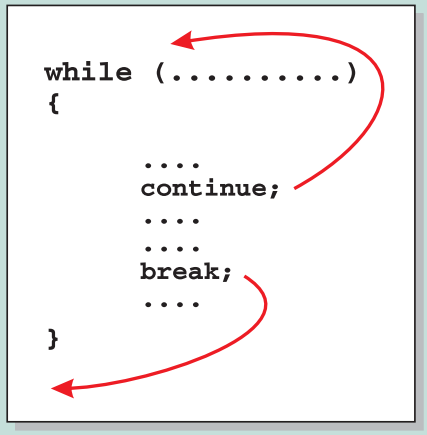
PROGRAMMA JAVA (*Voti.java*)

```
import java.io.*;

class Voti
{
  public static void main(String argv[])
  {
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);

    final int NUMERO_VOTI = 10;
    int voto;
    String testo;

    for(int i=0; i<NUMERO_VOTI; i++)
    {
      System.out.print("Voto numerico: ");
      try
      {
        String votoLetto = tastiera.readLine();
        voto = Integer.valueOf(votoLetto).intValue();
      }
      catch(Exception e)
      {
        System.out.println("Voto non valido.");
        // torna all'inizio del ciclo.
        i--;
        continue;
      }
```

```
      if (voto < 6)
      {
        testo = "negativo";
      }
      else if (voto == 6)
      {
        testo = "sufficiente";
      }
      else
      {
        testo = "positivo";
      }
      System.out.println("Hai ricevuto un voto " + testo);
    }
  }
}
```

CICLI INTERROTTI E CICLI INFINITI

Le istruzioni presenti nei blocchi delle strutture iterative vengono eseguite in sequenza. È però possibile modificare l'ordine di esecuzione usando due istruzioni: *break* e *continue*.
L'istruzione **break** causa l'interruzione anticipata della ripetizione, attivando l'esecuzione della prima istruzione presente dopo la struttura di controllo.
L'istruzione **continue** interrompe l'esecuzione del blocco e la fa riprendere dall'inizio del ciclo dopo aver valutato nuovamente la condizione.



Un programma Java può comunque essere interrotto in qualsiasi momento usando la combinazione di tasti **Ctrl + C**.
In particolare può capitare di scrivere un programma con strutture iterative che diventano dei *loop* infiniti, nei casi in cui la condizione del ciclo si mantiene sempre vera.
Se si esegue un loop infinito è possibile interrompere l'esecuzione del programma premendo la combinazione di tasti Ctrl + C.

Il seguente frammento di programma è un esempio di loop infinito:

```
while (true)
{
  // istruzioni
}
```

Anche con la struttura iterativa *for* si possono generare dei loop infiniti, se non si fa attenzione alla correttezza dell'istruzione di uscita dal ciclo.
Per esempio,

```
for (int i=1; i!=10; i=i+2)
{
  // istruzioni
}
```

AUTOVERIFICA
Domande da 17 a 22 pag. 144-145
Problemi da 14 a 25 pag. 147-148

MATERIALI ON LINE

5. Operatore di selezione ternario

14 La struttura di dati array

Quando si devono memorizzare più oggetti che sono dello stesso tipo, si possono utilizzare gli **array** a una dimensione (*vettori*) o a più dimensioni (*matrici*).
Un array è realizzato con un riferimento che punta all'area di memoria dove si trovano i suoi elementi.
Per indicare l'array, cioè un gruppo di elementi dello stesso tipo, si usa un unico nome collettivo. Ogni elemento è individuato attraverso un indice (numero intero).
Il primo elemento dell'array ha come indice 0. Il numero di elementi che compongono un array costituisce la *dimensione* dell'array. Quindi la posizione dell'ultimo elemento di un array è individuata dall'indice ottenuto decrementando la dimensione di 1.
Durante l'esecuzione del programma vengono controllati gli indici degli array e se non sono validi, cioè hanno superato i limiti consentiti, viene segnalato un errore.

Per poter creare e usare un array si devono seguire tre passaggi:
- dichiarazione dell'array
- allocazione
- eventuale inizializzazione.

La **dichiarazione di un array** consiste nella specificazione del suo tipo. Il tipo può essere un tipo predefinito di Java (per esempio *int*, *float*) oppure può essere una classe (per esempio *String*, *Automobile*). Si possono creare array che contengono un insieme di oggetti tutti della stessa classe.
Come per le variabili, la dichiarazione è fatta indicando un tipo e poi il nome da associare all'array. Però subito dopo il nome sono presenti le parentesi quadre, per indicare che è un array. Le parentesi potrebbero essere messe anche dopo il tipo, nel seguito si userà sempre la dichiarazione con le parentesi dopo il nome.

Per esempio:

```
int i[];
String nomi[];
```

L'array *i* è un array di interi. L'array *nomi* è un array di oggetti *String*. Si osservi che la sola dichiarazione non specifica quale sia la dimensione dell'array. L'array, a questo punto, non è ancora stato creato e assume il valore speciale **null**.

L'**allocazione di un array** consente di specificare quanto spazio riservare per l'array, cioè quanti elementi saranno memorizzati al suo interno. L'allocazione viene eseguita utilizzando l'operatore **new**. È lo stesso operatore che sarà usato successivamente anche per allocare gli oggetti.
L'allocazione delle variabili dell'esempio precedente viene fatta con le seguenti istruzioni:

```
i = new int[5];
nomi = new String[10];
```

L'operatore *new* è seguito dal tipo dell'array e dalla dimensione, specificata tra parentesi quadre. In questo caso è stato allocato l'array *i* per contenere 5 interi. L'array *nomi* può contenere 10 oggetti *String*.
Dopo la fase di allocazione, si può dire che l'array è stato creato e può essere utilizzato, assegnando i valori ai suoi elementi.
È possibile riferirsi ai singoli elementi dell'array utilizzando il nome e un numero che indica la posizione all'interno dell'array. Il numero usato è l'**indice** dell'array.
Gli elementi dell'array *i* sono accessibili nel seguente modo:

i[0], i[1], i[2], i[3], i[4]

Come si vede, si parte con l'indice 0 per indicare il primo elemento e si termina con l'indice dato dalla dimensione meno 1, per indicare l'ultimo elemento. Essendo 5 la dimensione dell'array, l'ultimo elemento è rappresentato dall'indice 4.
Se si cerca di accedere a un elemento che non fa parte dell'array, per esempio *i[5]*, viene generata l'eccezione **ArrayIndexOutOfBoundsException** e il programma non viene eseguito correttamente.
Per assegnare i valori agli elementi di un array si procede così:

```
i[0] = 45;
i[1] = 12;
i[2] = 3;

nomi[0] = "Francesca";
```

Questi assegnamenti, se vengono eseguiti subito dopo la dichiarazione e l'allocazione, rappresentano l'**inizializzazione dell'array**.
Un array mantiene al suo interno l'informazione della sua dimensione. Per richiamare questa informazione si fa seguire al nome dell'array un punto e la parola **length**.

Per esempio:
i.length restituisce il numero 5
nomi.length restituisce 10.

Questo modo di indicare la dimensione dell'array è utile all'interno dei cicli ed evita di scrivere esplicitamente la dimensione attraverso un numero.

Per esempio, con il seguente ciclo *for*, si assegna a ogni elemento dell'array *i* il valore corrispondente al quadrato del suo indice.

```
for (int indice=0; indice < i.length; indice++)
{
  i[indice] = indice*indice;
}
```

Java consente di unire le due fasi di dichiarazione e di allocazione per rendere più veloce la creazione di array.

La **dichiarazione e allocazione degli array** possono essere fatte contemporaneamente sulla stessa riga:

```
int i[] = new int[5];
String nomi[] = new String[10];
```

È possibile anche dichiarare, allocare e allo stesso tempo inizializzare un array, inserendo tra parentesi graffe i valori che deve contenere. La dimensione dell'array corrisponde al numero di elementi racchiusi tra parentesi graffe.

```
int i[] = {45, 12, 3, 10, 2000};
```

PROGETTO 5

Il metodo *main* riceve come parametro un array di stringhe chiamato *args[]*. In questo array sono contenuti i parametri che vengono passati al programma dalla riga di comando. Scrivere un programma che stampa tutti i parametri che vengono passati.

Il programma deve controllare se ci sono parametri forniti dall'utente insieme al comando. In caso positivo, ne visualizza il valore con una ripetizione che va da 0 a *args.length* – 1, essendo *args.length* la dimensione dell'array *args[]*.

PROGRAMMA JAVA (*Parametri.java*)

```
class Parametri
{
  public static void main(String args[])
  {
    System.out.println("Elenco parametri:");

    if (args.length == 0)
    {
      System.out.println("nessun parametro");
      return;
    }
    for(int i=0; i<args.length; i++)
    {
      System.out.println(args[i]);
    }
  }
}
```

Se si esegue il programma da linea comandi con i seguenti tre parametri:

```
java Parametri 45 -t Rossi
```

si ottiene come risultato:

```
Elenco parametri:
45
-t
Rossi
```

15 Gli array multidimensionali

Gli array considerati finora corrispondono in matematica ai *vettori*.
Esistono altri tipi di array chiamati *multidimensionali*, in particolare l'array bidimensionale, che corrisponde in termini matematici a una **matrice** con righe e colonne. Ogni elemento della matrice è individuato da una coppia di numeri di cui il primo indica la riga e il secondo la colonna. Lo schema seguente presenta una matrice di dimensione 3x4, all'interno di ogni cella sono indicati gli indici di riga e di colonna.

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3

In Java la dichiarazione e l'allocazione di un array a due dimensioni procede allo stesso modo della dichiarazione di un array monodimensionale: occorre specificare la nuova dimensione usando un secondo paio di parentesi quadre.

```
int matrice[][];
matrice = new int[3][4];
```

Per l'assegnamento di valori alla matrice bisogna riferirsi a uno specifico elemento della matrice indicando la sua posizione con una coppia di indici. Se, per esempio, si vuole assegnare il valore 10 al primo elemento della matrice si usa la seguente istruzione:

```
matrice[0][0] = 10;
```

PROGETTO 6

Dopo aver costruito una matrice 3x3, inizializzata con valori casuali, calcolare la sua matrice trasposta.

La matrice *trasposta* è una matrice avente le stesse dimensioni, ma con righe e colonne scambiate tra loro.

Algoritmo in pseudocodifica

```
inizio
    per riga da 0 a 2
        per colonna da 0 a 2
            assegna matr[riga][colonna] = valore causale compreso tra 0 e 9
```

```
            scrivi matr[riga][colonna]
        ripeti
    ripeti
    per riga da 0 a 2
        per colonna da 0 a 2
            assegna trasp[riga][colonna] =  matr[colonna][riga]
            scrivi trasp[riga][colonna]
        ripeti
    ripeti
fine
```

PROGRAMMA JAVA (*Trasposta.java*)

```
import java.io.*;

class Trasposta
{
  public static void main(String args[])
  {
    // dichiarazione e allocazione di due matrici 3x3
    int matr[][] = new int[3][3];
    int trasp[][] = new int[3][3];

    System.out.println("Matrice *******************");

    // inizializza casualmente la matrice
    for(int riga=0; riga<3; riga++)
    {
      for(int colonna=0; colonna<3; colonna++)
      {
        matr[riga][colonna] = (int) (Math.random()*10);
        System.out.print(matr[riga][colonna]+" ");
      }
      System.out.println();
    }

    System.out.println("\nMatrice trasposta *********");

    // calcola la matrice trasposta
    for(int riga=0; riga<3; riga++)
    {
      for(int colonna=0; colonna<3; colonna++)
      {
        trasp[riga][colonna] = matr[colonna][riga];
        System.out.print(trasp[riga][colonna]+" ");
      }
      System.out.println();
    }
  }
}
```

Il programma produce il seguente output:

```
Matrice ******************
9 1 6
5 5 4
1 9 8
Matrice trasposta *********
9 5 1
1 5 9
6 4 8
```

16 Le eccezioni

Un'**eccezione** è una situazione anomala che si verifica durante l'esecuzione del programma. Una condizione anomala può essere una divisione per zero, oppure l'utilizzo di un indice in un array maggiore della dimensione dell'array.

Quando si riscontra un'eccezione, questa viene segnalata al programma, che può decidere quali azioni compiere.

Il programma, usando particolari costrutti, può tenere sotto controllo certe parti del codice e agire in modo opportuno se viene generata un'eccezione.

Se un'eccezione non viene gestita, il programma viene interrotto e viene segnalato un errore.

La gestione delle eccezioni, come già visto in precedenza, viene realizzata in Java con il costrutto **try ... catch ...**:

```
try
{
  // istruzioni da controllare
}
catch(eccezione)
{
  // operazioni da eseguire se si verifica l'eccezione
}
```

Se vengono generate delle eccezioni all'interno del blocco *try*, il controllo dell'esecuzione passa al blocco *catch*. Dopo la parola *catch* si indica, tra parentesi tonde, l'eccezione che può verificarsi e, di seguito tra parentesi graffe, le operazioni da eseguire per cercare di ripristinare la situazione corretta.

Ci sono alcune eccezioni predefinite:

- **ArithmeticException**: segnala errori aritmetici,
- **NullPointerException**: errore dovuto all'utilizzo di un riferimento che possiede il valore *null*,
- **IndexOutOfBoundException**: errore nell'indice di un array,
- **IOException**: generico errore di input/output.

Tutte le eccezioni sono sottoclassi della classe **Exception**.

PROGETTO 7

Eseguire una divisione con divisore uguale a zero e gestire l'eccezione generata.

Un'implementazione banale della divisione per zero può essere realizzata con il seguente programma.

```
class Eccezione
{
  public static void main(String args[])
  {
    int a;
    int divisore = 0;
    a = 15 / divisore;
  }
}
```

Eseguendo il programma, viene segnalata un'eccezione dovuta alla divisione per zero (*ArithmeticException*). Poiché l'eccezione non viene gestita, il programma termina.
Per gestire questa eccezione bisogna aggiungere il costrutto *try...catch...*, modificando il programma precedente come segue:

PROGRAMMA JAVA (*Eccezione.java*)

```
class Eccezione
{
  public static void main(String args[])
  {
    int a;
    int divisore = 0;

    try
    {
      a = 15 / divisore;
    }
    catch(ArithmeticException e)
    {
      System.out.println("Divisione impossibile");
      return;
    }
  }
}
```

Quando si verifica l'eccezione, il programma visualizza un messaggio di errore e termina l'esecuzione.

AUTOVERIFICA

Domande da 23 a 28 pag. 145-146
Problemi da 26 a 31 pag. 148
Problemi di riepilogo da 32 a 44 pag. 149

MATERIALI ON LINE

6. L'array args[] per i valori passati da linea di comando
7. Valutazione di un polinomio di terzo grado
8. Programmi eseguibili e videate dei progetti del capitolo

AUTOVERIFICA

DOMANDE

Programmare in Java

1 Quali di queste affermazioni, riguardanti le caratteristiche principali del linguaggio Java, sono vere (V) e quali false (F)?

a) È un linguaggio orientato agli oggetti V F
b) Permette di realizzare applicazioni portabili V F
c) Genera programmi solo per la piattaforma Linux V F
d) Gestisce automaticamente la memoria (allocazione e deallocazione) V F

2 Quali di queste affermazioni, riguardanti il *bytecode*, sono vere (V) e quali false (F)?

a) Non è un codice portabile V F
b) È il risultato della compilazione di un programma Java V F
c) Può essere eseguito direttamente V F
d) Deve essere interpretato per essere eseguito V F
e) È contenuto nei file con estensione .class V F

3 Attraverso quale modalità operativa i programmi scritti in Java sono portabili, cioè eseguibili senza modifiche su piattaforme diverse ?

a) interpretazione
b) editing
c) compilazione
d) debugging

4 Qual è il corretto significato dell'acronimo API ?

a) Active Processing Interface
b) Active Programming Institute
c) Application Processing Institute
d) Application Programming Interface

5 Quali di queste dichiarazioni corrisponde alla corretta dichiarazione del metodo *main*?

a) public static void main (String args)
b) public void main (String args[])
c) public static void main (String args[])
d) public static void main ()

6 Qual è l'estensione di un file sorgente di Java?

a) .class
b) .txt
c) .cpp
d) .java

7 Associa ad ogni operazione della colonna a sinistra il corrispondente programma o comando scegliendo nella colonna a destra.

a) editare il codice Java	1) javac
b) compilare il codice Java	2) java
c) eseguire il programma Java	3) Blocco note

Elementi base del linguaggio

8 Per ognuno dei nomi elencati, indicare se è un identificatore valido in Java (si/no).
a) eta media .....
b) 1peso .....
c) COLORE_SFONDO .....
d) Velocita-max .....
e) NumeroAbbonati .....
f) Città .....

9 Qual è la forma della dichiarazione di una variabile con assegnamento di un valore iniziale?
a) <tipo> <nome variabile> = <valore iniziale>;
b) <nome variabile> <tipo> = <valore iniziale>;
c) <nome variabile> = <valore iniziale> <tipo>;
d) <tipo> <valore iniziale> = <nome variabile>;

10 Qual è la forma della dichiarazione di una costante di tipo carattere ?
a) final char RISPOSTA_POSITIVA = Y;
b) char final RISPOSTA_POSITIVA = 'Y';
c) final int RISPOSTA_POSITIVA = 'Y';
d) final char RISPOSTA_POSITIVA = 'Y';

11 Quale tra i seguenti tipi di dato messi a disposizione da Java non fa parte dei tipi di dato primitivi ?
a) numerici
b) carattere
c) array
d) booleani

12 Associa a ogni tipo di dato numerico a sinistra la corrispondente occupazione di memoria:

a) float	1) 8 bit
b) int	2) 16 bit
c) byte	3) 32 bit
d) double	4) 32 bit
e) long	5) 64 bit
f) short	6) 64 bit

13 Quali di queste affermazioni, riferite alla conversione tra tipi, sono vere (V) e quali false (F)?
a) La conversione dal tipo *float* al tipo *int* viene eseguita automaticamente da Java V F
b) La conversione dal tipo *float* al tipo *int* comporta una perdita di informazione V F
c) La conversione dal tipo *float* al tipo *int* è chiamata promozione V F
d) Dividendo un numero di tipo *int* per un *int* si ottiene un numero di tipo *float* V F
e) La conversione dal tipo *int* al tipo *float* viene eseguita con il casting V F

14 Quali di queste affermazioni, riferite al concetto di *casting* in Java, sono vere (V) e quali false (F)?
a) È sempre eseguito automaticamente V F
b) In alcuni casi deve essere il programmatore a richiederlo V F
c) È uno strumento per convertire un tipo di dato in un altro V F
d) La compilazione individua eventuali errori di casting V F
e) Il casting dal tipo *double* al tipo *int* è automatico V F

AUTOVERIFICA

AUTOVERIFICA

15 Supponendo che la variabile *a* di tipo intero assuma ogni volta il valore iniziale 5, quale valore assume alla fine di ogni gruppo di istruzioni ?

a)
```
a += 4;
b = a – 8;
a = 10 / a;
```
.....

b)
```
b = a * 2;
a -= b;
a = 3;
```
.....

c)
```
b = 18 % a;
a *= 2;
a = a + b;
```
.....

16 Indica il valore assunto dalla variabile *risultato* al termine di ogni gruppo di operazioni.

a)
```
risultato = 0;
num = 5;
risultato = (num- -) + 3;
```
.....

b)
```
risultato = 0;
num = 5;
risultato = (- - num) + 3;
```
.....

Strutture di controllo

17 Quali di queste affermazioni, riferite alle strutture di controllo Java, sono vere (V) e quali false (F)?

a) if (...) {...} else {...} è una struttura di selezione V F
b) for (...) {...} è una struttura di selezione V F
c) while (...) {...} è una struttura di iterazione V F
d) switch (...) {case: ...break;} è una struttura di iterazione V F
e) try {...} catch (...) {...} è una struttura di iterazione V F

18 Come vengono tradotti in Java gli operatori logici AND, OR e NOT?

Operatore logico	Operatore in Java
AND	
OR	
NOT	

19 Supponendo che la variabile *x* di tipo intero assuma ogni volta il valore iniziale 3, quale valore assume alla fine dei tre segmenti di codice ?

a)
```
if (x > 3)
{
  x *= 2;
}
else
{
  x += 2;
}
```

b)
```
if (x == 3)
{
  x = 5;
}
if (x <= 5)
{
  x -= 2;
}
```

c)
```
if (x >= 3)
{
  x++;
  if (x > 4)
  {
    x = 10;
  }
}
```

20 Quale valore assume la variabile *y* alla fine del segmento di programma?

```
int y=0;
int a=2;
while (a <= 1024)
{
  y++;
  a *= 2;
}
```

21 Quale valore assume la variabile *y* alla fine del segmento di programma?

```
int y=1;
boolean fine;
do
{
  y++;
  fine = (y > 10);
}
while (!fine);
```

22 Quale valore assume la variabile *y* alla fine del segmento di programma?

```
y=30;
for (int i=0; i <= y; i=i+2)
{
  y -= 5;
}
```

Array

23 Metti in ordine, indicando un valore da 1 a 3, i passaggi che si devono seguire per la creazione di un array.

Passaggi per la creazione	Ordine
Inizializzazione	
Dichiarazione	
Allocazione	

24 Qual tra le seguenti dichiarazioni è corretta per allocare un array di 10 elementi?
a) valori[10] = new double[];
b) valori[] = new double[10];
c) valori = new double[10];
d) valori = double[10];

25 Qual è l'errore contenuto nella dichiarazione di un array: int lista[50];?
a) Doveva essere int lista[50] = new int[];
b) L'array non è stato inizializzato
c) Non si può indicare la dimensione dell'array nella dichiarazione
d) Non si può dichiarare un array di tipo int

26 Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.
a) Si può indicare la dimensione dell'array usando la parola .....
b) Per accedere ad un elemento dell'array si indica l'indice tra parentesi .....
c) Se si usa un indice maggiore della dimensione dell'array viene generata l'eccezione .....
d) Un array dichiarato ma non ancora allocato assume il valore speciale .....
graffe, tonde, dimens, ArrayIndexOutOfBoundsException, length, ArrayLimitException, null, int, quadre, void

27 Qual tra le seguenti dichiarazioni è corretta per assegnare all'ultimo elemento di una matrice 4x4 il valore 99 ?
a) matrice[matrice.length] = 99;
b) matrice[0][0] = 99;
c) matrice[3][3] = 99;
d) matrice[4][4] = 99;

AUTOVERIFICA

Eccezioni

28 Quali di queste affermazioni, riferite al concetto di *eccezione*, sono vere (V) e quali false (F)?

a) È una situazione anomala che può verificarsi durante l'esecuzione V F
b) Si verifica in fase di compilazione V F
c) Se non gestita causa la terminazione del programma V F
d) È controllata con il blocco try … catch … V F

PROBLEMI

Programmare in Java

1 Effettuare un collegamento al sito *http://www.oracle.com/technetwork/java/index.html* per individuare la versione più recente del JDK (*Java Development Kit*). Accedere alla sezione Java SE (*Standard Edition*), fare il download del JDK ed eseguire l'installazione sul computer.

2 Scrivere, usando *Blocco Note*, il seguente programma che calcola la somma di due numeri e salvarlo con il nome appropriato.

```
class Somma
{
  public static void main(String args[])
  {
    int a = 3;
    int b = 8;
    int somma = a + b;
    System.out.println(somma);
  }
}
```

3 Compilare ed eseguire il programma precedente.

Elementi base del linguaggio

4 Scrivere un programma contenente la seguente dichiarazione di una variabile intera e verificare quale errore viene generato in fase di compilazione.
int volume = 15.0;

5 Scrivere un programma che dichiara una costante intera *percIva* il cui valore è 20. Successivamente modificare il valore di *percIva* assegnando il valore 21. Verificare che, compilando il programma, viene visualizzato il messaggio di errore "*cannot assign a value to final variable percIva*".

6 Scrivere un programma che dichiara una variabile di nome *lunghezza* e di tipo *byte*. Assegnare alla variabile il valore 140. Verificare che, compilando il programma, viene visualizzato il messaggio di errore "*possible loss of precision*".

7 Calcolare quanti byte occupano le seguenti variabili:
- 5 variabili di tipo *float*
- 7 variabili di tipo *short*
- 4 variabili di tipo *double*
- 4 variabili di tipo *long*
- 10 variabili di tipo *int*
- 10 variabili di tipo *char*

Input e output

8 Scrivere un programma che restituisce il seguente output a video:

TITOLO:
"La divina commedia"
AUTORE
D. Alighieri

9 Scrivere un programma che legge da tastiera due valori di tipo *double* e esegue la divisione restituendo il risultato sullo schermo.

10 Scrivere un programma che legge da tastiera il nome e il cognome di una persona e stampa sul video il nome completo.

11 Scrivere un programma che legge da tastiera un numero e comunica con un messaggio se il numero è pari o dispari. Si ricorda che un numero è pari se il resto della divisione per 2 restituisce 0.

12 Scrivere un programma che legge da tastiera la misura del lato e calcola l'area e il perimetro di un quadrato.

13 Dato il prezzo di un prodotto e la percentuale di sconto, calcolare e visualizzare il prezzo scontato.

Strutture di controllo

14 Avendo la lunghezza di un'auto espressa con un valore in millimetri, costruire un programma per convertire il valore nel formato metri, centimetri tralasciando gli eventuali millimetri residui.

15 Supponendo che una rete televisiva abbia trasmesso 5.876 minuti di documentari in un anno, scrivere un programma per convertire il valore nel formato giorni, ore, minuti.

16 Avendo la variabile intera *h* e usando gli operatori, scrivere le condizioni booleane che siano vere quando:
- h si trova nell'intervallo [2, 7],
- h si trova nell'intervallo [-4, 3],
- h è uguale a 0 oppure è maggiore di 100,
- h non è negativo,
- h è diverso da 100.

17 Riscrivere la seguente struttura *while* usando la struttura iterativa *for*.

```
int c;
c = 8;
while (c > 0)
{
  c -= 2;
}
```

18 Calcolare quante volte viene eseguito il ciclo precedente.

19 Scrivere un programma che stampa i primi 100 numeri pari usando la struttura di ripetizione *while*.

AUTOVERIFICA

20 Scrivere un programma che stampa i primi 100 numeri pari usando la struttura di ripetizione *do while*.

21 Scrivere un programma che stampa i primi 100 numeri pari usando la struttura di ripetizione *for*.

22 Scrivere un programma che produce a video la tavola pitagorica.

23 Dato un elenco di 10 prodotti con descrizione e prezzo, visualizzare il prezzo massimo.

24 Indicare qual è il comportamento del seguente frammento di codice

```
int i;
for (i=1; i <=10; i = i - 1)
{
  System.out.println(i);
}
```

25 Verificare se il seguente frammento di codice rappresenta un ciclo infinito.

```
int i = 5;
while (i < 300)
{
  if (i < 100)
  {
    i++;
    break;
  }
  i--;
  continue;
}
```

Array

26 Dichiarare e allocare un array contenente 12 numeri a doppia precisione.

27 A partire dalla seguente definizione di array,

```
int tab[][] = new int[7][4];
```

scrivere due frammenti di codice per calcolare il totale di ogni riga e il totale di ogni colonna della matrice.

28 Scrivere un programma per definire un array di 5 numeri e per valorizzarli con 5 numeri casuali tra 1 e 90. Visualizzare successivamente i numeri scelti.

29 Dopo aver acquisito da tastiera un array di 10 numeri a singola precisione, sommare le sue componenti e visualizzare il risultato.

30 Dopo aver acquisito da tastiera due matrici 2x2 di numeri interi, calcolare la somma delle due matrici sommando le singole componenti.

31 Definire un array di dimensione 10 e successivamente provare ad accedere all'elemento di posizione 20 gestendo l'eccezione che viene generata.

PROBLEMI DI RIEPILOGO

La realizzazione dei seguenti programmi deve essere preceduta da un'analisi che soddisfi i seguenti punti:
- descrizione generale del problema
- dati di input e di output
- pseudocodifica
- programma Java.

32 Dati due numeri interi compresi tra 0 e 49, generati casualmente, visualizzare il numero più grande.

33 Generare cinque numeri casuali, reali e compresi tra 0 e 1, e calcolarne la media.

34 Una scuola è composta da N classi. Per ogni classe, viene inserito da tastiera il numero di studenti. Calcolare quanti studenti frequentano la scuola e in media quanti studenti ci sono per classe.

35 Scrivere un programma che ricevendo da linea di comando la base e l'altezza di un rettangolo, restituisce il valore dell'area.

36 Dato come input il tempo in ore, minuti e secondi, convertire in secondi. Per esempio 5 ore, 9 minuti, 24 secondi = 18564 secondi.

37 Dopo aver ottenuto da tastiera un numero decimale, calcolare la rappresentazione dello stesso numero nel sistema binario.

38 Scrivere un programma che continua a richiedere numeri finché viene inserito il valore zero. Alla fine indica quanti sono stati i numeri positivi e i numeri negativi inseriti.

39 Un anno è bisestile se è divisibile per 4 e non è divisibile per 100. Sono però bisestili anche gli anni divisibili per 400. Scrivere un programma che, inserendo un anno da tastiera, risponda se è un anno bisestile o no.

40 Un'urna contiene venti palline numerate da 1 a 20. Creare un programma che simuli un'estrazione casuale dall'urna e che consideri come vittoria l'estrazione di un numero pari.

41 Data la velocità espressa in Km/h calcolare la conversione in m/s.

42 Nella trasmissione di dati tramite reti di calcolatori, l'unità di misura utilizzata è il bit. Ricevuto in input un numero indicante i bit trasmessi, calcolare a quanti byte, Kb, Mb, Gb corrisponde. Per esempio 84.054.321 bit = 10.506.790 byte = 10.260,5 Kb = 10,02 Mb = 0,0098 Gb.

43 Scrivere un programma che calcola il prodotto tra due matrici A e B di dimensioni rispettivamente 3x2 e 2x3.

44 Problema di ricerca in un array: dopo aver generato un array di 30 numeri casuali tra 0 e 99, determinare se è presente il numero 50.

Focus notes

JAVA PROGRAMMING LANGUAGE

Variables

Java programming language has its own rules and conventions for naming variables:

- Variable names are case-sensitive;
- Variable characters are usually letters, digits or underscore characters;
- The name must not be a keyword or reserved word;
- If the name consists of only one word, that word is in all lowercase letters. If it consists of more than one word, the first letter of each subsequent word is capitalized.

Data types

The primitive data types supported by the Java programming language are:
Byte, 8-bit signed
Int, 32-bit signed
Long, 64-bit signed
Float, single-precision 32-bit
Double, double-precision 64-bit
Boolean, only two possible values, true and false
Char, single 16-bit Unicode character.
In addition the Java programming language provides support for character *strings* via the *java.lang.String* class.
An *array* is a data container that holds a fixed number of values of a single type. The length of an array is established when the array is created.
Each item in an array is called an *element*, and each element is accessed by its numerical *index*. Numbering begins with 0.

Operators

Operators are special symbols that perform specific operations on *operands* and return a result. The operators supported by the Java programming language are:
Assignment Operator (=)
Arithmetic Operators (+, -, *, /, %)
Increment/Decrement Operators (++, --)
Equality and Relational Operators (==, <, >, <=, >=, !=)
Conditional Operators (&&, ||, !)

Control flow statements

Control flow statements break up the sequential flow of execution in a program by employing decision making and looping: so the program can conditionally execute specific blocks of code.
Control flow statements are the decision making statements (*if-then, if-then-else, switch*) and the looping statements (*for, while, do-while*).
Java has also the branching statements (*break* and *continue*) that allow a program to conditionally branch to a different section of code.

Focus notes

Glossary

array
A collection of data elements, all of the same type, in which an element's position is uniquely designated by an integer.

boolean
An expression or variable that can only have a true or false value.

casting
Explicit conversion from one data type to another.

comment
Explanatory text that is ignored by the compiler. In Java programming language comments are delimited using // or /*...*/.

exception
An event, generally an error, during the execution of a program that prevents the program from continuing normally.

scope
Where an identifier can be used.

try ... catch
A block of statements to be executed when a Java exception, or run time error, occurs.

unicode
A 16-bit character set defined by ISO.

variable
An item of data defined by a name, a type and a scope.

Acronyms

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AWT	Abstract Window Toolkit
IDE	Integrated Development Environment
I/O	Input/Output
JDK	Java Development Kit
JVM	Java Virtual Machine

SCHEDA DI AUTOVALUTAZIONE

CONOSCENZE

- ☐ Compilazione e interpretazione di un programma Java
- ☐ Operazioni su standard input e standard output
- ☐ Identificatori, variabili e costanti
- ☐ Operatori aritmetici, di confronto e booleani
- ☐ Operatori di incremento prefissi e postfissi
- ☐ Strutture di sequenza, selezione e ripetizione
- ☐ Cicli interrotti e cicli infiniti
- ☐ Array a una e due dimensioni
- ☐ Eccezioni

ABILITÀ

- ☐ Scrivere un semplice programma in Java
- ☐ Rappresentare le operazioni di input/output standard
- ☐ Dichiarare le variabili e le costanti
- ☐ Inserire frasi di commento nel programma
- ☐ Utilizzare le strutture di controllo
- ☐ Individuare i cicli infiniti
- ☐ Dichiarare le strutture di dati array
- ☐ Gestire le eccezioni

SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 539

Ambienti di sviluppo in Java

Nel seguito vengono presentati due esempi di ambienti di sviluppo software per la programmazione nel linguaggio Java.

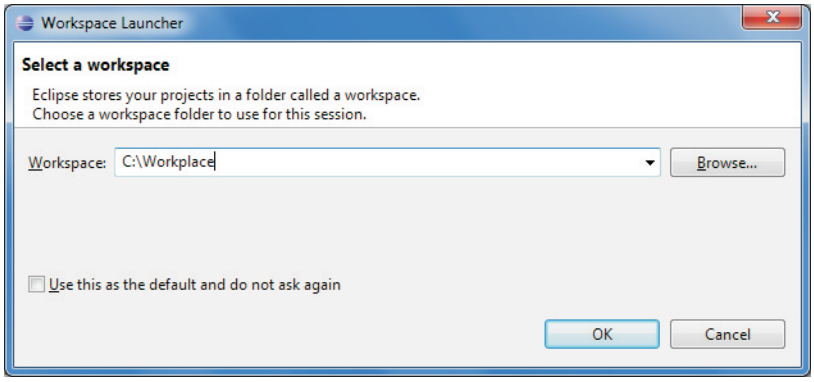
Eclipse

Eclipse è un ambiente di sviluppo integrato (**IDE**, *Integrated Development Environment*) per la programmazione in linguaggio Java. È stato realizzato e viene continuamente aggiornato da un'organizzazione non-profit, *Eclipse Foundation* (*www.eclipse.org*), di cui fanno parte molte importanti società nel mondo dell'informatica. Questo IDE è distribuito con una licenza *open source* che ne permette il libero utilizzo.

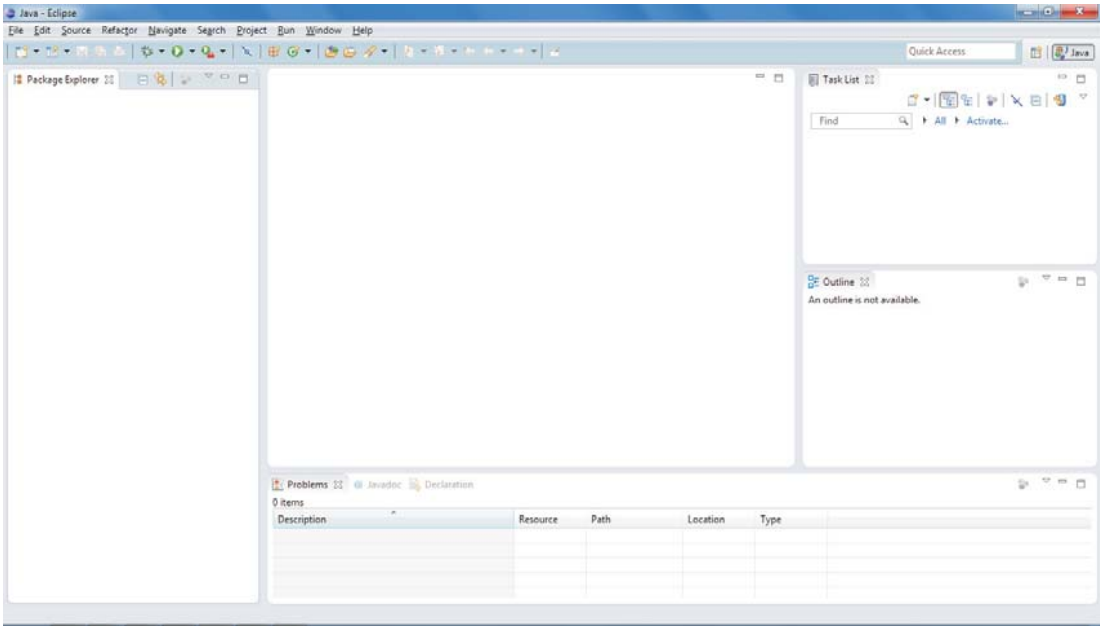
Eclipse è una piattaforma su cui si possono inserire diversi **plug-in**, cioè dei componenti software realizzati per scopi specifici e che permettono di estendere le funzionalità di base dell'ambiente di sviluppo. Per esempio, ci sono plug-in per la costruzione guidata delle interfacce grafiche, per la programmazione in diversi linguaggi (PHP, Python, C), per la realizzazione di diagrammi di documentazione, per la diagnostica degli errori (*bug*) e per la gestione delle versioni del codice (*versioning*).

Nella pagina di **download** di Eclipse è possibile scegliere tra vari pacchetti di installazione che si differenziano per il numero di *plug-in* che sono già inclusi. È opportuno utilizzare i pacchetti **Eclipse Classic** oppure **Eclipse IDE for Java Developers**. Si noti che l'installazione di Eclipse deve essere eseguita successivamente all'installazione del **JDK** (*Java Development Kit*).

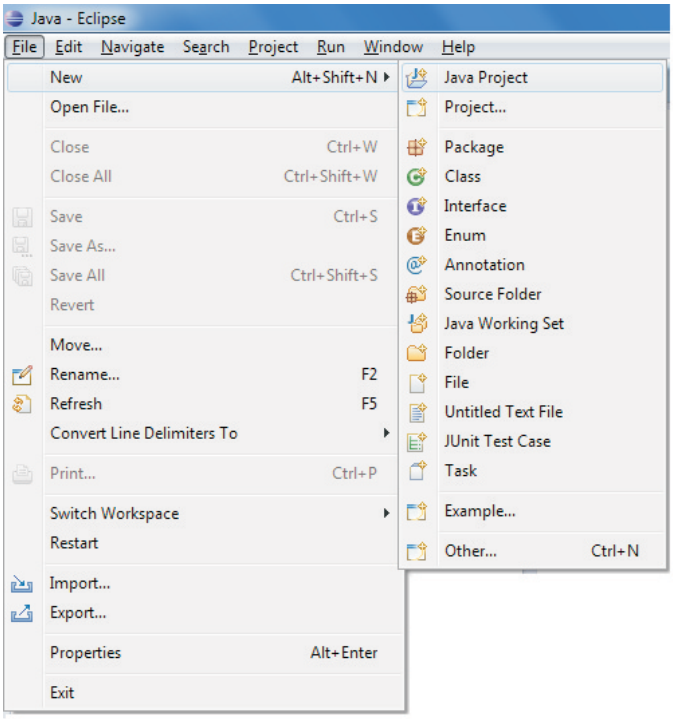
All'avvio del programma viene richiesto al programmatore di indicare una cartella (**Workplace**) per la sessione di lavoro appena iniziata. Il *Workplace* corrisponde alla cartella di lavoro in cui Eclipse memorizza i progetti. Attivando l'apposita casella di spunta, si può impostare il *Workplace* predefinito in modo che le successive aperture di Eclipse facciano riferimento a questa cartella di lavoro predefinita.



La finestra dell'IDE è composta, in alto, dal *barra dei menu* e dalla *barra degli strumenti* mentre la parte centrale, chiamata anche **Workbench** (letteralmente, tavolo di lavoro), si presenta con un insieme di riquadri configurabili. Solitamente il riquadro a sinistra visualizza la lista dei progetti e consente la navigazione nelle cartelle del disco, il riquadro centrale è l'*editor*, il riquadro inferiore è usato per l'output e i messaggi di errore, mentre a destra altri riquadri contengono i dettagli della struttura delle classi, degli attributi e dei metodi.

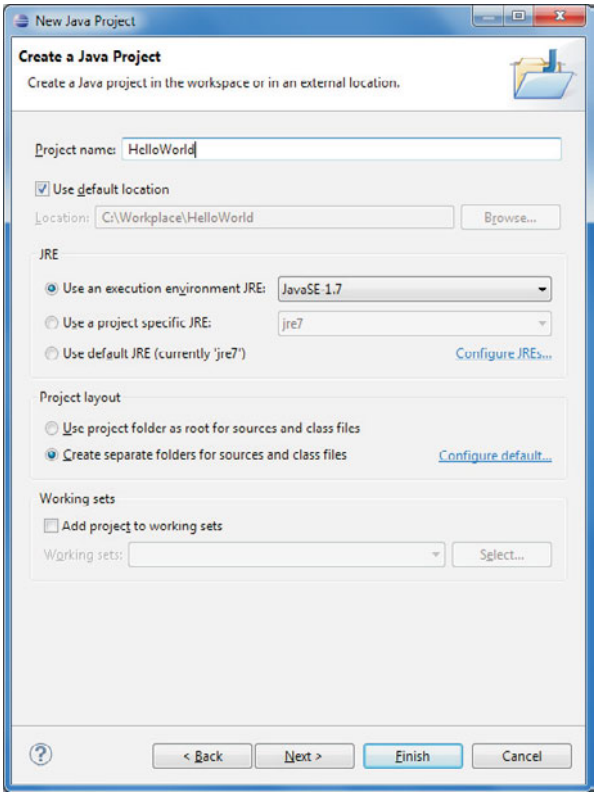


Per creare un nuovo progetto si deve fare clic sul menu **File** e poi, all'interno della voce **New**, fare clic su **Java Project**. In alternativa si può fare clic con il tasto destro nel riquadro di sinistra e poi scegliere dal menu contestuale la voce *New* e poi *Java Project*.



Viene richiesto il nome del progetto. Volendo realizzare un semplice esempio che visualizza il messaggio "Hello, World", scriviamo *HelloWorld* nella casella **Project name**. Si noti che il progetto viene inserito nel *Workplace* indicato all'inizio della sessione di lavoro, all'interno di una cartella con lo stesso nome assegnato al progetto.

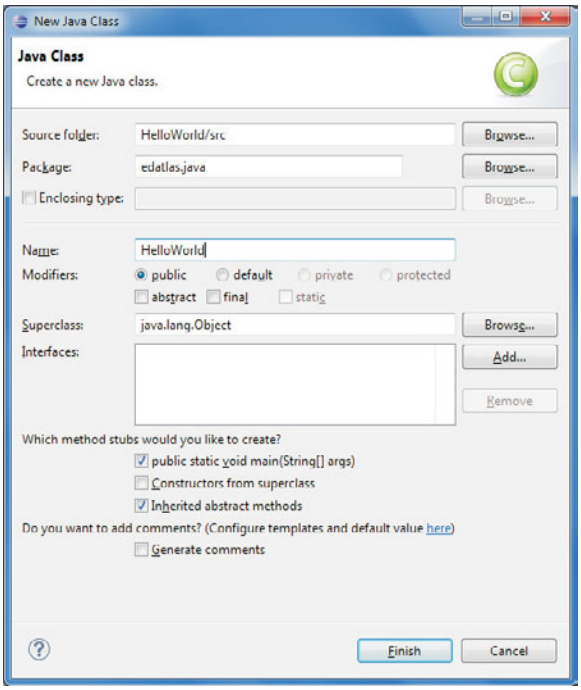
Facciamo infine clic su **Finish**.



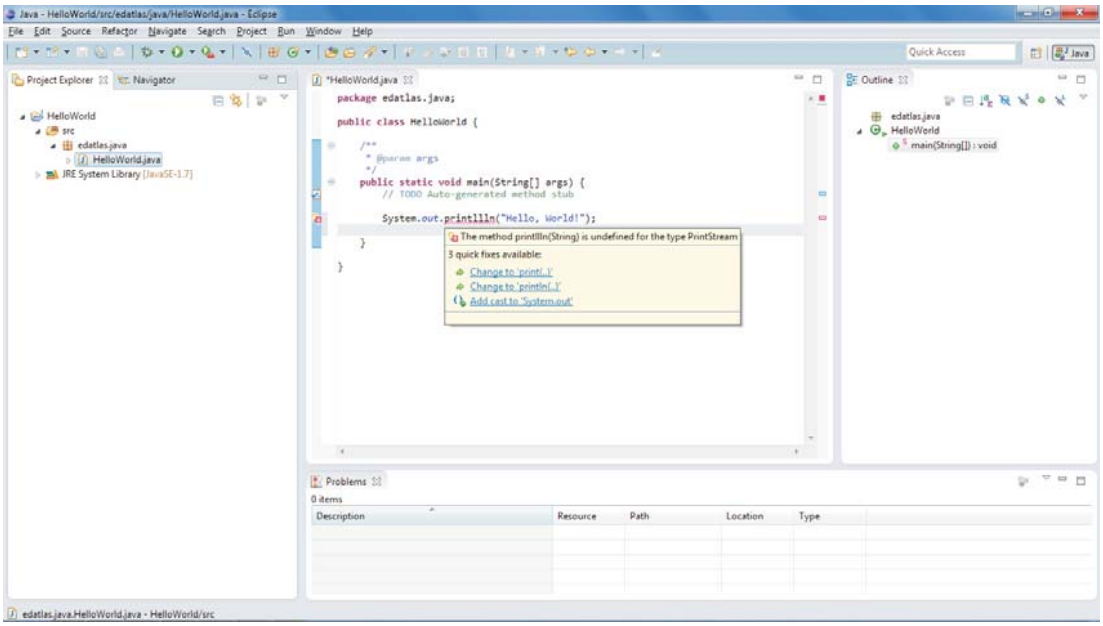
Viene creato il progetto e vengono create le cartelle *src* (raggruppa i file sorgente **.java**) e *bin* (raggruppa i file compilati **.class**).

Per creare una nuova classe si deve fare clic sul menu **File** e poi, all'interno della voce **New**, fare clic su **Class**. In alternativa si può fare clic con il tasto destro nel riquadro di sinistra e poi scegliere dal menu contestuale la voce *New* e poi *Class*.

Scriviamo il nome della classe *HelloWorld* nella casella **Name**, successivamente selezioniamo la casella di spunta per attivare la creazione automatica del metodo *main*. Si noti che l'indicazione del nome di un *package*, seppur non obbligatoria, è consigliata per migliorare l'organizzazione del codice sorgente.
Facciamo infine clic su **Finish**.



Viene creato il file *HelloWorld.java* e viene inserito nella cartella *src* del progetto *HelloWorld*. Nel riquadro centrale viene visualizzato il codice sorgente mentre nel riquadro a sinistra (*Project Explorer*) viene visualizzata la struttura del progetto. Per aprire e modificare un file presente nel progetto è sufficiente fare doppio clic sul relativo nome nel riquadro *Project Explorer*. Il file viene visualizzato nell'editor di testi.



Si noti che il testo del codice viene colorato in modo automatico: in fucsia le parole chiave, in nero i nomi delle classi e dei metodi, in blu le stringhe e in verde i commenti. Inoltre eventuali errori sono segnalati con la linea rossa ondulata: passando con il mouse sopra la parola sottolineata, viene visualizzato il messaggio di errore con un suggerimento per la correzione.

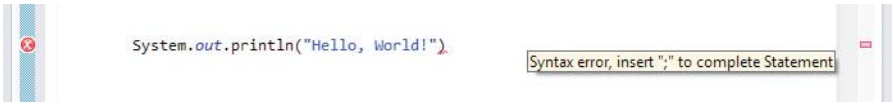
Salviamo il lavoro facendo clic sul menu **File** e poi, su **Save** oppure con un clic sulla relativa icona nella barra degli strumenti (la scorciatoia da tastiera è la combinazione di tasti **Ctrl + S**).

Il programma può essere compilato ed eseguito facendo clic sul menu **Run** e poi sulla voce **Run**, oppure usando la combinazione di tasti **Ctrl + F11**. I messaggi generati dall'esecuzione vengono visualizzati nel riquadro **Console** nella parte inferiore del video.

Eclipse è impostato per compilare il programma ogni volta che viene eseguito il salvataggio. I messaggi generati del compilatore si possono distinguere in due categorie: gli **avvertimenti** (*warning*) e gli **errori** (*error*).
Gli *avvertimenti* sono evidenziati con etichette gialle poste ai lati della finestra di editing. La loro presenza non pregiudica la compilazione e nemmeno l'esecuzione, ma segnala i punti in cui il programma può essere migliorato e reso più efficiente.

Per esempio, una variabile non utilizzata produce il seguente avvertimento:

Gli *errori sintattici* invece non permettono la compilazione e l'esecuzione del programma. Vengono evidenziati con etichette rosse poste ai lati della finestra di editing. Per esempio, la mancanza del punto e virgola al termine di una riga produce il seguente errore:



Gli errori di programmazione che non vengono evidenziati dal compilatore possono essere analizzati utilizzando la modalità di esecuzione in *debug*.
Le attività più comuni che si possono attivare per il **debugging** dell'applicazione sono:
- impostare punti di interruzione (**breakpoints**) che fermano temporaneamente l'esecuzione in corrispondenza di linee di codice prefissate;
- osservare il valore assunto dalle variabili;
- eseguire il programma passo passo;
- impostare la prossima istruzione da eseguire.

Per fissare punti di interruzione è possibile fare doppio clic a lato della linea di codice. Un punto di interruzione attivo viene visualizzato con un cerchio blu.

Facendo doppio clic sul cerchio, il punto di interruzione viene eliminato. La scorciatoia da tastiera per attivare e disattivare i punti di interruzione è la combinazione di tasti **Ctrl + Maiuscolo + B**.

Dopo aver indicato almeno un punto di interruzione, per eseguire il *debugging* dell'applicazione si deve fare clic sul menu **Run** e poi sulla voce **Debug**, oppure si deve premere il tasto **F11**.

Per vedere il valore delle variabili e delle espressioni si deve passare con il mouse sopra la variabile, oppure aprire il riquadro **Expressions**, dal menu **Debug** e poi **Watch**, e scrivere il nome della variabile.

Per eseguire il programma passo passo:

- fare clic su **Step Into** (oppure premere il tasto **F5**)

- fare clic su **Step Over** (oppure premere il tasto **F6**).

Entrambi i comandi eseguono una sola riga di codice e poi sospendono l'esecuzione. Nel caso di *Step Into*, se la riga in esecuzione è un metodo, l'attività di debug entra nel metodo eseguendo le sue istruzioni passo passo.

Per continuare l'esecuzione fino al prossimo punto di interruzione, fare clic su **Resume** (oppure premere il tasto **F8**).

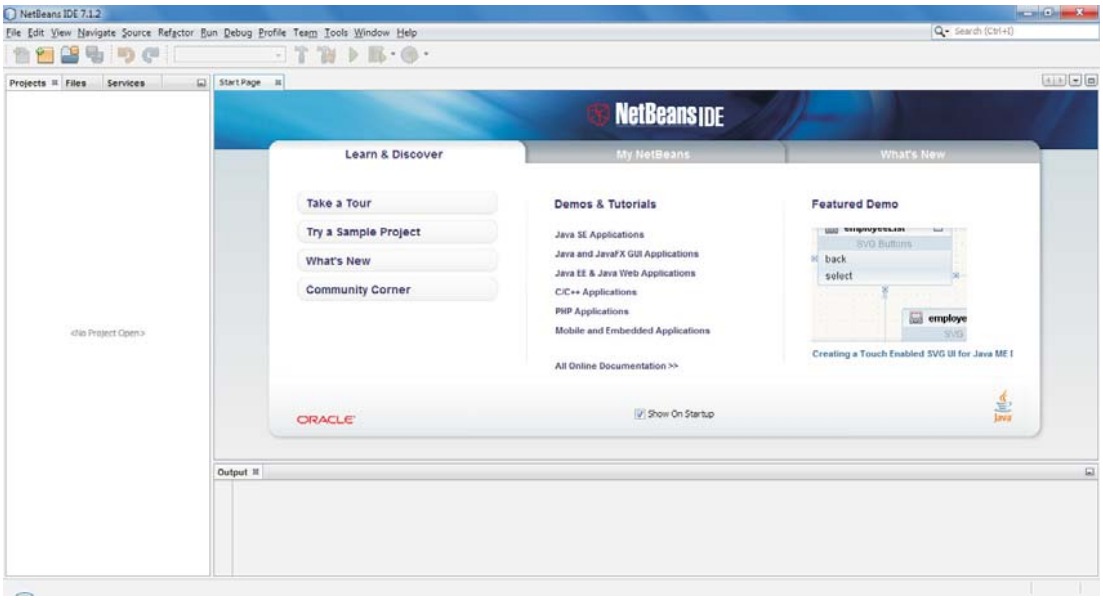
Infine, per interrompere la modalità di debug, fare clic su **Terminate** (oppure premere la combinazione di tasti **Ctrl + F2**).

La sessione di lavoro si conclude, dopo aver salvato i file, nel momento in cui viene chiusa la finestra di Eclipse. L'ultima disposizione dei riquadri sul piano di lavoro viene memorizzata e viene presentata nuovamente al programmatore nella successiva sessione di lavoro.

NetBeans

NetBeans è un ambiente di sviluppo integrato (**IDE**, *Integrated Development Environment*) per la programmazione in linguaggio Java. L'IDE fornisce diversi strumenti per facilitare l'attività del programmatore: suggerisce la struttura di base delle classi, visualizza gli errori sintattici direttamente nella finestra dell'editor e, tramite l'autocompletamento, velocizza la scrittura del codice.

NetBeans è un software *open source* e può essere installato sui sistemi operativi *Windows*, *Linux* e *Mac OS*.



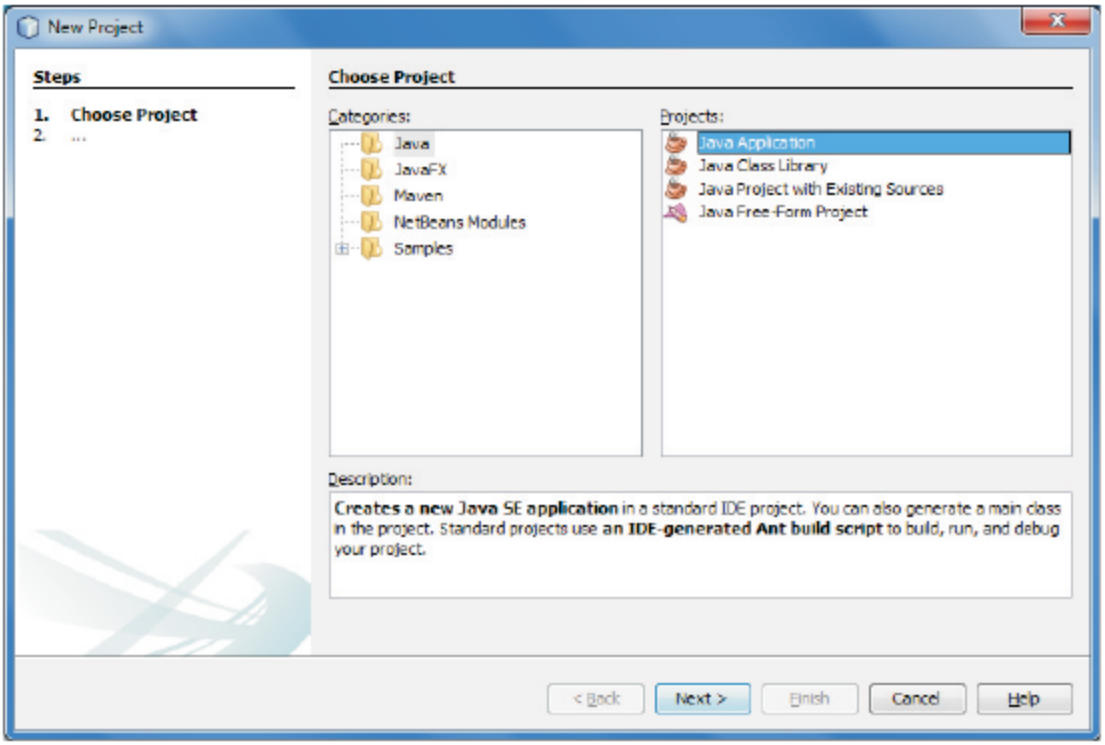
All'avvio del programma viene aperta la precedente videata, con:
- in alto, la barra dei menu e la barra degli strumenti;
- a sinistra, il riquadro per la navigazione tra i progetti e i file;
- al centro, la pagina iniziale con i collegamenti alla documentazione, ai progetti recentemente aperti e alle novità;
- in basso, il riquadro per l'output e i messaggi di errore.

L'attività di programmazione in NetBeans inizia con la creazione di un **progetto**. Ogni progetto può essere visto come un programma Java, composto da un insieme di file sorgenti e dalle impostazioni necessarie per la sua compilazione ed esecuzione.

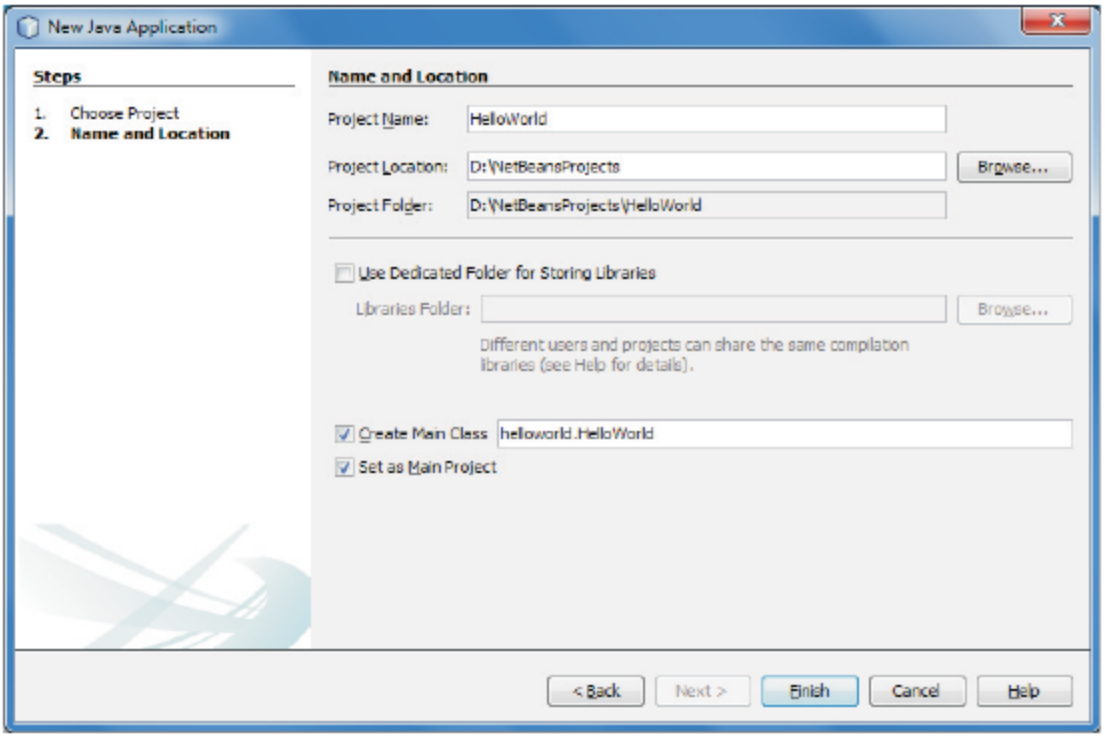
Per creare un nuovo progetto si deve fare clic sul menu **File** e poi su **New Project** (la scorciatoia da tastiera è la combinazione di tasti **Ctrl + Maiuscolo + N**). In alternativa si può fare clic sull'icona nella barra degli strumenti.

Viene aperta una finestra in cui è possibile scegliere tra varie tipologie di progetto. Scegliamo, all'interno della categoria *Java*, il tipo di progetto **Java Application** e poi facciamo clic su **Next**.

Si noti che, nella categoria **Sample**, è possibile scegliere tra un insieme di applicazioni di esempio già preconfezionate. Creare un progetto a partire da questi esempi è un modo efficace per apprendere la programmazione in linguaggio Java.

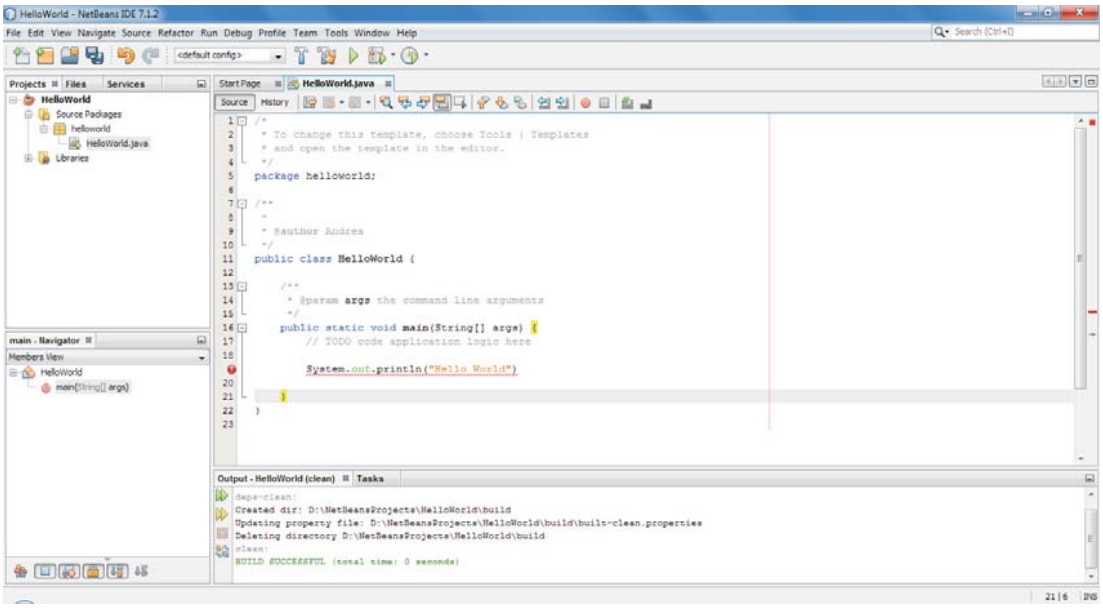


Nella successiva pagina, scriviamo il nome del progetto nella casella **Project Name** e selezioniamo il percorso dove salvare il progetto facendo clic sul pulsante **Browse**. Si noti che il nome del progetto viene in automatico assegnato anche alla classe principale. Per completare la creazione del progetto facciamo clic sul pulsante **Finish**.



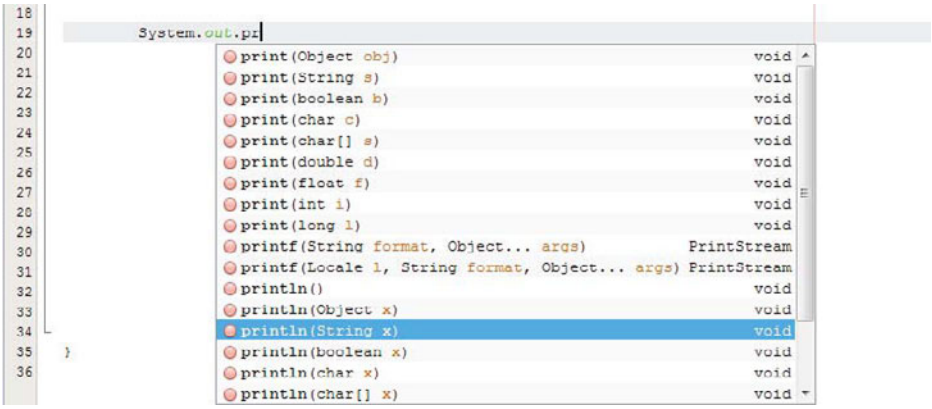
Il nuovo progetto viene aperto nell'IDE e vengono visualizzati i seguenti riquadri:

- **Projects**: contiene l'elenco dei progetti e dei file che li compongono, facendo doppio clic su un file lo si apre nell'editor;
- **Navigator**: visualizza gli attributi e i metodi della classe, facendo doppio clic su un elemento permette di visualizzarlo velocemente nell'editor senza dover usare le barre di scorrimento per muoversi lungo il codice sorgente;
- **Editor**: mostra il file sorgente *.java* colorato in modo automatico, con le parole chiave in blu, i nomi degli elementi della classe in nero e i commenti in grigio;
- **Output**: mostra i messaggi, tra cui quelli di conferma della creazione del progetto.



Si noti che gli eventuali errori sintattici sono segnalati con una linea ondulata rossa e un simbolo rosso nella parte sinistra dell'editor.
Si noti inoltre che, posizionando il cursore su una parentesi graffa, viene evidenziato in giallo il blocco a cui la parentesi fa riferimento. In questo modo il programmatore può verificare la corretta disposizione delle parentesi e accorgersi, in modo agevole, di eventuali errori.

La scrittura del codice sorgente è facilitata dalla funzionalità di **autocompletamento** che mostra, in tempo reale, l'elenco degli elementi (oggetti, attributi e metodi) che più si avvicinano a quanto si sta digitando. Usando le frecce direzionali si può scegliere l'elemento d'interesse e lo si può inserire nel codice premendo il tasto *Invio*.

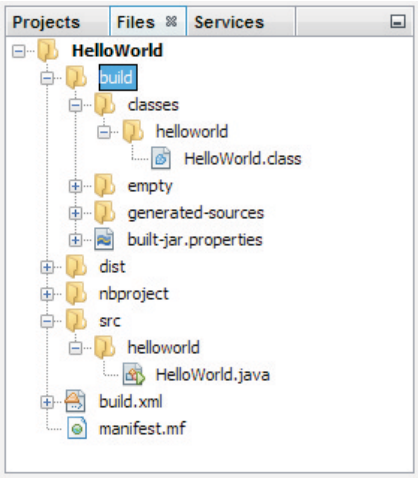


Dopo aver modificato il codice sorgente, salviamo il lavoro dal menu **File** con la scelta **Save All** oppure con un clic sull'icona della barra degli strumenti
(la scorciatoia da tastiera è la combinazione di tasti **Ctrl + Maiuscolo + S**).

Per compilare il programma facciamo clic sull'icona **Build Main Project**
(la scorciatoia da tastiera è il tasto **F11**).
L'IDE esegue il compilatore del JDK (*javac*) e visualizza, nella finestra di *Output* in basso, lo stato della compilazione ed eventuali errori. Se la compilazione termina senza errori viene mostrata la scritta *BUILD SUCCESSFUL* in verde.

```
Output - HelloWorld (jar)   Tasks
Updating property file: D:\NetBeansProjects\HelloWorld\build\built-jar.properties
compile:
Copying 1 file to D:\NetBeansProjects\HelloWorld\build
Nothing to copy.
To run this application from the command line without Ant, try:
java -jar "D:\NetBeansProjects\HelloWorld\dist\HelloWorld.jar"
jar:
BUILD SUCCESSFUL (total time: 0 seconds)
```

Usando il riquadro **Files** si può verificare che i file compilati *.class* sono stati posizionati nelle sottocartelle della cartella *build*.



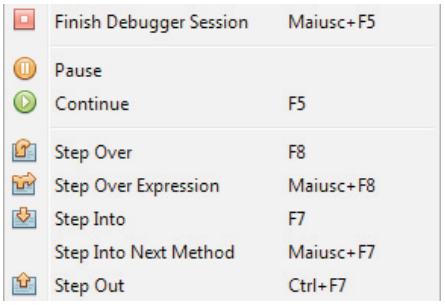
Per eseguire il programma facciamo clic sull'icona **Run Main Project**
(la scorciatoia da tastiera è il tasto **F6**).

Il risultato dell'esecuzione e i messaggi di testo generati con il metodo *System.out.println* vengono mostrati nella finestra *Output*.

Per attivare il **debug** del programma facciamo clic sull'icona **Debug Main Project**
(la scorciatoia da tastiera è la combinazione di tasti **Ctrl + F5**).

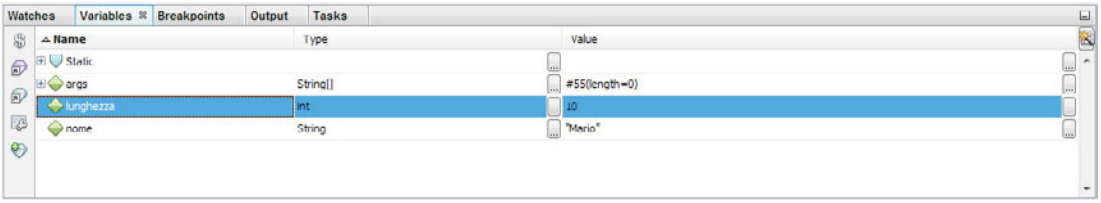
Prima di attivare il debug è necessario indicare almeno un punto di interruzione (**Breakpoint**) all'interno del codice sorgente. Il punto di interruzione deve essere posizionato sulla prima riga di codice da cui si intende iniziare l'analisi e la ricerca degli errori.
Per inserire un punto di interruzione si deve fare clic a lato della linea di codice. Un punto di interruzione attivo viene visualizzato con un quadrato rosa posto sulla sinistra e con l'intera linea evidenziata in rosa. Facendo clic sul quadrato, il punto di interruzione viene eliminato.

Attivando il debug, l'esecuzione del programma parte e si sospende nel punto in cui è stato inserito il *Breakpoint*. Per proseguire con un'esecuzione passo passo o per interrompere il programma si devono usare i comandi presenti nel menu **Debug**.



I comandi mostrati in figura, con le stesse icone, possono essere attivati dalla barra degli strumenti.

Ad ogni passo di esecuzione, è possibile controllare il valore delle variabili passando il mouse sopra il nome della variabile nella finestra dell'editor. Inoltre, in modalità *debugging*, viene aperto il riquadro **Variables**, nella parte inferiore dell'IDE, che facilita la consultazione dell'elenco completo delle variabili e permette di monitorare il valore delle espressioni.



Quando l'applicazione da realizzare è composta da più classi, per aggiungere una nuova classe al progetto si deve fare clic sul menu **File** e poi su **New File** (la scorciatoia da tastiera è la combinazione di tasti **Ctrl + N**). In alternativa si può fare clic sull'icona corrispondente nella barra degli strumenti.

Viene aperta una finestra in cui si deve: scegliere il tipo di file **Java Class**, fare clic su *Next*, poi inserire il nome della classe nella casella *Class Name* e infine fare clic su *Finish*.

Per chiudere il progetto aperto nell'ambiente di sviluppo, dopo aver salvato il lavoro, facciamo clic su **File** e poi su **Close Project**.

parte terza

Programmazione ad oggetti

4 Classi e oggetti

OBIETTIVI DI APPRENDIMENTO

In questo capitolo conoscerai i concetti di base della programmazione ad oggetti e potrai individuare gli aspetti della metodologia orientata agli oggetti. Imparerai a definire le classi con attributi e metodi. Sarai anche in grado di descrivere le classi attraverso diagrammi. Potrai applicare i principi della programmazione ad oggetti utilizzando il linguaggio Java.

1 Orientamento agli oggetti

La **programmazione** è un'attività che si occupa di risolvere i problemi per mezzo di programmi eseguibili dai computer. Questa attività può essere suddivisa in quattro fasi:

- definizione del problema, dei dati di input e di output;
- organizzazione dell'algoritmo risolutivo;
- stesura del programma, cioè la traduzione dell'algoritmo nel linguaggio di programmazione;
- prove di esecuzione del programma.

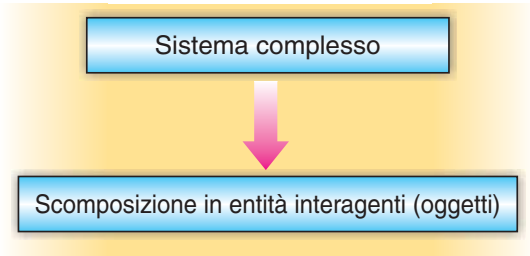
La costruzione di programmi ordinati, basati sull'uso delle strutture di controllo e sull'organizzazione modulare del codice, si chiama **programmazione strutturata**.

Questa metodologia pone l'attenzione principalmente sulla seconda fase dell'attività di programmazione, cioè sull'organizzazione dell'algoritmo risolutivo. L'interesse è rivolto a quello che il programma deve fare e il problema viene sviluppato individuando le procedure e le sottoprocedure, cioè i passi dell'elaborazione che portano alla soluzione. Il programma viene costruito come un insieme di funzioni che vengono richiamate nell'ordine corretto a partire dalla funzione principale chiamata **main**.

La programmazione strutturata ha permesso di costruire programmi ordinati e basati sull'organizzazione e la suddivisione dei programmi in **moduli** funzionalmente indipendenti.
Con il passare del tempo è aumentata la richiesta di sviluppare programmi sempre più complessi e in grado di gestire grandi quantità di dati. Queste necessità hanno messo in mostra alcuni limiti della tecnica della programmazione strutturata, tra cui la difficoltà nel riutilizzare il codice già scritto e la mancanza di strumenti per analizzare e descrivere i sistemi complessi, composti da varie entità interagenti tra loro.

Programmazione strutturata

Programmazione ad oggetti



A partire dal 1980 si è quindi sviluppato un nuovo stile di scrittura del codice che ne consente un reimpiego sempre maggiore. L'idea, di per sé molto semplice, rivoluziona il modo di implementare il codice inserendo un nuovo approccio, rappresentato dai dati del problema. Questo modo di programmare si chiama **programmazione orientata agli oggetti**.
La programmazione orientata agli oggetti non presuppone l'eliminazione delle tecniche precedenti, ma piuttosto le completa, aggiungendo loro una nuova dimensione.

La **programmazione orientata agli oggetti**, in breve **OOP** (*Object-Oriented Programming*), prende il nome dall'elemento su cui si basa, l'oggetto.

Questa metodologia è un modo di pensare al problema in termini di **sistema** e può essere utilizzata efficacemente nelle fasi di analisi del problema e di progettazione, prima che nella fase di programmazione.

Durante la fase di analisi si crea un **modello** del sistema, individuando gli elementi di cui è formato e i comportamenti che deve avere. In questa fase non interessano le modalità con le quali i comportamenti vengono effettivamente implementati, ma soltanto gli oggetti che compongono il sistema e le interazioni tra essi. Successivamente, durante la stesura del programma, il *linguaggio di programmazione orientato agli oggetti* faciliterà la traduzione degli oggetti e delle interazioni tra essi in un programma eseguibile.

Al contrario della programmazione strutturata, l'orientamento ad oggetti si focalizza sulla prima fase dell'attività di programmazione, cioè sull'analisi dei dati, che in questo contesto assumono il nome di **oggetti**. Al centro dell'attenzione c'è il sistema che si vuole analizzare: l'analisi del sistema si occupa di individuare le entità che fanno parte del sistema stesso e le interazioni tra queste entità.

Il programma, realizzato con un orientamento ad oggetti, si sviluppa attraverso le interazioni tra gli oggetti: durante l'esecuzione del programma, gli oggetti possono cambiare il loro stato e possono richiedere l'esecuzione di operazioni associate ad altri oggetti.

Lo stile di programmazione orientato agli oggetti procura diversi vantaggi:
- facilità di **lettura** e di **comprensione** del codice, anche per persone diverse dall'autore;
- rapidità nella **manutenzione** del programma nel tempo per correzioni o miglioramenti;
- **robustezza** del programma in situazioni che modellano sistemi complessi e con grandi quantità di dati;
- **riusabilità** del codice all'interno di altri programmi.

Questa metodologia, grazie ai suoi vantaggi, si è sviluppata, oltre che nei linguaggi di programmazione, anche in altri settori dell'informatica, come i database, per cui, in generale, la sigla OO (*Object-Oriented*) si riferisce alla presenza di un orientamento agli oggetti.

2 Gli oggetti e le classi

Gli **oggetti** rappresentano le entità del problema o della realtà che si vuole automatizzare con l'informatica. Un oggetto è in grado di memorizzare le informazioni che riguardano il suo stato; è anche possibile associare a un oggetto un insieme di operazioni che esso può compiere.

In particolare, un oggetto può essere definito elencando sia le sue caratteristiche (*attributi*), sia il modo con cui interagisce con l'ambiente esterno, cioè i suoi comportamenti (*metodi*).

Gli **attributi** rappresentano gli elementi che caratterizzano l'oggetto, utili per descrivere le sue proprietà e definirne il suo stato.
I **metodi** rappresentano le funzionalità che l'oggetto mette a disposizione.

Prendiamo come esempio un'automobile e analizziamo questo oggetto in termini di caratteristiche e comportamenti.
Alcune sue *caratteristiche* sono: la velocità, il colore, il numero di porte, il livello del carburante e la posizione della marcia.
Come si vede, queste caratteristiche possono descrivere le proprietà fisiche dell'oggetto, come il colore e il numero di porte. Possono anche indicare lo stato dell'oggetto in un determinato momento, come la velocità che può variare se si accelera.

Tra i *comportamenti* dell'oggetto automobile ci sono i seguenti: accelera, frena, gira (a destra o a sinistra), cambia marcia e rifornisciti.
Le funzionalità espresse dai comportamenti possono concretizzarsi con le azioni oppure con il cambiamento dello stato dell'oggetto. I comportamenti *accelera* e *fermati* agiscono modificando la velocità dell'automobile, nel primo caso aumentandola e nel secondo diminuendola. Il comportamento *gira* esegue l'azione di far girare l'automobile a destra o a sinistra. Infine i comportamenti *cambia marcia* e *rifornisciti* influenzano rispettivamente la posizione della marcia e il livello del carburante.
Si noti che tutti i comportamenti sono indicati tramite dei verbi, per evidenziare il loro aspetto operativo.

La **struttura** di un oggetto è completamente descritta quando vengono elencate le caratteristiche e i comportamenti dell'oggetto. Nei linguaggi di programmazione orientati agli oggetti, la struttura di un oggetto viene descritta con le classi.

La **classe** è la descrizione astratta degli oggetti attraverso gli attributi e i metodi.

Una classe viene rappresentata con uno schema grafico detto **diagramma delle classi**, che ne evidenzia il nome, gli attributi e i metodi. Il diagramma è costituito da un rettangolo diviso in tre zone tramite linee: in alto si indica il nome della classe che viene definita, nella zona centrale l'elenco degli attributi e in basso l'elenco dei metodi.
La struttura generale è la seguente:

<nome classe>
<attributo1> <attributo2> …
<metodo1> <metodo2> …

Con riferimento all'esempio dell'automobile, la classe *Automobile* è rappresentata con il seguente diagramma della classe:

Automobile
velocità colore numero di porte livello del carburante posizione della marcia
accelera frena gira cambia marcia rifornisciti

La classe può essere immaginata come uno stampo dal quale vengono creati più esemplari (*oggetti*), tutti con gli stessi attributi e gli stessi metodi.

Nei programmi orientati agli oggetti, un oggetto può esistere solo se esiste la relativa classe che ne descrive le caratteristiche e le funzionalità.

Per utilizzare un oggetto occorre crearlo come esemplare della classe, cioè come **istanza** di una classe.

Creando due istanze, *auto1* e *auto2* della classe *Automobile*, si ottengono due oggetti:

auto1
velocità = 60 colore = rosso numero di porte = 5 livello del carburante = ½ posizione della marcia = 3

auto2
velocità = 48 colore = bianco numero di porte = 5 livello del carburante = ¾ posizione della marcia = 2

Per descrivere gli oggetti abbiamo usato la notazione grafica, chiamata **diagramma degli oggetti**, che utilizza un rettangolo contenente nella parte alta il nome dell'oggetto e nella parte centrale l'elenco delle caratteristiche dell'oggetto con i rispettivi valori.
Si noti che i metodi non vengono riportati nel diagramma in quanto sono comuni a tutti gli oggetti della stessa classe.

I simboli utilizzati negli schemi grafici precedenti rispettano gli standard del linguaggio **UML** (*Unified Modeling Language*), un insieme di regole riconosciute a livello internazionale per specificare, rappresentare, progettare e realizzare software, in particolare, nel contesto della programmazione, per descrivere le classi e gli oggetti.

Nella terminologia orientata agli oggetti si dice che l'oggetto *auto1* è un'**istanza** della classe *Automobile*. Più brevemente, l'oggetto *auto1* è di classe *Automobile*. Allo stesso modo l'oggetto *auto2* è un'istanza della classe *Automobile*. Quindi la stessa classe può generare più istanze: ogni istanza si differenzia per il valore assunto dai suoi attributi, ma tutte possono utilizzare i metodi della classe.

Ogni classe possiede un particolare metodo predefinito, detto **costruttore**, che viene attivato quando si crea un oggetto. Dopo aver creato un oggetto è possibile modificare le sue proprietà o invocare l'attivazione di un suo metodo.
Tramite l'uso delle classi, tutto ciò che si riferisce a un certo oggetto è racchiuso e contenuto all'interno della classe stessa. Questo raggruppamento conferisce alla classe il significato di un'unità di programmazione, riutilizzabile in altri programmi. Inoltre, la gestione di sistemi complessi e la modifica nel tempo dei programmi risulta più semplice perché tutte le informazioni utili che riguardano gli oggetti del sistema sono ben localizzate. Questi aspetti descrivono il concetto di **incapsulamento**, che è uno dei concetti alla base della programmazione ad oggetti.

Il termine **incapsulamento** indica la proprietà degli oggetti di incorporare al loro interno sia gli attributi che i metodi, cioè le caratteristiche e i comportamenti dell'oggetto.

Si crea come una capsula, una barriera concettuale, che isola l'oggetto dalle cose esterne. Si dice che gli attributi e i metodi sono incapsulati nell'oggetto. In questo modo tutte le informazioni utili che riguardano un oggetto sono ben localizzate. Questa imposizione di raccogliere tutto quello che riguarda una singola entità all'interno di un oggetto è uno dei vantaggi che viene offerto dalla programmazione orientata agli oggetti.

L'ASTRAZIONE NELLA OOP

Gli oggetti della OOP sono solitamente un'astrazione di un oggetto reale. In particolare, per modellare un oggetto dobbiamo compiere un'astrazione sulle caratteristiche e sui comportamenti, individuando quelli che noi riteniamo utili per quel particolare programma. Lo stesso oggetto può essere definito in molti modi diversi a seconda degli attributi e dei metodi che si vogliono utilizzare. Queste scelte sono strettamente legate al tipo di problema che si sta analizzando.

La struttura dell'oggetto automobile, descritta precedentemente, evidenzia un sottoinsieme delle caratteristiche e dei comportamenti delle generiche automobili. Se il problema avesse chiesto di tenere traccia del fatto che l'automobile è accesa o spenta e di comandare l'accensione e lo spegnimento, la classe *Automobile* avrebbe assunto una struttura diversa. Si sarebbe dovuto aggiungere l'attributo
- accesa,

indicante lo stato di accensione (*si/no* oppure *on/off*);
e i metodi:
- avviati,
- spegniti,

il primo per accendere e il secondo per spegnere l'automobile.

In generale gli oggetti rappresentano qualunque cosa possa essere modellata in un programma. Abbiamo visto un esempio di un oggetto reale, l'automobile, ma si possono considerare come oggetti anche i concetti astratti. Alcuni esempi possono essere la finestra di un'interfaccia grafica oppure una struttura dati come la lista.

La **finestra di un'interfaccia grafica**, per esempio una finestra del sistema operativo Windows, può essere descritta con un orientamento ad oggetti con il seguente schema.

Finestra
posizione sullo schermo visibile titolo colore di sfondo
apri muovi (nuova posizione) ingrandisci riduci chiudi

L'attributo *posizione sullo schermo* indica le coordinate x,y dell'angolo in alto a sinistra della finestra.
L'attributo *visibile* indica se la finestra è nascosta da altre finestre oppure è visibile all'utente.
I metodi indicano comportamenti operativi e, in questo esempio, permettono di modificare le caratteristiche della finestra. Per esempio, usando il metodo *apri* si può rendere visibile la finestra; usando il metodo *muovi* e indicando le nuove coordinate, si può cambiare la posizione della finestra.
A partire da questo modello di finestra si possono costruire degli oggetti concreti. Per esempio, l'oggetto *blocco note* e l'oggetto *esplora risorse* sono due istanze della classe *Finestra*.

blocco note
posizione sullo schermo = 100, 50 visibile = no titolo = Senza nome – Blocco note colore di sfondo = bianco

esplora risorse
posizione sullo schermo = 0, 0 visibile = si titolo = Computer – Disco locale (C:) colore di sfondo = bianco

La struttura di dati **lista**, intesa come un insieme modificabile di elementi su cui eseguire delle ricerche, può essere descritta con il seguente diagramma di classe.

Lista
numero di elementi elementi
inserisci cancella cerca

Il primo attributo conta il numero di elementi presenti nella lista, il secondo descrive l'elenco dei dati che formano la lista.
I primi due metodi (*inserisci* e *cancella*) servono per modificare la lista, aggiungendo e togliendo un elemento. Il metodo *cerca* permette di ritrovare un dato presente nella lista.

AUTOVERIFICA
Domande da 1 a 4 pag. 225
Problemi da 1 a 12 pag. 228-229

MATERIALI ON LINE
1. Strumenti per disegnare i diagrammi delle classi (UML)
2. I linguaggi orientati agli oggetti puri e ibridi

3 Dichiarazione e utilizzo di una classe

La struttura base della **dichiarazione di una classe** in Java è la seguente:

```
class NomeClasse
{
  // attributi
  // metodi
}
```

La parola chiave **class** serve per iniziare la dichiarazione di una classe ed è seguita dal nome della classe. Per convenzione, i nomi delle classi si indicano con la lettera iniziale maiuscola. Tra le parentesi graffe si inserisce tutto il contenuto della classe, costituito dagli attributi e dai metodi. Naturalmente possono esistere classi formate da soli attributi oppure da soli metodi.

Le dichiarazioni degli attribuiti e dei metodi della classe verranno illustrate con maggiore dettaglio nei paragrafi successivi.
Si noti che la dichiarazione di una classe era stata introdotta anche negli esempi del capitolo precedente, in particolare il suo scopo era quello di contenere solo il metodo speciale *main*. Le classi di questo tipo non sono utilizzate per creare degli oggetti, ma servono per indicare il punto da cui inizia l'esecuzione del programma.
L'utilizzo della classe all'interno del programma avviene attraverso la creazione delle sue **istanze**, cioè degli *oggetti*. La **dichiarazione di un oggetto** segue la normale sintassi della dichiarazione delle variabili:

```
NomeClasse nomeOggetto;
```

La creazione dell'istanza *nomeOggetto* si concretizza con l'uso della parola chiave **new**, seguito dal nome della classe e dalle parentesi tonde, nel seguente modo:

```
nomeOggetto = new NomeClasse();
```

Per convenzione, i nomi degli oggetti si indicano con la lettera iniziale minuscola.

PROGETTO 1

Descrivere la classe per il programma che calcola l'area di un cerchio conoscendone il raggio.

Il progetto mostra la modalità per la creazione di una classe con cui gestire l'entità *Cerchio*. La classe si compone di un attributo per memorizzare il valore del raggio, di un metodo per impostare il valore del raggio e di un altro metodo per calcolare l'area.

Cerchio
raggio
setRaggio area

Il diagramma mostra che la classe *Cerchio* è composta da un attributo *raggio* e da due metodi, *setRaggio* e *area*.
La codifica della classe *Cerchio* nel linguaggio Java è la seguente.

IMPLEMENTAZIONE DELLA CLASSE (*Cerchio.java*)

```
class Cerchio
{
  // attributo
  private double raggio;

  // metodo1
  public void setRaggio(double r)
  {
    raggio = r;
  }

  // metodo2
  public double area()
```

```
  {
    return (raggio * raggio * Math.PI);
  }
}
```

L'attributo *raggio* è stato dichiarato come numero reale a doppia precisione. Il metodo *setRaggio* imposta il valore del raggio del cerchio, mentre il metodo *area* restituisce un numero reale corrispondente al valore dell'area del cerchio. Nel linguaggio Java **Math.PI** è una costante definita nella classe *Math* che contiene il valore di *pigreco*.
Il codice dell'esempio precedente deve essere memorizzato in un file chiamato *Cerchio.java*. Eseguendo la compilazione della classe, si crea il file *Cerchio.class*. Non si ottiene un programma, ma un nuovo tipo di dato che potrà essere usato all'interno di altre classi. Non si può dire di aver creato un programma completo perché la classe non include il metodo *main*. Quindi non è possibile invocare l'interprete Java. Se si tenta di eseguire il comando:

```
java Cerchio
```

viene segnalato l'errore: "*il metodo principale non è stato trovato nella classe Cerchio*".
Per completare il progetto si deve utilizzare la classe *Cerchio* all'interno di un programma per calcolare l'area dopo che è stato impostato il valore del raggio.
Supponendo di voler calcolare l'area di un tavolo di dimensioni circolari, la codifica del programma che utilizza la classe *Cerchio* è la seguente.

PROGRAMMA JAVA (*ProgCerchio.java*)

```
class ProgCerchio
{
  public static void main(String argv[])
  {
    // dichiarazione dell'oggetto
    Cerchio tavolo;

    // creazione dell'istanza
    tavolo = new Cerchio();

    tavolo.setRaggio(0.75);
    System.out.println("Area del tavolo = " + tavolo.area() );
  }
}
```

Dopo aver dichiarato l'oggetto tavolo e aver creato l'istanza con la parola chiave *new*, si possono richiamare i metodi della classe *Cerchio* con la notazione *nomeOggetto* e *nomeMetodo* separata dal punto: *tavolo.setRaggio()* e *tavolo.area()*.
Il programma deve essere salvato in un file chiamato *ProgCerchio.java*. L'esecuzione del programma viene attivata tramite la compilazione (*javac*) e la successiva interpretazione (*java*), come mostrato dai seguenti comandi:

```
javac ProgCerchio.java
java ProgCerchio
```

Il risultato dell'esecuzione del programma è la scritta:

```
Area del tavolo = 1.7671458676442586
```

4 Dichiarazione degli attributi

Gli **attributi** possono essere immaginati come fossero le variabili viste nel capitolo precedente, con la differenza che vengono dichiarati nel blocco di una classe anziché all'interno del metodo *main*. Gli attributi vengono anche chiamati **variabili di istanza** perché ogni oggetto (istanza di una classe) possiede le proprie variabili nelle quali memorizza il valore dei propri attributi.

La **dichiarazione di un attributo** è simile a quella di una variabile con l'aggiunta dell'indicazione del livello di visibilità.

```
livelloDiVisibilità tipo nomeAttributo;
```

Usando la stessa dichiarazione si possono elencare anche più attributi separandoli con la virgola. È anche possibile assegnare un valore iniziale all'attributo.
Alcuni esempi di dichiarazioni di attributi sono i seguenti:

```
// dichiarazione di un attributo
private boolean trovato;

// dichiarazione di tre attributi dello stesso tipo
private int x, y, z;

// dichiarazione e inizializzazione di un attributo
public double altezza = 15.76;

// dichiarazione di un oggetto come attributo
public Cerchio c;
```

Se a un attributo non è stato associato un valore di inizializzazione, il sistema inizializza automaticamente l'attributo a seconda del tipo. Se è di tipo *boolean* è inizializzato a *false*, se è un carattere o un numero con il valore *zero*, se è un oggetto con *null*.

I **livelli di visibilità** di un attributo stabiliscono se l'attributo è accessibile da altre classi, cioè se dall'esterno della classe che lo contiene si può leggere o modificare il suo valore.
I livelli di visibilità possono essere:
- public
- private
- protected.

Con la visibilità **public** l'attributo è accessibile da qualsiasi altra classe.
Al contrario, l'uso di **private** permette di nascondere l'attributo all'interno dell'oggetto. Non può essere visto da nessun'altra classe. In questo modo solo la classe che lo contiene può decidere di modificarne il valore.
Un attributo dichiarato come **protected** è visto all'esterno solo dalle classi che appartengono alla stessa libreria oppure dalle sottoclassi della classe in cui è stato dichiarato l'attributo.
Se non viene specificato alcun livello di visibilità, l'attributo è visibile solo dalle classi che appartengono alla stessa libreria.
I seguenti esempi mostrano dichiarazioni di attributi con visibilità diverse:

```
private int a;
public int b;
protected int c;
int d;
```

In aggiunta ai livelli di visibilità, in fase di dichiarazione si possono specificare altre due caratteristiche per gli attributi:
- static
- final.

La parola **static** indica un attributo legato alla classe, nel senso che, se vengono creati più oggetti di quella classe, esiste solo una copia dell'attributo (il significato di *static* verrà illustrato nel dettaglio in un paragrafo successivo).
Un attributo è invece dichiarato **final** se lo si vuole rendere una costante, in modo che possa assumere un unico valore. Dopo aver ricevuto un valore iniziale, non può essere sottoposto a operazioni di assegnazione di valori diversi. Solitamente gli attributi *final* vengono anche dichiarati pubblici, perché non esiste il rischio che possano venire modificati dall'esterno in modo indesiderato.
Un esempio di dichiarazione per un attributo costante e pubblico è il seguente:

```
public final int DIECI = 10;
```

5 Dichiarazione dei metodi

I metodi rappresentano la parte dinamica di una classe. Servono per implementare le operazioni che una classe può eseguire: i metodi possono modificare gli attributi, cioè lo stato interno di un oggetto oppure calcolare i valori che verranno restituiti al richiedente.

Un metodo è caratterizzato da cinque diversi elementi che devono essere dichiarati:
- un nome,
- un blocco di istruzioni,
- un tipo di valore di ritorno,
- un elenco di parametri,
- un livello di visibilità.

La **dichiarazione dei metodi** deve essere fatta all'interno del blocco di una classe e assume la seguente struttura generale:

```
livelloDiVisibilità tipoRestituito nomeMetodo (parametri)
{
  // variabili
  // istruzioni
}
```

Gli elementi obbligatori sono il *tipoRestituito* e il *nomeMetodo*, mentre il *livelloDiVisibilità* e i *parametri* sono facoltativi e possono anche non essere presenti nella dichiarazione.

Il metodo che esegue la somma di due numeri interi passati come parametri viene dichiarato con il seguente frammento di codice.

```
public int addizione (int a, int b)
{
  int sum;

  sum = a+b;

  return sum;
}
```

Il **nome** del metodo (nell'esempio precedente, *addizione*) è una parola che identifica un insieme di istruzioni e, per convenzione, lo si fa iniziare con una lettera minuscola. Il nome potrebbe essere composto anche da un insieme di parole unite tra loro, in cui l'iniziale della prima parola è minuscolo, mentre le iniziali delle successive parole sono maiuscole; per esempio *sommaDiDueNumeri*.

Il **blocco di istruzioni** è identificato da una coppia di parentesi graffe, all'interno delle quali vengono specificate le istruzioni, intese come dichiarazioni di variabili e istruzioni operative.

La variabile (nell'esempio precedente, *sum*) usata all'interno del metodo è chiamata **variabile locale**. Le variabili locali, a differenza delle variabili di istanza, non permettono di indicare il livello di visibilità perché sono locali ai metodi. Vengono create quando viene eseguito il metodo e vengono distrutte non appena il metodo termina l'esecuzione. La loro durata corrisponde alla durata dell'esecuzione del metodo.
Oltre alle variabili, il blocco del metodo contiene le istruzioni che permettono di eseguire una determinata operazione. Si possono usare tutte le istruzioni mostrate nel capitolo precedente oppure, come vedremo, si possono richiamare altri metodi.
Non possono esistere istruzioni all'esterno dei blocchi definiti dai metodi.

Il **tipo del valore di ritorno** (nell'esempio precedente, *int*) viene specificato nella dichiarazione del metodo e indica quale sarà il valore restituito al termine dell'esecuzione del metodo. Il tipo può essere uno qualunque dei tipi di dato che sono usati in Java e deve essere indicato prima del nome del metodo.
Un metodo può anche non restituire alcun valore. Per esprimere questa situazione, nella dichiarazione viene usata la parola chiave **void** per indicare il tipo restituito.
Per esempio, il metodo *setRaggio* usato nella classe *Cerchio* non ha valori di ritorno:

```
public void setRaggio(double r)
{
  raggio = r;
}
```

Un metodo con questa dichiarazione termina la sua esecuzione quando viene raggiunta l'ultima istruzione contenuta nel blocco.
Al contrario, quando è indicato un tipo per il valore di ritorno, l'esecuzione del metodo termina quando viene eseguita l'istruzione **return**, che indica qual è il valore restituito. Il valore indicato dopo la parola chiave *return* deve essere dello stesso tipo di quello specificato nell'intestazione del metodo.
Solitamente l'istruzione *return* rappresenta l'ultima istruzione del metodo, ma in molti casi è più utile posizionarla all'interno del codice per interrompere in anticipo l'esecuzione del metodo.
Per esempio, nel seguente metodo che calcola il maggiore tra due numeri, le due istruzioni *return* sono posizionate all'interno del codice:

```
public int max(int a, int b)
{
  if (a > b)
  {
    return a;
  }
  else
  {
    return b;
  }
}
```

L'**elenco di parametri** (nell'esempio iniziale, i valori interi *a* e *b*) vengono elencati dopo il nome del metodo, all'interno delle parentesi tonde. I parametri sono separati con la virgola. Se un metodo non possiede i parametri, viene dichiarato usando solo le due parentesi tonde.

In Java, i parametri possono essere passati ai metodi solo **per valore**.

Ogni volta che viene richiamato un metodo, i parametri assumono il ruolo di variabili locali. Prima di iniziare l'esecuzione del metodo, il valore passato viene copiato all'interno di queste nuove variabili locali. Le eventuali modifiche apportate ai parametri all'interno del metodo non vengono applicate ai parametri originari.

Per quanto riguarda i **tipi riferimento**, cioè gli array e gli oggetti, viene creata una copia del riferimento e non dell'oggetto. L'array quindi non viene duplicato, viene creata una variabile locale che contiene la copia del riferimento allo stesso array. Ne segue che le modifiche all'interno del metodo, fatte usando il parametro, modificano anche l'array originale.

Nell'esempio seguente il metodo *raddoppia* riceve come parametro un array di numeri interi e li raddoppia modificando direttamente l'array originale.

```
public void raddoppia (int valori[])
{
  for(int i=0; i<valori.length; i++)
  {
    valori[i] *= 2;
  }
}
```

I **livelli di visibilità di un metodo**, come per gli attributi, specificano se il metodo può essere visto e richiamato da altri oggetti. I livelli di visibilità che possono essere specificati sono:
- public
- private
- protected.

Questi tre diversi livelli applicabili ai metodi, hanno lo stesso significato già visto per i livelli riferiti agli attributi: un metodo dichiarato **public** può essere richiamato da qualunque altra classe; un metodo **private** non può essere richiamato esternamente alla classe in cui è dichiarato; un metodo dichiarato **protected** è visibile solo dalle classi che appartengono alla stessa libreria oppure dalle sottoclassi della classe in cui è dichiarato.
Solitamente i metodi vengono dichiarati usando la parola chiave *public*. Il livello *private* viene scelto quando il metodo è usato solo all'interno della classe. Questo si verifica quando un'operazione può essere scomposta in diverse azioni e ad ognuna viene associato un metodo privato.

PROGETTO 2

Dichiarare una classe per rappresentare le informazioni di una persona e per registrare il suo contatto email.

Il progetto realizza una classe per immagazzinare i dati anagrafici di una persona, in particolare il nome, il cognome e il contatto email, oltre ad un valore booleano per indicare se l'email è stata registrata. Gli attributi *nome* e *cognome* sono accessibili in modo pubblico, mentre le informazioni sul contatto email sono private e modificabili solo tramite il metodo *registraEmail*. È stato definito anche un metodo *stampaDati* per visualizzare i dati della persona e il suo contatto solo nel caso sia stato registrato.

Anagrafica
nome cognome email registrata
registraEmail stampaDati

Dopo aver dichiarato la classe *Anagrafica*, nel programma viene creato un oggetto con la seguente istruzione:

```
Anagrafica contatto = new Anagrafica();
```

Avendo dichiarato gli attributi pubblici *nome* e *cognome*, il loro valore può essere assegnato con le due istruzioni:

```
contatto.nome = tastiera.readLine();
contatto.cognome = tastiera.readLine();
```

Infine, il valore degli attributi privati *email* e *registrata* viene impostato attivando il metodo *registraEmail* con la seguente istruzione:

```
contatto.registraEmail(email);
```

Il seguente esempio completo mostra la dichiarazione della classe *Anagrafica* e un suo utilizzo.

IMPLEMENTAZIONE DELLA CLASSE (*Anagrafica.java*)

```
class Anagrafica
{
  // attributi pubblici
  public String nome;
  public String cognome;

  // attributi privati
  private String email;
  private boolean registrata;

  public void registraEmail(String p_email)
  {
    email = p_email;
    registrata = true;
  }

  public void stampaDati()
  {
    System.out.println("Nome     = " + nome);
    System.out.println("Cognome  = " + cognome);
    if (registrata)
```

```
    {
      System.out.println("Email    = " + email);
    }
    else
    {
      System.out.println("Email non registrata");
    }
  }
}
```

PROGRAMMA JAVA (*ProgAnagrafica.java*)

```
import java.io.*;

class ProgAnagrafica
{
  public static void main(String argv[])
  {
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);

    // creazione dell'oggetto
    Anagrafica contatto = new Anagrafica();

    System.out.print("Inserisci il nome: ");
    try
    {
      contatto.nome = tastiera.readLine();
    }
    catch(IOException e) {}

    System.out.print("Inserisci il cognome: ");
    try
    {
      contatto.cognome = tastiera.readLine();
    }
    catch(IOException e) {}

    System.out.print("Inserisci l'email: ");
    try
    {
      String email = tastiera.readLine();
      contatto.registraEmail(email);
    }
    catch(IOException e) {}

    System.out.println("\nRIEPILOGO CONTATTO");
    contatto.stampaDati();
  }
}
```

PROGETTO 3

Dichiarare una classe per rappresentare un conto corrente e per gestire le operazioni di versamento e prelevamento.

Il progetto descrive la classe *ContoCorrente* che memorizza al suo interno il valore del saldo, con un attributo privato, e lo movimenta tramite i metodi pubblici *versa* e *preleva*. La classe dichiara altri due metodi pubblici per l'interrogazione del saldo, *getSaldo*, e per la sua visualizzazione, *stampaSaldo*.
La classe *ContoCorrente* è schematizzata con il seguente diagramma di classe.

ContoCorrente
saldo
versa preleva getSaldo stampaSaldo

L'attributo privato *saldo* contiene un valore di tipo *double* e viene dichiarato nel seguente modo:

```
private double saldo;
```

I metodi che gestiscono le operazioni di versamento e prelevamento sono dichiarati pubblici e non hanno un valore di ritorno (*void*), inoltre ricevono come parametro un valore *importo* da utilizzare per movimentare il saldo.
L'intestazione del metodo *versa*, che è simile a quella del metodo *preleva*, è la seguente:

```
public void versa(double importo)
```

L'unico metodo della classe che restituisce un valore di ritorno è *getSaldo*. Questo metodo rende visibile all'esterno della classe il valore dell'attributo *saldo*, che in quanto *private*, non sarebbe accessibile. La dichiarazione del metodo *getSaldo* contiene l'indicazione del tipo del valore di ritorno (*double*) e, all'interno del blocco di istruzioni, viene utilizzata la parola chiave *return* per restituire il valore.
Il seguente codice completo mostra la dichiarazione della classe *ContoCorrente* e un suo utilizzo.

IMPLEMENTAZIONE DELLA CLASSE (*ContoCorrente.java*)

```
class ContoCorrente
{
  // attributo privato
  private double saldo;

  public void versa(double importo)
  {
    saldo += importo;
  }

  public void preleva(double importo)
  {
    saldo -= importo;
  }
```

```
  public double getSaldo()
  {
    return saldo;
  }

  public void stampaSaldo()
  {
    System.out.println("SALDO = " + saldo);
  }
}
```

PROGRAMMA JAVA (*ProgContoCorrente.java*)

```
import java.io.*;

class ProgContoCorrente
{
  public static void main(String argv[])
  {
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);

    // creazione dell'oggetto
    ContoCorrente conto = new ContoCorrente();

    // dichiarazione e inizializzazione delle variabili
    String valore = "";
    double importo = 0;

    System.out.print("Importo da versare: ");
    try
    {
      valore = tastiera.readLine();
      importo = Double.valueOf(valore).doubleValue();
    }
    catch(IOException e) {}

    conto.versa(importo);
    conto.stampaSaldo();

    System.out.print("Importo da prelevare: ");
    try
    {
      valore = tastiera.readLine();
      importo = Double.valueOf(valore).doubleValue();
    }
    catch(IOException e) {}

    if (conto.getSaldo() >= importo)
    {
      conto.preleva(importo);
      conto.stampaSaldo();
    }
    else
    {
      System.out.println("Prelevamento non disponibile.");
    }
  }
}
```

6 Creazione degli oggetti

Come abbiamo visto nei paragrafi precedenti, le classi non possono essere sfruttate direttamente, ma è necessario **creare un oggetto**, cioè un'istanza particolare di quella classe.

Un'**istanza di classe** è in pratica una variabile (oggetto) che viene dichiarata indicando come tipo il nome della classe. Facendo un parallelo tra le variabili e gli oggetti, si può dire che alle variabili è associato un tipo, mentre agli oggetti è associata una classe.

Le classi in Java sono considerate **tipi di dato riferimento** per indicare che le variabili di questo tipo (istanze) contengono il riferimento a un oggetto.
La creazione di un oggetto può essere eseguita negli stessi punti dove vengono dichiarate le variabili e si compone dei seguenti passi:
- dichiarazione dell'oggetto,
- allocazione dell'oggetto.

La **dichiarazione di un oggetto** deve indicare il nome della classe seguito dal nome che si vuole associare all'oggetto, nel seguente modo:

```
NomeClasse nomeOggetto;
```

Per esempio, supponendo che esistano le classi *Rettangolo* e *Cerchio*, si possono dichiarare le seguenti istanze di classe:

```
// dichiarazione di un oggetto
Rettangolo schermo;

// dichiarazione di due oggetti dello stesso tipo
Cerchio tavolo, ruota;
```

L'**allocazione dell'oggetto** riserva lo spazio per memorizzare gli attributi dell'oggetto e viene codificata con l'operatore **new** seguito dal nome della classe. Questo operatore restituisce un riferimento a un'area di memoria in cui l'oggetto viene memorizzato.

Con l'allocazione dell'oggetto, viene attivato in modo implicito un particolare metodo predefinito, detto **costruttore** della classe, che consente di creare un oggetto.

Se una classe viene costruita senza specificare il costruttore, come è stato fatto per la classe *Cerchio* nei paragrafi precedenti, ad essa viene associato un costruttore vuoto, e, al momento della creazione di un nuovo oggetto, non vengono eseguite operazioni.
L'allocazione di due istanze della classe *Cerchio* a cui è associato un costruttore vuoto è mostrata nelle seguenti istruzioni. I due assegnamenti impostano nelle variabili a sinistra il riferimento ai corrispondenti oggetti.

```
tavolo = new Cerchio();
ruota = new Cerchio();
```

La creazione di un oggetto può essere raggruppata in una sola riga unendo la dichiarazione e l'allocazione:

```
Cerchio tavolo = new Cerchio();
```

Durante l'implementazione di una classe, oltre a definire attributi e metodi, si deve considerare anche la creazione di uno o più metodi *costruttori*: essi vengono eseguiti automaticamente ogni volta che viene creato un nuovo oggetto e sono solitamente usati per le operazioni di inizializzazione dell'oggetto.

Per esempio, nella definizione della classe *Cerchio* si può aggiungere il metodo costruttore *Cerchio* per l'assegnazione del valore iniziale per il raggio del cerchio.

```
class Cerchio
{
  private double raggio;

  // costruttore
  public Cerchio(double r)
  {
    raggio = r;
  }

  public void setRaggio(double r)
  {
    raggio = r;
  }

  public double area()
  {
    return (raggio * raggio * Math.PI);
  }
}
```

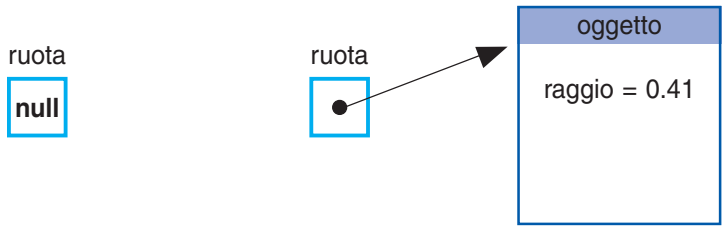
Il costruttore è un metodo appartenente alla classe e dichiarato usando lo stesso nome della classe. Non deve restituire alcun valore di ritorno e non deve nemmeno essere dichiarato *void*. Il livello di visibilità usato per i costruttori è solitamente *public*. Per compiere le operazioni di inizializzazione il costruttore può anche usare dei parametri.

Nell'esempio che segue viene creata un'istanza della classe *Cerchio* usando il nuovo costruttore e indicando un valore di inizializzazione per l'attributo *raggio*.

```
Cerchio ruota = new Cerchio(0.41);
```

Si noti che il parametro previsto dal costruttore della classe *Cerchio* viene aggiunto tra parentesi dopo l'operatore *new*.

La seguente figura riassume i passi nella creazione dell'oggetto *ruota*.



Dichiarazione dell'oggetto **Allocazione dell'oggetto**

La dichiarazione dell'oggetto crea una variabile vuota che contiene un valore particolare indicato con la parola chiave **null**. L'istanza di classe a cui viene assegnato il valore *null*, non fa riferimento ad alcun oggetto.
L'allocazione dell'oggetto inserisce nella variabile il riferimento all'area di memoria contenente i valori degli attributi dell'oggetto. Durante la fase di allocazione, la macchina virtuale Java controlla se c'è spazio sufficiente in memoria per contenere l'oggetto, cioè per memorizzare tutti i suoi attributi. Se non c'è spazio segnala un errore.
Le prime istruzioni che vengono eseguite, dopo l'allocazione dell'oggetto, sono quelle contenute nel metodo costruttore della classe il cui scopo principale è di assegnare i valori iniziali agli attributi dell'oggetto.

RIFERIMENTI NULLI

Una variabile definita come istanza di classe, durante l'esecuzione del programma, può assumere uno dei seguenti valori:
- *null*
- un riferimento a un oggetto il cui tipo è quello della classe
- un riferimento a un oggetto il cui tipo è quello di qualunque sottoclasse della classe.

In particolare, l'istanza può assumere il valore *null* in due casi:
- se fa riferimento a un oggetto dichiarato ma non allocato,
- se viene assegnato esplicitamente il valore *null*.

Nel primo caso, l'istanza è stata dichiarata con la seguente istruzione, a cui non è seguita l'operazione di allocazione con il comando *new*.

```
Cerchio tavolo;
```

Nel secondo caso, alla variabile è stato assegnato esplicitamente il valore *null* con la seguente istruzione:

```
tavolo = null;
```

In entrambi i casi, la variabile non fa riferimento ad un oggetto e per questo tutte le operazioni di accesso agli attributi o ai metodi della classe non possono essere eseguite.
Il seguente codice Java assegna il valore *null* a un oggetto e successivamente tenta di eseguire un suo metodo.

```
class ProgCerchioNull
{
  public static void main(String argv[])
  {
    Cerchio tavolo = new Cerchio(1.2);

    tavolo = null;

    System.out.println("Area = " + tavolo.area());
  }
}
```

Durante l'esecuzione del precedente programma, quando si tenta di invocare il metodo *area* dell'oggetto *null*, viene segnalata l'eccezione **NullPointerException** e il programma termina in errore.

La gestione dei riferimenti nulli diventa importante con i programmi di grandi dimensioni in cui gli oggetti possono essere creati in punti diversi da dove poi verranno utilizzati. Diventa quindi indispensabile, prima di utilizzare un oggetto, verificare che sia stato effettivamente creato e quindi non contenga un valore *null*. Questo controllo può essere fatto con la seguente istruzione di selezione:

```
if (tavolo != null)
{
  System.out.println("Area = " + tavolo.area());
}
```

UGUAGLIANZA TRA OGGETTI

Il concetto di uguaglianza tra oggetti è diverso rispetto all'uguaglianza tra variabili.
Due variabili sono uguali se contengono lo stesso valore. Per esempio,

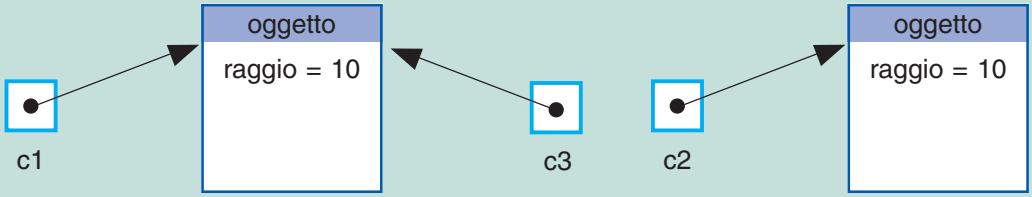
```
int v1 = 10;
int v2 = 10;
```

Le variabili *v1* e *v2* sono uguali: se si confrontano con l'operatore == viene restituito il valore *true*.

Due oggetti sono uguali se contengono il riferimento alla stessa area di memoria. Per esempio,

```
Cerchio c1 = new Cerchio(10);
Cerchio c2 = new Cerchio(10);
Cerchio c3 = c1;
```

Gli oggetti *c1* e *c2* sono diversi: se si confrontano con l'operatore == viene restituito il valore *false*. Invece gli oggetti *c1* e *c3* sono uguali.



Il confronto tra gli oggetti, usando l'operatore ==, verifica l'uguaglianza dei riferimenti piuttosto che l'uguaglianza degli attributi dell'oggetto.

All'interno dei programmi, si possono creare più riferimenti allo stesso oggetto tramite l'operazione di assegnamento. Nell'esempio precedente l'oggetto *c1* è stato assegnato all'oggetto *c3*.

La **copia dei riferimenti** si verifica anche quando un oggetto viene passato come parametro ad un metodo: il parametro rappresenta una copia del riferimento allo stesso oggetto.

Il seguente metodo dichiara un parametro *c4* di classe *Cerchio*:

```
public void stampa(Cerchio c4) { ... }
```

Al momento dell'invocazione del metodo, se viene passato il cerchio *c1* con l'istruzione *stampa(c1)*, si crea un ulteriore riferimento allo stesso oggetto: sia *c4* che *c1* puntano alla stessa area di memoria. All'interno del blocco del metodo viene usato il riferimento *c4*, ma le eventuali modifiche agli attributi dell'oggetto *c4* influenzeranno anche l'oggetto *c1*.

7 Utilizzo degli oggetti

Dopo aver creato l'oggetto, esso può essere utilizzato in due modi:
- accedendo e manipolando il valore dei suoi attributi,
- invocando ed eseguendo i suoi metodi.

Si può accedere agli attributi di un oggetto usando l'**operatore punto**.

```
nomeOggetto.attributo
```

Con *nomeOggetto* si intende un'istanza di classe, che sia stata precedentemente dichiarata e allocata. L'operatore punto viene inserito tra l'oggetto e l'attributo. Quest'ultimo si riferisce a un attributo visibile e quindi dichiarato con il livello di visibilità *public* o *protected*. Gli attributi *private* non sono accessibili con questa modalità.

L'**invocazione di un metodo** viene eseguita usando l'operatore punto nel seguente modo:

```
nomeOggetto.metodo(parametri)
```

Nel programmazione orientata agli oggetti si usa il termine **scambio di messaggi** per indicare l'interazione tra gli oggetti, realizzata tramite l'invocazione dei metodi.

Quando un oggetto vuole comunicare con un altro per eseguire un'azione, manda ad esso un messaggio.
Il *nomeOggetto* che precede il punto corrisponde al destinatario del messaggio, mentre il *metodo* deve essere uno dei metodi pubblici dichiarati nella classe del destinatario. Se il metodo richiede dei parametri, essi devono essere indicati alla fine tra parentesi tonde; se non ci sono parametri, le parentesi tonde vuote vanno comunque indicate.
Si noti che i metodi, per essere richiamati, devono possedere un corretto livello di visibilità: un metodo dichiarato *private* non può essere invocato con la modalità vista prima.

Un oggetto o un programma può interagire con un oggetto per diversi motivi: per modificarne lo stato, per richiedere un'informazione o per attivare un comportamento. Per esempio, un oggetto della classe *Cerchio* può ricevere i seguenti messaggi dal programma principale:
- il messaggio *setRaggio*, per modificare il valore del raggio;
- il messaggio *area*, per calcolare e ottenere il valore dell'area.

L'invocazione di un metodo, se restituisce un valore di ritorno, può essere posizionata alla destra di un'istruzione di assegnamento e in tutte le posizioni in cui vengono utilizzate delle espressioni. Per esempio,

```
superficie = tavolo.area();
```

In ogni classe è implicitamente presente un oggetto speciale identificato dalla parola chiave **this**. Questa parola chiave costituisce un riferimento all'oggetto medesimo, permettendo all'oggetto di riconoscere se stesso. Inoltre l'uso di *this* permette di risolvere le eventuali omonimie presenti tra i nomi degli attributi e i nomi dei parametri.
Per esempio, il costruttore della classe *Cerchio* può essere riscritto nel seguente modo:

```
// costruttore
public Cerchio(double raggio)
{
  this.raggio = raggio;
}
```

Si noti che il nome utilizzato per il parametro del costruttore, *raggio*, è identico al nome dell'attributo. L'uso dello stesso nome consente di non inventare nomi diversi di variabile, rendendo così più semplice la lettura del codice. Il parametro *raggio*, all'interno del costruttore, diventa una variabile locale che maschera l'attributo dichiarato con lo stesso nome. Per poter accedere all'attributo si deve usare l'operato punto e la parola chiave **this** per indicare l'oggetto a cui il costruttore si riferisce.

PROGETTO 4

Scrivere il programma che calcola la circonferenza e l'area di due cerchi, con arrotondamento alle prime tre cifre decimali per il calcolo dell'area del primo cerchio.

La classe usata per risolvere questo problema è simile a quella presentata nel Progetto 1, ma viene adattata per rispondere alle esigenze specifiche di questo progetto. La richiesta di calcolare la circonferenza del cerchio è risolta aggiungendo un nuovo metodo *circonferenza*, allo stesso modo l'arrotondamento dell'area è gestito dal metodo *areaArrotondata*.

CerchioArrot
raggio
setRaggio area areaArrotondata circonferenza

I calcoli sui due cerchi possono essere implementati creando due istanze della classe *CerchioArrot* oppure, come faremo, creando un solo oggetto e inizializzandolo con il valore del raggio del primo cerchio.

```
CerchioArrot c = new CerchioArrot(5.5);
```

Dopo aver calcolato la circonferenza e l'area del primo cerchio, aggiorneremo il valore del raggio per eseguire i calcoli sul secondo cerchio, invocando il metodo *setRaggio*.

```
c.setRaggio(8.0);
```

L'algoritmo di arrotondamento utilizza il metodo statico **Math.round**, che riceve come parametro un numero reale e restituisce il valore intero più vicino. Per esempio:

```
area = 4.32186;
temp = Math.round(area);       // temp vale 4
area = 4.78945;
temp = Math.round(area);       // temp vale 5
```

L'arrotondamento alla terza cifra decimale può essere calcolato con la seguente sequenza di istruzioni:

```
area = 4.32186;
temp = area * 1000;            // temp vale 4321.86
temp = Math.round(temp);       // temp vale 4321
temp = (double) temp / 1000;   // temp vale 4.321
```

Il metodo *areaArrotond* utilizza questo algoritmo per arrotondare e restituire il valore dell'area calcolato dal metodo *area*.

Di seguito è riportato il codice completo dell'implementazione della classe e del programma.

IMPLEMENTAZIONE DELLA CLASSE (*CerchioArrot.java*)

```
class CerchioArrot
{
  private double raggio;

  // costruttore
  public CerchioArrot(double raggio)
  {
    this.raggio = raggio;
  }

  // metodo per modificare il raggio
  public void setRaggio(double r)
  {
    raggio = r;
  }

  // metodo per calcolare l'area
  public double area()
  {
    return (raggio * raggio * Math.PI);
  }

  // metodo per calcolare l'area arrotondata
  public double areaArrotondata()
  {
    double areaTemp;
```

```
    areaTemp = this.area() * 1000;
    areaTemp = Math.round(areaTemp);
    areaTemp = (double) areaTemp / 1000;

    return areaTemp;
  }

  // metodo per calcolare la circonferenza
  public double circonferenza()
  {
    return (2 * raggio * Math.PI);
  }
}
```

PROGRAMMA JAVA (*ProgCerchioArrot.java*)

```
class ProgCerchioArrot
{
  public static void main(String argv[])
  {
    CerchioArrot c = new CerchioArrot(5.5);

    System.out.println("Primo cerchio.");
    System.out.println("Circonferenza = " + c.circonferenza() );
    System.out.println("Area = " + c.areaArrotondata() );

    c.setRaggio(8.0);
    System.out.println("Secondo cerchio.");
    System.out.println("Circonferenza = " + c.circonferenza() );
    System.out.println("Area = " + c.area() );
  }
}
```

Si noti che all'interno del metodo *areaArrotondata* viene invocato il metodo *area* usando l'oggetto speciale **this**. In questa situazione l'uso del *this* non è necessario, infatti si può anche sostituire la riga con la seguente:

```
areaTemp = area() * 1000;
```

Il *this* è servito per sottolineare la possibilità di poter invocare un metodo anche all'interno della classe in cui è stato definito, e non solo dagli oggetti esterni. Questa tecnica di richiamare i metodi interni può essere applicata nella scomposizione di un comportamento complesso in metodi più semplici, e questi ultimi, se devono essere inaccessibili dall'esterno, possono essere dichiarati come *private*.

ATTRIBUTI E METODI *STATIC*

Un attributo o un metodo, dichiarati usando la parola chiave **static**, possiedono caratteristiche particolari che li differenziano dagli altri dichiarati non *static*.
Un **attributo statico** viene anche chiamato **variabile di classe**, perché è un attributo legato alla classe: se vengono creati più oggetti per quella classe, esiste solo una copia dell'attributo. Normalmente, per gli attributi non statici, ne viene creata una copia per ogni oggetto istanziato. Per gli attributi *static*, invece, esiste una sola copia. Tutti gli oggetti di quella classe possono fare riferimento agli unici attributi statici. Un attributo statico è quindi condiviso da tutte le istanze della classe. Tutte possono leggere e modificare l'attributo comune.
Un esempio di attributo statico è l'attributo **PI**, contenente il valore di *pigreco* e dichiarato nella classe *Math*, che fa parte delle librerie di Java:

```
public static final double PI;
```

Questo attributo è dichiarato come costante, usando la parola chiave *final*, ed è accessibile da chiunque.
Un attributo *static* è accessibile in un modo diverso dagli altri attributi. Normalmente gli attributi sono accessibili usando l'operatore punto preceduto dal *nome dell'oggetto* a cui ci si riferisce.
Gli attributi statici possono essere indicati anche in un altro modo, usando l'operatore punto preceduto dal *nome della classe*, perché gli attributi statici sono unici; per questo motivo si chiamano anche variabili di classe.

```
NomeClasse.attributoStatic
```

Per esempio, usando *Math.PI*, ci si riferisce all'attributo statico *PI*. Non è stato necessario creare un oggetto di classe *Math* per usare l'attributo.
I **metodi statici** sono anche chiamati **metodi di classe** perché vengono invocati usando il nome della classe e non quello dell'oggetto:

```
NomeClasse.metodoStatic(parametri)
```

I metodi statici, o di classe, possono quindi essere utilizzati senza dover creare in anticipo un'istanza della classe.
Abbiamo già visto l'esempio del metodo **random()**, che è un metodo presente nella classe *Math* e dichiarato nel seguente modo:

```
public static double random()
```

È un metodo dichiarato pubblico e statico e restituisce un valore di tipo *double*. Per richiamare questo metodo statico, basta far precedere all'operatore punto il nome della sua classe: *Math.random()*.
I metodi statici possono operare solo su attributi statici e richiamare altri metodi statici.

AUTOVERIFICA

Domande da 5 a 14 pag. 225-226
Problemi da 13 a 19 pag. 229-230

MATERIALI ON LINE

3. Dichiarazione e utilizzo di un metodo *static*
4. *Singleton*: le classi con una sola istanza

8 Mascheramento dell'informazione nelle classi

I linguaggi di programmazione ad oggetti, per distinguere ciò che una classe vuole mostrare all'esterno da ciò che vuole tenere nascosto, permettono di definire una sezione *pubblica* e una sezione *privata* all'interno della classe.
Come abbiamo visto nei paragrafi precedenti, il *livello di visibilità* degli attributi e dei metodi permettono di identificare ciò che rientra nella sezione pubblica (livello *public*) e ciò che rientra nella sezione privata (livello *private*).

Nella **sezione pubblica** devono essere inseriti gli attributi e i metodi che si vuole rendere visibili all'esterno e quindi utilizzabili da altri oggetti. Ciò che è presente in questa sezione rappresenta l'interfaccia tra l'oggetto e l'ambiente esterno.
Nella **sezione privata** ci sono gli attributi e i metodi che non sono accessibili e che vengono usati dall'oggetto per implementare i suoi comportamenti. Si inseriscono in questa sezione le strutture dati private ed eventuali procedure usate solo dall'oggetto stesso.

I livelli di visibilità consentono di realizzare con le classi la tecnica dell'*information hiding*.

Il termine **information hiding** indica il mascheramento delle modalità di implementazione di un'oggetto, rendondone disponibile all'esterno solo le funzionalità.

L'applicazione pratica dell'**information hiding**, secondo cui devono essere tenuti nascosti i dettagli implementativi della classe, consiste nel dichiarare gli attributi usando il livello di visibilità *private*. In questo modo le strutture dati di un oggetto risultano nascoste e non sono direttamente accessibili per la lettura e per la modifica.
Una classe che possiede attributi dichiarati come *private*, può comunque permettere la loro lettura e modifica definendo opportuni metodi. In particolare la modifica può essere gestita tramite i metodi con livello di visibilità *public*, all'interno dei quali si possono inserire gli opportuni controlli per validare le modifiche richieste.

Solitamente i due metodi per leggere e modificare il valore degli attributi vengono indicati con i termini inglesi *get* e *set*.
Il metodo **get** è usato per leggere il valore di un attributo e restituirlo.
Il metodo **set** è usato per modificare il valore di un attributo.

Per esempio:

```
class Automobile
{
  private final int POSIZIONE_MAX = 6;
  private int posMarcia;

  public Automobile()
  {
    posMarcia = 1;
  }

  public int getMarcia()
  {
    return posMarcia;
  }
```

```
    public void setMarcia(int m)
    {
      if ((m >= 1) && (m <= POSIZIONE_MAX))
      {
        posMarcia = m;
      }
    }
  }
```

In questo esempio viene dichiarata una classe chiamata *Automobile* con un costruttore che inizializza il valore dell'attributo *posMarcia* a 1.

Il metodo *getMarcia* dichiara come valore di ritorno un intero e ha il compito di restituire il valore dell'attributo *posMarcia*. Poiché l'attributo è *private* e non può essere visibile al di fuori della classe, l'uso del metodo pubblico consente la lettura del valore dell'attributo anche all'esterno.
Il metodo *setMarcia* riceve come parametro un valore intero e lo assegna all'attributo *posMarcia*. Il suo compito è quello di modificare il valore dell'attributo solo se il nuovo valore rispecchia certi criteri di validità. In questo modo non viene modificato direttamente il valore dell'attributo, che viene invece cambiato indirettamente attraverso l'esecuzione di un metodo.

Nella programmazione ad oggetti si utilizza spesso anche il termine *interfaccia* riferito alle classi.

L'**interfaccia** di una classe indica l'elenco dei metodi pubblici, cioè l'insieme delle funzionalità utilizzabili dalle istanze della classe.

Per esempio, l'interfaccia della classe *Automobile* è rappresentata dai tre metodi:

```
  public Automobile()
  public int getMarcia()
  public void setMarcia(int m)
```

I programmi che vogliono usare la classe *Automobile* devono conoscere la sua interfaccia, in particolare hanno bisogno di sapere il nome dei metodi pubblici, il tipo dei valori di ritorno, il numero e il tipo dei parametri.
L'interfaccia descrive all'utilizzatore qual è il costruttore da usare per creare nuovi oggetti, quali sono i metodi che possono essere invocati, quali sono i parametri da passare e che cosa l'utilizzatore riceverà come valore di ritorno.
L'utilizzatore può ignorare come è stata concretamente implementata la classe.

L'applicazione dell'*information hiding* e la definizione di una corretta *interfaccia* permettono di semplificare la manutenzione del codice nel tempo. Se la modifica viene fatta rispettando l'interfaccia, l'utilizzatore della classe non deve apportare modifiche alle sue modalità di interazione con l'oggetto. L'eventuale modifica è quindi invisibile (*trasparente*) all'esterno dell'oggetto, garantendo l'*information hiding*.

Per esempio, se aggiungiamo una nuova struttura dati, *arrayMarce*, per memorizzare le marce in un array di booleani, dobbiamo modificare il codice dei metodi pubblici senza cambiare l'interfaccia della classe *Automobile*. In questo modo, la modifica risulterà trasparente al programma che usa la classe *Automobile*.

La nuova classe *Automobile* ha un'implementazione diversa, ma mantiene inalterata la sua interfaccia.

```
class Automobile
{
  private final int POSIZIONE_MAX = 6;
  private int posMarcia;
  private boolean arrayMarce[];

  public Automobile()
  {
    posMarcia = 1;
    arrayMarce = new boolean[POSIZIONE_MAX];
    arrayMarce[0] = true;
  }

  public int getMarcia()
  {
    return posMarcia;
  }

  public void setMarcia(int m)
  {
    if ((m >= 1) && (m <= POSIZIONE_MAX))
    {
      arrayMarce[posMarcia] = false;
      posMarcia = m;
      arrayMarce[m] = true;
    }
  }
}
```

Nello sviluppo dei programmi ad oggetti di grandi dimensioni, chi usa un certo oggetto solitamente non è interessato a conoscere come è stato implementato. Infatti, molto spesso alcune classi del programma sono costruite da un programmatore diverso da colui che le utilizzerà istanziando gli oggetti concreti. A quest'ultimo interessa sapere come interagire con l'oggetto, quali sono i metodi che mette a disposizione e come poterli richiamare, cioè è interessato all'*interfaccia delle classi*.

Il programmatore è come il meccanico che conosce i dettagli del funzionamento dell'automobile, mentre il pilota che la utilizza può ignorare questi dettagli. Questo modo di intendere gli oggetti induce a considerare le classi come una scatola nera (**blackbox**). I dettagli sulle caratteristiche e la struttura dell'oggetto sono nascosti all'interno, garantendo l' *information hiding* (mascheramento dell'informazione).

Partendo dai principi dell'*information hiding* si è sviluppato un nuovo modo di concepire il software come formato da scatole, che mettono a disposizione un insieme di funzionalità. È nata un'industria di componenti software. Il contenuto di questi componenti resta nascosto, ma si conosce il modo con cui operano. Per creare nuovi programmi si mettono insieme diverse componenti, facendole interagire nel modo voluto.

PROGETTO 5

Rappresentare una frazione con numeratore e denominatore come una classe e operare su di essa la riduzione ai minimi termini.

Gli elementi che compongono la frazione, il numeratore e il denominatore, sono incapsulati nella classe *Frazione* e sono dichiarati come attributi interi e privati.
L'interfaccia pubblica della classe *Frazione* è rappresentata, oltre che dal costruttore, dal metodo *semplifica* per eseguire la riduzione ai minimi termini e dal metodo *mostra* per visualizzare il numeratore e il denominatore separati dalla linea di frazione.
Il metodo *calcolaMCD* contiene l'algoritmo per calcolare il massimo comun divisore tra numeratore e denominatore. Il metodo non fa parte dell'interfaccia pubblica ma viene dichiarato come privato ed è utilizzato solo internamente alla classe, dal metodo *semplifica*.
Il diagramma della classe *Frazione* è mostrato nella figura.

Frazione
numeratore denominatore
calcolaMCD semplifica mostra

Il programma crea un oggetto di classe *Frazione* e imposta il valore degli attributi mascherati con il costruttore, come mostrato dalla seguente istruzione:

```
fraz = new Frazione(num, den);
```

Il programma può ignorare i dettagli con cui è stato implementato il meccanismo di riduzione ai minimi termini, ma lo può utilizzare invocando il metodo pubblico *semplifica* con la seguente istruzione:

```
semplificata = fraz.semplifica();
```

Il metodo *semplifica* restituisce il valore booleano *true* per indicare che la frazione è stata semplificata, altrimenti restituisce il valore *false* se il numeratore e il numeratore sono già espressi ai minimi termini.
Nel seguito è riportata la dichiarazione della classe e un programma di esempio.

IMPLEMENTAZIONE DELLA CLASSE (*Frazione.java*)

```
class Frazione
{
  // attributi mascherati
  private int numeratore = 1;
  private int denominatore = 1;

  // costruttore
  // imposta solo valori positivi
  public Frazione(int num, int den)
  {
    if (num > 0)
    {
      numeratore = num;
    }
```

```
    if (den > 0)
    {
      denominatore = den;
    }
  }

  // metodo mascherato
  private int calcolaMCD(int a, int b)
  {
    int temp, resto;

    // ordina i due valori
    if (a < b)
    {
      temp = a;
      a = b;
      b = temp;
    }

    resto = a % b;

    while (resto != 0)
    {
      a = b;
      b = resto;
      resto = a % b;
    }

    return b;
  }

  // metodo di interfaccia
  public boolean semplifica()
  {
    int mcd = calcolaMCD(numeratore, denominatore);

    if (mcd != 1)
    {
      numeratore = numeratore / mcd;
      denominatore = denominatore / mcd;
      return true;
    }

    return false;
  }

  // metodo di interfaccia
  public void mostra()
  {
    System.out.println(numeratore);
    System.out.println("------------");
    System.out.println(denominatore);
  }
}
```

PROGRAMMA JAVA (*ProgRiduzione.java*)

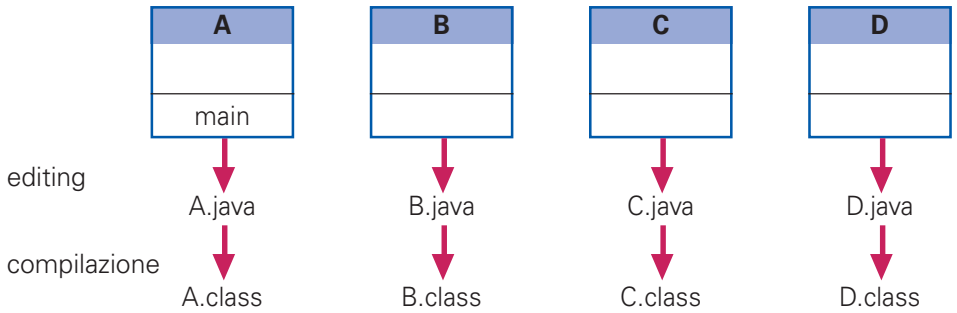
```
import java.io.*;

class ProgRiduzione
{
  public static void main(String argv[])
  {
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);

    // dichiarazione dell'oggetto
    Frazione fraz;

    // dichiarazione delle variabili
    String valore = "";
    int num = 1;
    int den = 1;
    boolean semplificata;

    System.out.print("Inserisci il numeratore: ");
    try
    {
      valore = tastiera.readLine();
      num = Integer.valueOf(valore).intValue();
    }
    catch(IOException e) {}

    System.out.print("Inserisci il denominatore: ");
    try
    {
      valore = tastiera.readLine();
      den = Integer.valueOf(valore).intValue();
    }
    catch(IOException e) {}

    // creazione dell'oggetto
    fraz = new Frazione(num, den);

    System.out.println("\nFrazione inserita");
    fraz.mostra();

    semplificata = fraz.semplifica();

    if (semplificata)
    {
      System.out.println("Frazione semplificata");
      fraz.mostra();
    }
    else
    {
      System.out.println("Non puo' essere semplificata.");
    }
  }
}
```

9 Realizzazione di programmi *object-oriented*

Nei linguaggi di programmazione orientati ad oggetti, l'attività principale consiste nella progettazione delle classi. Questa attività, eseguita durante l'analisi del problema, ha il compito di individuare le classi, dettagliando le loro caratteristiche (attributi) e il modo con cui interagiscono tra di loro (metodi). Inizialmente, le classi possono essere descritte in maniera sintetica utilizzando il diagramma delle classi. La fase successiva alla progettazione delle classi è la loro implementazione tramite un linguaggio di programmazione.

Con il linguaggio Java, una classe viene implementata con i seguenti passi:
- dichiarazione della classe, specificando:
  il nome,
- dichiarazione degli attributi, specificando:
  il livello di visibilità,
  il tipo,
  il nome,
- dichiarazione dei metodi, specificando:
  il nome,
  le istruzioni,
  il tipo del valore di ritorno,
  l'elenco di parametri,
  il livello di visibilità.

Un programma *object-oriented* scritto in Java è solitamente composto da più classi.
Anche se è possibile fare diversamente, conviene salvare su disco ogni classe in un file separato. In questo modo, a ogni classe viene associato un file con estensione **.java** che, una volta compilato, genera il corrispondente file **.class**.
Tra tutte le classi di un programma, una deve contenere il metodo **main**, che è il primo metodo attivato dall'interprete durante l'esecuzione del programma. Se un programma è composto da più classi, l'interprete Java deve essere richiamato indicando il nome della classe con il metodo *main*.



In figura è presentato un programma Java composto da quattro classi: *A*, *B*, *C*, *D*.
La classe *A* contiene il solo metodo *main* e, per questo motivo, la sua rappresentazione con il diagramma delle classi risulta poco significativa e non viene mostrata nei progetti descritti in questo capitolo. L'esecuzione del precedente programma composto da quattro classi inizia dopo aver invocato l'interprete Java con il seguente comando:

```
java A
```

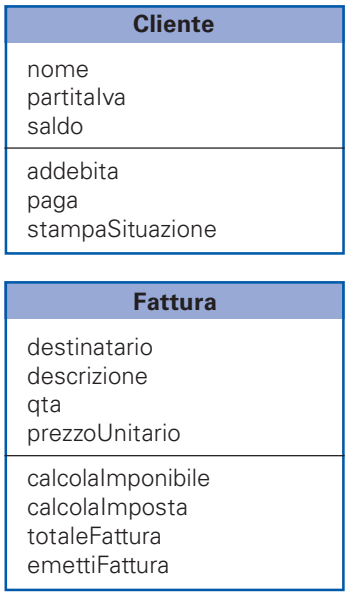
dove *A* è il nome della classe contenente il metodo *main*. Dopo il comando *java* va inserito il nome del file, senza indicare l'estensione *.class*.

PROGETTO 6

Realizzare un programma per gestire la fatturazione, addebitando le fatture ai clienti e accreditando i pagamenti ricevuti.

Il progetto richiede la descrizione di un sistema di fatturazione che limiteremo al caso di emissione della fattura ad un cliente per un solo prodotto. Le entità rilevanti per questa descrizione sono il *Cliente* e la *Fattura*. I clienti sono descritti con i loro dati anagrafici e il saldo dei loro pagamenti. Il saldo aumenta se al cliente viene addebitata una fattura, al contrario diminuisce se il cliente esegue un pagamento. La fattura è intestata ad un cliente e contiene la descrizione del prodotto, la quantità e il prezzo unitario. Il totale della fattura viene calcolato sommando l'imponibile (prezzo unitario * quantità) e l'imposta (imponibile * aliquota IVA).

Il diagramma delle classi *Cliente* e *Fattura* è mostrato nella figura.



Il costruttore della classe *Cliente* inizializza gli attributi *nome* e *partitaIva* con il valore dei parametri e azzera l'attributo *saldo*. I metodi *addebita* e *paga* gestiscono la posizione debitoria o creditoria di ogni cliente, aumentando il saldo per ogni fattura emessa e togliendo dal saldo quanto incassato. Il metodo *stampaSituazione* visualizza i dati del cliente e il suo saldo.
Si noti che tra gli attributi della classe *Fattura* è stato definito il *destinatario*, cioè il cliente a cui verrà addebitata. Ogni volta che viene creata una fattura deve essere indicato come parametro l'oggetto di classe *Cliente* a cui si riferisce, come mostra la seguente istruzione:

```
Fattura ordine1 = new Fattura(computerSpa);
```

I metodi privati *calcolaImponibile* e *calcolaImposta* vengono utilizzati all'interno del metodo *totaleFattura* per calcolare il totale della fattura.
Il metodo pubblico *emettiFattura* visualizza il totale e lo addebita al cliente.

L'implementazione delle classi *Cliente* e *Fattura*, e un programma di esempio, composto da una classe contenente il metodo *main*, sono riportati nelle pagine successive.

IMPLEMENTAZIONE DELLA CLASSE (*Cliente.java*)

```
class Cliente
{
  private String nome;
  private String partitaIva;
  private double saldo;

  public Cliente(String nome, String partitaIva)
  {
    this.nome = nome;
    this.partitaIva = partitaIva;
    saldo = 0;
  }

  public void addebita(double importo)
  {
    saldo += importo;
  }

  public void paga(double importo)
  {
    saldo -= importo;
  }

  public void stampaSituazione()
  {
    System.out.println("--------------------");
    System.out.println("Cliente     = " + nome);
    System.out.println("Partita IVA = " + partitaIva);
    System.out.println("Saldo       = " + saldo);
    System.out.println("--------------------");
  }
}
```

IMPLEMENTAZIONE DELLA CLASSE (*Fattura.java*)

```
class Fattura
{
  // costante
  private final int percIva = 20;

  // attributi
  private Cliente destinatario;

  public String descrizione;
  public int qta;
  public double prezzoUnitario;

  public Fattura(Cliente dest)
  {
    destinatario = dest;
  }
```

```
  private double calcolaImponibile()
  {
    return (qta * prezzoUnitario);
  }

  private double calcolaImposta()
  {
    double imp = calcolaImponibile();

    return (imp * percIva) / 100;
  }

  // calcola il totale fattura
  public double totaleFattura()
  {
    double totale;

    totale = calcolaImponibile() + calcolaImposta();

    return totale;
  }

  public void emettiFattura()
  {
    double totale = totaleFattura();

    System.out.println("Totale fattura = " + totale);

    // addebita la fattura al cliente
    destinatario.addebita(totale);
    destinatario.stampaSituazione();
  }
}
```

PROGRAMMA JAVA (*ProgFatturazione.java*)

```
import java.io.*;

class ProgFatturazione
{
  public static void main(String argv[])
  {
    // creazione del cliente
    Cliente computerSpa = new Cliente("Computer SPA", "01122334455);

    // crea due fatture per lo stesso cliente
    Fattura ordine1 = new Fattura(computerSpa);
    Fattura ordine2 = new Fattura(computerSpa);
```

```
    // registra i dati delle fatture e le addebita
    ordine1.descrizione = "Mouse";
    ordine1.qta = 28;
    ordine1.prezzoUnitario = 17.5;
    ordine1.emettiFattura();

    ordine2.descrizione = "Monitor";
    ordine2.qta = 7;
    ordine2.prezzoUnitario = 328.99;
    ordine2.emettiFattura();

    // incassa il pagamento di un bonifico
    double bonifico = 1000.0;
    System.out.println("Pagamento = " + bonifico);
    computerSpa.paga(bonifico);
    computerSpa.stampaSituazione();
  }
}
```

10 Array di oggetti

La dichiarazione e l'allocazione di un array di oggetti funzionano nello stesso modo con cui vengono dichiarati e allocati gli array basati sui tipi predefiniti. Con riferimento alla classe *Cerchio* utilizzata nei progetti precedenti, la dichiarazione di un array di 10 cerchi è la seguente:

```
Cerchio collezione[] = new Cerchio[10];
```

Si noti che vengono usate le parentesi quadre per dichiarare la variabile *collezione* e per indicare la dimensione dell'array: in questo momento i singoli elementi non esistono ancora. Dopo la dichiarazione dell'array si devono allocare i singoli oggetti: per ogni oggetto della collezione si deve predisporre lo spazio in memoria nel seguente modo:

```
collezione[0] = new Cerchio(0.58);
collezione[1] = new Cerchio(4.3);
```

e così via, indicando tra parentesi tonde il valore del raggio.

L'elemento *collezione[0]* fa riferimento al primo cerchio, l'elemento *collezione[1]* fa riferimento al secondo cerchio e così via.

L'uso degli array è utile nelle situazioni in cui si devono creare molti oggetti della stessa classe e li si vuole gestire in maniera efficace. Hanno però la limitazione che la dimensione deve essere prestabilita e non è più possibile modificarla a meno di creare un nuovo array.
Vedremo comunque nel prossimo capitolo che il linguaggio Java mette a disposizione, tra le sue librerie, una classe che gestisce i vettori di dimensione variabile; questa classe rende più flessibile la manipolazione delle collezioni di oggetti.

PROGETTO 7

Si vogliono confrontare 5 diversi modelli di hard disk. A ogni modello viene assegnato un punteggio nel seguente modo: per la velocità viene assegnato 1 punto ogni giro al minuto, per il tempo di accesso vengono assegnati –200 punti al millisecondo (più alto è il tempo, più basso è il punteggio), per la capacità vengono assegnati 500 punti ogni Gigabyte.
Dopo aver inserito i dati dei 5 modelli, calcolare il punteggio medio assegnato agli hard disk e mostrare i dati dell'hard disk migliore e di quello peggiore.

Le caratteristiche degli hard disk utili per la soluzione di questo problema sono: la marca, la velocità (espressa in RPM, rotazioni per minuto), il tempo di accesso (espresso in millisecondi) e la capacità (espressa in Gigabyte). Ogni caratteristica viene codificata come un attributo privato della classe. I valori di questi attributi vengono letti con il metodo *leggiDati* e vengono visualizzati con il metodo *stampaDati*. Il punteggio viene calcolato con il metodo *punteggio*. Il diagramma della classe è quindi il seguente:

Harddisk
marca rpm accesso capacita
leggiDati punteggio stampaDati

La classe *Hardisk* contiene inoltre tre attributi privati costanti per indicare i punti assegnati ad ogni caratteristica dell'hard disk. Le costanti, il cui identificatore è normalmente scritto tutto in maiuscolo, sono raggruppate all'inizio della dichiarazione della classe: questo ne facilita un'eventuale modifica successiva senza dover intervenire all'interno del codice dei metodi.

```
private final int PUNTI_RPM      = 1;
private final int PUNTI_ACCESSO  = -200;
private final int PUNTI_CAPACITA = 500;
```

Gli hard disk sono contenuti in un array di oggetti, che rappresenta un array di istanze della classe *Harddisk*. La dichiarazione dell'array è la seguente:

```
Harddisk hd[] = new Harddisk[MAX_HD];
```

Il valore *MAX_HD*, numero massimo di hard disk, è dichiarato come costante all'inizio del metodo *main* per permettere una sua eventuale modifica successiva in modo semplice.

```
final int MAX_HD = 5;
```

Si noti che *MAX*_HD è definito all'interno di un metodo e quindi non è un attributo. Per questo non deve specificare il livello di visibilità *private* come invece è stato fatto per i primi tre attributi.
Di seguito è riportato il codice completo della classe *Harddisk* e del programma.

IMPLEMENTAZIONE DELLA CLASSE (*Harddisk.java*)

```
import java.io.*;

class Harddisk
{
  // attributi
  private String marca;
  private int rpm;
  private double accesso;
  private double capacita;

  // costanti per il calcolo del punteggio
  private final int PUNTI_RPM       = 1;
  private final int PUNTI_ACCESSO   = -200;
  private final int PUNTI_CAPACITA  = 500;

  // legge gli attributi dell'hard disk
  public void leggiDati(int i)
  {
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);

    System.out.println("\nHARD DISK " + i);

    System.out.print("Inserisci la marca: ");
    try
    {
      marca = tastiera.readLine();
    }
    catch(IOException e) {}

    System.out.print("Inserisci la velocita' (rpm): ");
    try
    {
      String numeroLetto = tastiera.readLine();
      rpm = Integer.valueOf(numeroLetto).intValue();
    }
    catch(Exception e)
    {
      System.out.println("\nVelocita' non corretta!");
      System.exit(1);
    }

    System.out.print("Inserisci il tempo di accesso (ms): ");
    try
    {
      String numeroLetto = tastiera.readLine();
      accesso = Double.valueOf(numeroLetto).doubleValue();
    }
```

```
    catch(Exception e)
    {
      System.out.println("\nTempo di accesso non corretto!");
      System.exit(1);
    }

    System.out.print("Inserisci la capacita' (Gb): ");
    try
    {
      String numeroLetto = tastiera.readLine();
      capacita = Double.valueOf(numeroLetto).doubleValue();
    }
    catch(Exception e)
    {
      System.out.println("\nCapacita' non corretta!");
      System.exit(1);
    }
  }

  // formatta i dati e li stampa su standard output
  public void stampaDati()
  {
    System.out.println("Marca            = " + marca);
    System.out.println("Velocita'        = " + rpm + " rpm");
    System.out.println("Tempo di accesso = " + accesso + " ms");
    System.out.println("Capacita'        = " + capacita + " Gb");
    System.out.println("Punteggio        = " + punteggio());
  }

  // restituisce il punteggio assegnato a questo HD
  public int punteggio()
  {
    int punt = 0;

    punt += (int) (rpm      * PUNTI_RPM);
    punt += (int) (accesso  * PUNTI_ACCESSO);
    punt += (int) (capacita * PUNTI_CAPACITA);

    return punt;
  }
}
```

PROGRAMMA JAVA (*Proghd.java*)

```
class Proghd
{
  public static void main(String argv[])
  {
    // costante
    final int MAX_HD = 5;
```

```
    // variabili
    int totalePunteggio = 0;
    int punteggioMedio = 0;

    // oggetti
    Harddisk peggiore, migliore;

    // array di oggetti
    Harddisk hd[] = new Harddisk[MAX_HD];

    // crea gli harddisk e legge i dati
    for(int i=0; i<hd.length; i++)
    {
      hd[i] = new Harddisk();

      hd[i].leggiDati(i);
    }

    peggiore = hd[0];
    migliore = hd[0];

    // calcola il migliore-peggiore e il punteggio medio
    for(int i=0; i<hd.length; i++)
    {
      totalePunteggio += hd[i].punteggio();

      if (hd[i].punteggio() < peggiore.punteggio())
      {
        peggiore = hd[i];
      }
      else if (hd[i].punteggio() > migliore.punteggio())
      {
        migliore = hd[i];
      }
    }

    punteggioMedio = totalePunteggio/MAX_HD;

    System.out.println("\nPunteggio medio = " + punteggioMedio);
    System.out.println("\n*** HardDisk migliore ***");
    migliore.stampaDati();
    System.out.println("\n*** HardDisk peggiore ***");
    peggiore.stampaDati();
  }
}
```

L'istruzione **System.exit(1)** invoca il metodo statico **exit** della classe **System**. La sua chiamata provoca l'interruzione dell'esecuzione del programma Java. Il parametro passato indica il codice di stato che viene restituito al sistema operativo. Per convenzione un valore diverso da zero indica che l'esecuzione è terminata con un errore.

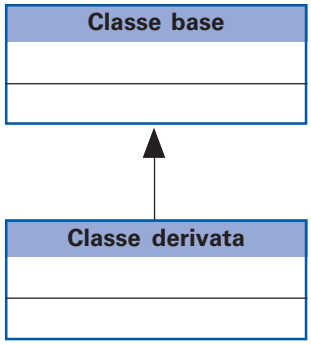
AUTOVERIFICA
Domande da 15 a 16 pag. 227
Problemi da 20 a 27 pag. 230

11 Ereditarietà

Come abbiamo già visto, sviluppare un sistema ad oggetti significa trovare e definire le classi necessarie per risolvere il problema, dalle quali verranno creati gli oggetti. Non sempre occorre partire dal nulla nel costruire una classe, soprattutto se si dispone già di una classe che è simile a quella che si vuole costruire. In questo caso si può pensare di prendere la classe esistente e di estenderla per adattarla alle nuove necessità.

L'**ereditarietà** è un concetto fondamentale della programmazione ad oggetti e rappresenta la possibilità di creare nuove classi a partire da una classe già esistente (**classe base**). La nuova classe eredita tutti gli attributi e i metodi della classe base e può essere arricchita con nuovi attributi e nuovi metodi. La classe così ottenuta si chiama **classe derivata**.

La derivazione di una classe si rappresenta con i diagrammi di classe attraverso i due rettangoli delle classi uniti da una freccia che parte dalla classe derivata verso la classe base:



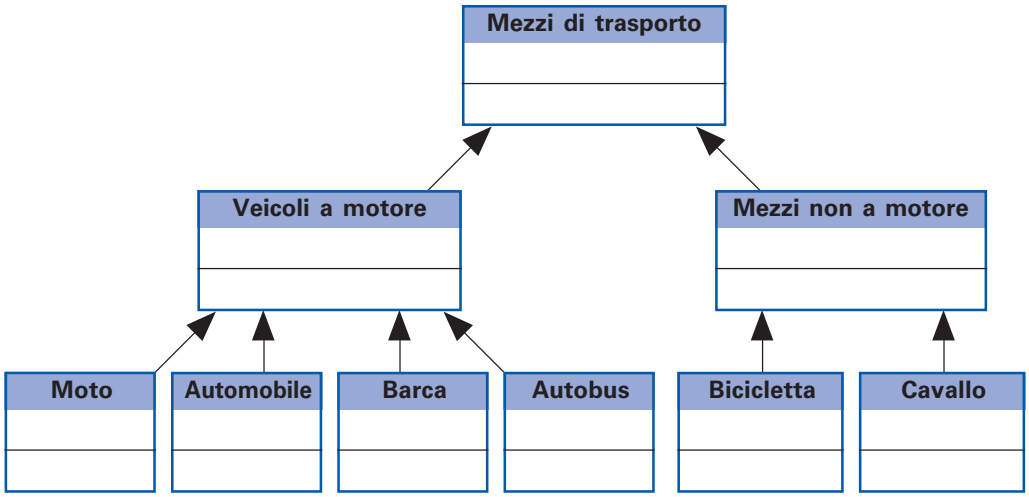
Il concetto di ereditarietà è familiare in biologia: è la capacità che hanno gli organismi di trasmettere i loro caratteri ai discendenti. In informatica, e particolarmente nella programmazione ad oggetti, gli organismi rappresentano le classi e i caratteri che vengono trasmessi sono gli attributi e i metodi.
In questo modo, quando una classe viene creata attraverso il meccanismo di ereditarietà a partire da un'altra classe, la classe derivata riceve in eredità tutti gli attributi e i metodi della classe generatrice. Gli attributi e i metodi privati non vengono ereditati dalle classi derivate.

La classe che è stata derivata da un'altra usando l'ereditarietà prende il nome di **sottoclasse**. La classe generatrice di una sottoclasse si chiama **superclasse** o *sopraclasse*.

Queste relazioni tra le classi individuano una **gerarchia di classi** che nasce da un processo di specializzazione. Infatti le classi che si trovano in cima alla gerarchia sono le più generali e man mano che si scende si trovano classi più specializzate.

La gerarchia delle classi si può descrivere graficamente usando il **grafo di gerarchia**: esso è costituito da un grafo orientato in cui i nodi sono rappresentati dalle classi, o meglio dai diagrammi delle classi, e gli archi individuano la presenza di una relazione di ereditarietà. La direzione della freccia è dalla sottoclasse verso la sopraclasse.

Nell'esempio della pagina seguente, la classe *Automobile* è sottoclasse della classe *Veicoli a motore* che a sua volta è sottoclasse di *Mezzi di trasporto*. In alto ci sono le classi più generiche che diventano più specializzate man mano che si scende lungo la gerarchia. Nel diagramma sono state indicate le classi solo attraverso il loro nome, tralasciando l'elenco degli attributi e dei metodi.

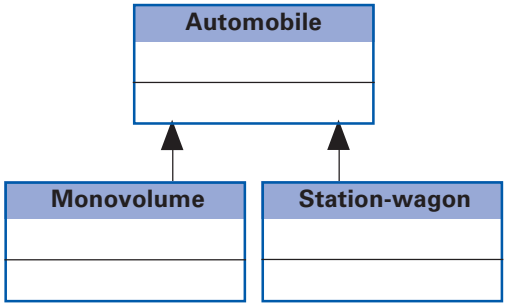


La sottoclasse eredita dalla sopraclasse tutti gli attributi e tutti i metodi, evitando di ripetere la descrizione degli elementi comuni nelle sottoclassi. L'ereditarietà consente di condividere ciò che è simile tra le classi con la possibilità di inserire le differenze. La nuova classe si differenzia dalla sopraclasse in due modi:

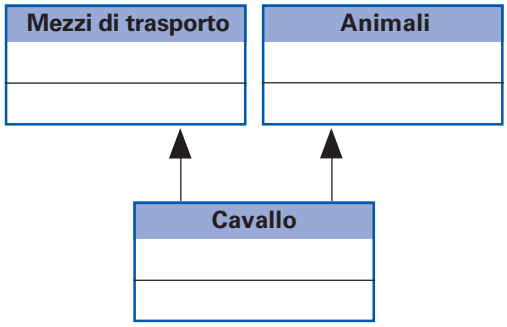
- **per estensione**, quando la sottoclasse aggiunge nuovi attributi e metodi che si sommano a quelli ereditati;
- **per ridefinizione**, quando la sottoclasse ridefinisce i metodi ereditati. In pratica viene data un'implementazione diversa di un metodo. Si crea un nuovo metodo che ha lo stesso nome del metodo ereditato da una sopraclasse, ma che ha una funzionalità diversa. Quando a un oggetto della sottoclasse arriva un messaggio che richiama un metodo ridefinito, viene eseguito il nuovo codice e non quello che era stato ereditato.
  Una sottoclasse ridefinisce un metodo perché vuole associare un comportamento diverso oppure perché vuole utilizzare un algoritmo più efficiente rispetto a quello della sopraclasse. Quando una sottoclasse ridefinisce un metodo si dice che lo sovrascrive (**overriding** del metodo) utilizzando lo stesso nome usato dalla sopraclasse.

In sostanza, l'ereditarietà serve a definire nuove classi, derivate da altre, che ereditano gli attributi e i metodi, con la possibilità di ridefinirli o di aggiungerne di nuovi.

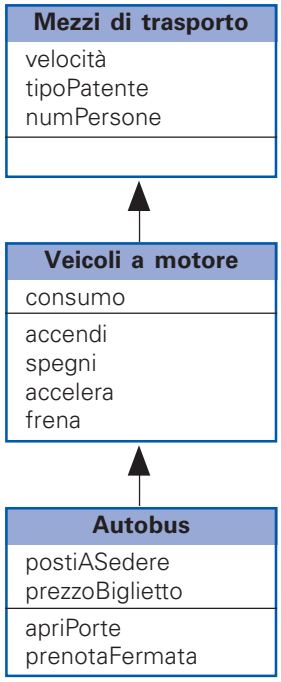
Esistono due tipi di ereditarietà: *singola* e *multipla*.
L'**ereditarietà singola** si verifica quando una sottoclasse deriva da un'unica sopraclasse. Usando i diagrammi delle classi, l'ereditarietà singola è indicata usando un'unica freccia uscente dalla sottoclasse e diretta verso la sopraclasse.
La sottoclasse ha una sola classe genitrice, ma questo non vieta alla sopraclasse di essere ereditata da più classi. Un esempio di ereditarietà singola è rappresentata dalla sottoclasse *Monovolume*. Possiamo pensare di definire un'altra sottoclasse, per esempio *Station-wagon*, derivata sempre dalla sopraclasse *Automobile*.



L'**ereditarietà multipla**, invece, comporta che una classe derivi da due o più sopraclassi. In questo modo si possono fondere insieme gli attributi e i metodi provenienti da classi diverse per crearne una nuova. Usando i diagrammi delle classi, l'ereditarietà multipla è indicata con due o più frecce che escono dalla sottoclasse e sono dirette verso le sopraclassi.
Per esempio si può considerare la classe *Cavallo*, derivante dalla classe *Mezzi di trasporto* e dalla classe *Animali*.



L'ereditarietà multipla è usata meno frequentemente rispetto all'ereditarietà singola: per questo motivo non tutti i linguaggi di programmazione la prevedono. Java, come vedremo nel prossimo paragrafo, utilizza solo l'ereditarietà singola.
L'esempio seguente spiega l'ereditarietà sviluppando una gerarchia composta da tre classi, complete di attributi e metodi.
Le classi considerate sono: *Mezzi di trasporto*, *Veicoli a motore* e *Autobus*. Si ipotizza che tutti gli attributi e i metodi siano pubblici. In questo modo possono essere ereditati dalle sottoclassi.



La classe *Mezzi di trasporto* è caratterizzata dagli attributi *velocità*, *tipoPatente*, *numPersone*.
La classe *Veicoli a motore* ha come attributi *consumo* e come metodi *accendi*, *spegni*, *accelera*, *frena*.
La classe *Autobus* ha gli attributi *postiASedere*, *prezzoBiglietto* e i metodi *apriPorte* e *prenotaFermata*.

La classe *Veicoli a motore* è sottoclasse di *Mezzi di trasporto* ed è sopraclasse di *Autobus*. Le sottoclassi sono definite per estensione delle sopraclassi: questo significa che la classe *Veicoli a motore* riceve in eredità i tre attributi della classe *Mezzi di trasporto*, la classe *Autobus* riceve in eredità tutti gli attributi e i metodi delle due sopraclassi *Veicoli a motore* e *Mezzi di trasporto*.

L'**ereditarietà** è uno strumento che consente il riutilizzo programmabile del software creato, cioè non è necessario partire da zero nella costruzione di un nuovo programma, basta utilizzare un insieme di classi già definite che hanno comportamenti simili a quelli del nuovo programma ed estenderle. L'estensione consente di inserire gli elementi aggiuntivi usati dalla nuova applicazione. Tutte le classi che sono già state scritte non vengono ricodificate, con il vantaggio di utilizzare classi già collaudate e con un buon livello di affidabilità, cioè prive di errori, perché sono state usate e controllate in altri programmi. Il riutilizzo, favorito dall'ereditarietà, consente un notevole risparmio di tempo nella produzione del software.
La programmazione orientata agli oggetti riserva molta importanza alle **librerie**. Tutti i linguaggi *Object-Oriented* (OO) forniscono librerie di classi molto vaste che possono essere sfruttate usando il meccanismo dell'ereditarietà. Quindi oltre a imparare la sintassi di un linguaggio OO, è fondamentale conoscere la gerarchia e la struttura delle classi che compongono le sue librerie.

12 Dichiarazione e utilizzo di una sottoclasse

L'**ereditarietà** è lo strumento che consente di creare le sottoclassi a partire da classi già definite.
In Java è presente solo un'**ereditarietà singola**, cioè ogni classe può avere al più una sopraclasse.
Una sottoclasse si dichiara aggiungendo nell'intestazione il nome della sopraclasse dopo la parola chiave **extends** nel seguente modo:

```
class NomeSottoClasse extends NomeSopraClasse
{
  . . .
}
```

Dopo la parola chiave *extends* può comparire solo il nome di una classe. Per il resto la sottoclasse mantiene la struttura di una classe normale.
La sottoclasse eredita tutti gli attributi e i metodi definiti dalla sopraclasse ad esclusione di quelli che sono stati dichiarati privati. Questo significa che gli oggetti creati come istanza di una sottoclasse, possono accedere anche agli attributi e ai metodi che vengono ereditati dalla sopraclasse.

Supponiamo di avere definito le seguenti due classi:

```
class A
{
  public int numA;
}

class B extends A
{
  public int numB;
}
```

Un oggetto creato come istanza della classe *B* nel seguente modo:

```
B oggB = new B();
```

può accedere sia all'attributo *numB* che all'attributo *numA*, ereditato dalla sopraclasse A, con le seguenti istruzioni:

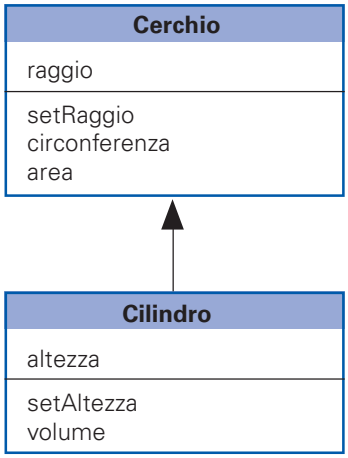
```
oggB.numB = 10;
oggB.numA = 5;
```

Quanto detto vale anche per i metodi ereditati.
La creazione di sottoclassi offre il vantaggio di poter riutilizzare una classe già esistente, sfruttando gli attributi e i metodi già definiti.
I membri di una classe dichiarati come *private* non sono ereditati dalle sottoclassi.
Solo gli attributi e i metodi che sono dichiarati *protected* o *public* vengono ereditati.
Anche i costruttori e gli elementi *static* non vengono ereditati.

PROGETTO 8

Descrivere la classe per il programma che calcola il volume di un cilindro utilizzando la classe Cerchio già esistente.

Con riferimento alla classe *Cerchio* sviluppata negli esempi precedenti, si vuole creare una classe per il cilindro come estensione della classe *Cerchio*. Il cilindro eredita gli attributi e i metodi del cerchio e in aggiunta definisce gli elementi propri. Questi ultimi includono l'attributo *altezza* e i metodi *setAltezza* e *volume*. Le due classi sono descritte graficamente con il seguente grafo di gerarchia.



Il costruttore della classe *Cilindro* riceve come parametri il raggio del cerchio di base del cilindro e l'altezza e viene definito nel seguente modo:

```
public Cilindro(double raggio, double altezza)
{
  super(raggio);
  this.altezza = altezza;
}
```

La parola chiave **super** è un oggetto speciale che fa riferimento alla sopraclasse della classe in cui viene utilizzata, in questo caso si riferisce alla classe *Cerchio*. In particolare, l'uso di *super* all'interno del costruttore serve per richiamare il costruttore della sopraclasse *Cerchio* e, tra parentesi, viene indicato il parametro *raggio*.

Un oggetto dispone quindi di due riferimenti speciali:
- **this**, che è un riferimento implicito all'oggetto stesso;
- **super**, che è un riferimento implicito alla sopraclasse dell'oggetto.

Il metodo *volume*, per calcolare il valore di ritorno, usa il metodo *area* che viene ereditato dalla classe *Cerchio*.

Di seguito è riportato il programma completo in linguaggio Java, che mostra la codifica della classe derivata e dei suoi metodi, oltre ad alcuni esempi di utilizzo del nuovo oggetto.

IMPLEMENTAZIONE DELLA CLASSE (*Cilindro.java*)

```
class Cilindro extends Cerchio
{
  private double altezza;

  public Cilindro(double raggio, double altezza)
  {
    // richiama il costruttore della classe cerchio
    super(raggio);
    this.altezza = altezza;
  }

  public void setAltezza(double altezza)
  {
    this.altezza = altezza;
  }

  public double volume()
  {
    // usa il metodo ereditato
    double vol = area() * altezza;

    return vol;
  }
}
```

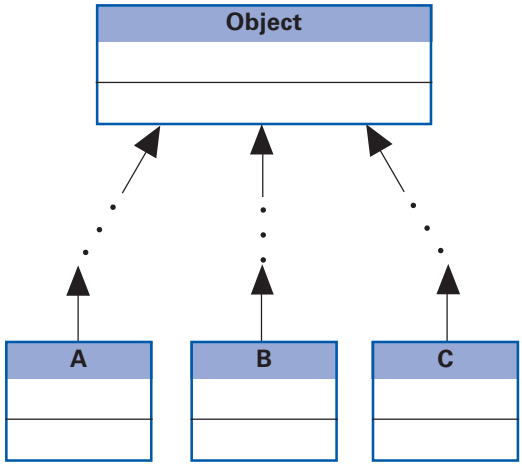
PROGRAMMA JAVA (*ProgCilindro.java*)

```
class ProgCilindro
{
  public static void main(String argv[])
  {
    // istanza dell'oggetto
    Cilindro cil = new Cilindro(4.0, 10.0);
```

```
    System.out.println("Dimensioni del cilindro");
    System.out.println("Area di base   = " + cil.area());
    System.out.println("Volume:        = " + cil.volume());

    // modifica il cilindro usando il metodo ereditato
    cil.setRaggio(7.6);
    cil.setAltezza(23.5);

    System.out.println("Dimensioni del nuovo cilindro");
    System.out.println("Area di base   = " + cil.area());
    System.out.println("Volume:        = " + cil.volume());
  }
}
```

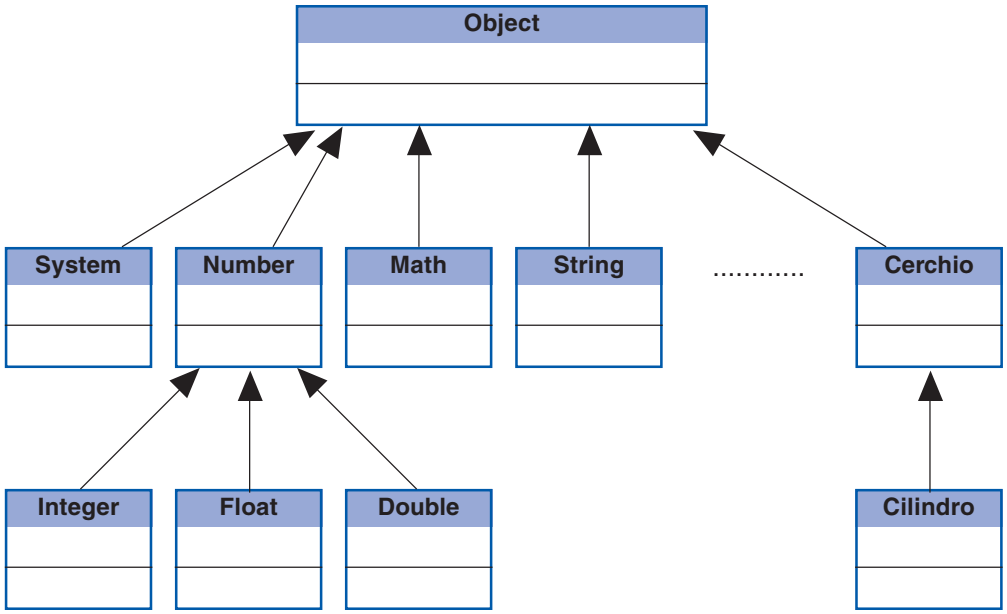
13 La gerarchia delle classi

In Java esiste una classe, chiamata **classe Object**, da cui vengono derivate tutte le altre classi e che è la sopraclasse di ogni altra classe. Volendo generalizzare, il grafo di gerarchia delle classi in Java si potrebbe rappresentare con la seguente figura, in cui la radice è la sopraclasse *Object*.



Tutte le classi che non sono dichiarate come estensione di altre classi, vengono automaticamente considerate sottoclassi della classe *Object*. Inoltre, anche una classe dichiarata come sottoclasse di un'altra, risalendo nella gerarchia, ha tra le sue sopraclassi la classe *Object*. La classe *Object* contiene solo i metodi generali che vengono ereditati da tutte le classi. Tra questi è presente un metodo che restituisce la classe a cui appartiene l'oggetto.

Gli ambienti di programmazione Java, quali *JDK*, non contengono solo il compilatore e l'interprete, ma anche un insieme di **librerie**. Queste sono composte, come vedremo in un successivo paragrafo, da tante classi. Alcune di queste sono già state usate nei programmi precedenti: per esempio le classi *String*, *System*, *Math*, *Integer*. Queste classi, come tutte quelle che vengono create, possono essere inserite in un unico grafo di gerarchia che le racchiude tutte: il grafo può essere visto come un albero, perché esiste una classe che può essere considerata la radice ed è la classe *Object*.



Le classi *Integer*, *Float* e *Double*, non vanno confuse con i tipi di dato predefiniti *int*, *float* e *double*. Rappresentano invece delle vere classi con propri attributi e propri metodi. Esse sono sottoclassi della classe *Number*.
Il grafo di gerarchia della figura è stato molto semplificato per renderlo comprensibile, in realtà il numero di classi è molto più grande e il grafo è più complesso. Tutte le classi che vengono create si innestano in questo grafo di gerarchia: per esempio la classe *Cerchio* è automaticamente considerata come sottoclasse della classe *Object*.

LE ULTIME CLASSI DELLA GERARCHIA

Se una classe viene dichiarata usando la parola chiave **final**, da questa non potranno essere generate le sottoclassi. L'uso di *final* serve per terminare la gerarchia di ereditarietà.

```
final class Ultima { … }
```

La classe *Ultima* non può più avere sottoclassi.
Se si inserisce dopo la parola chiave *extends* il nome di questa classe, viene segnalato un errore, cioè non è possibile dichiarare la seguente classe:

```
class Nuova extends Ultima { … }
```

Anche i metodi possono essere dichiarati usando la parola chiave *final* nell'intestazione e prima del nome del metodo.
Per esempio:

```
public final void aggiorna() { … }
```

Un metodo così dichiarato non può essere sovrascritto all'interno delle sottoclassi, cioè non possono esistere dei metodi nelle sottoclassi che hanno lo stesso nome. Tutti i metodi di una classe dichiarata *final* sono implicitamente *final*. Lo stesso vale per i metodi dichiarati con *private*.

14 Polimorfismo

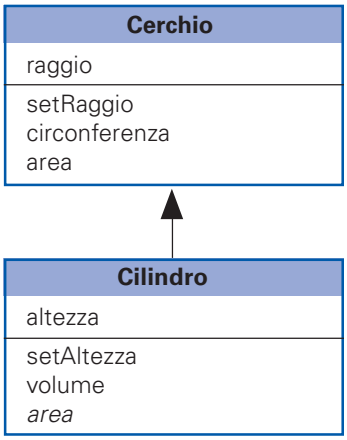
Abbiamo visto che, tramite l'ereditarietà, le sottoclassi ereditano gli attributi e i metodi delle sopraclassi e le estendono con l'aggiunta di nuovi attributi e nuovi metodi. Inoltre, nella programmazione ad oggetti, le sottoclassi possono ridefinire e sovrascrivere i metodi ereditati.

Il termine *polimorfismo*, in generale, indica la capacità di assumere più forme. Nella programmazione orientata agli oggetti è strettamente legato all'ereditarietà e alla ridefinizione dei metodi. Considerando una relazione di ereditarietà, le sottoclassi hanno la possibilità di ridefinire i metodi ereditati (mantenendo lo stesso nome) oppure lasciarli inalterati perché già soddisfacenti.

Il **polimorfismo** indica la possibilità per i metodi di assumere forme, cioè implementazioni, diverse all'interno della gerarchia delle classi.

Nei linguaggi ad oggetti sono presenti due tipi di polimorfismo:
- l'**overriding**, o *sovrapposizione* dei metodi,
- l'**overloading**, o *sovraccarico* dei metodi.

L'**overriding** di un metodo consiste nel ridefinire, nella classe derivata, un metodo ereditato, con lo scopo di modificarne il comportamento. Il nuovo metodo deve avere lo stesso nome e gli stessi parametri del metodo che viene sovrascritto.
Si consideri la classe *Cilindro* definita nel progetto del paragrafo precedente. Tra i metodi che essa eredita dalla classe *Cerchio* c'è il metodo *area*, che calcola l'area del cerchio alla base del cilindro. Questo metodo può essere sovrascritto per calcolare l'area della superficie totale del cilindro, formata dall'area dei due cerchi di base più l'area della superficie laterale.



Il codice del metodo sovrascritto è il seguente:

```
public double area()
{
  double supBase, supLaterale;

  supBase = super.area() * 2;
  supLaterale = circonferenza() * altezza;

  return supBase + supLaterale;
}
```

Per calcolare il valore della superficie di base viene richiamato il metodo *area* della classe *Cerchio* che calcola l'area del cerchio di base. Viene moltiplicato per 2 perché il cilindro ha due superfici circolari. La superficie laterale è calcolata richiamando il metodo *circonferenza* che è ereditato dalla classe *Cerchio*.
Dopo questa dichiarazione il metodo *area*, ereditato dalla classe *Cerchio*, viene sovrascritto con il nuovo metodo. Tutte le volte che un oggetto di classe *Cilindro* invoca l'esecuzione del metodo *area*, si attiva l'esecuzione del nuovo metodo. Il metodo che è stato sovrascritto può essere richiamato facendo riferimento all'oggetto speciale **super**, perché quest'ultimo contiene il riferimento alla sopraclasse. Il metodo *area* può essere richiamato sia dalle istanze della classe *Cerchio*, che dalle istanze della classe *Cilindro*. A seconda dell'istanza che richiede l'esecuzione, viene attivato un metodo piuttosto che l'altro.

L'**overloading** di un metodo è la possibilità di utilizzare lo stesso nome per compiere operazioni diverse. Solitamente si applica ai metodi della stessa classe che si presentano con lo stesso nome, ma con un numero o un tipo diverso di parametri.

All'interno della stessa classe, usando l'overloading, si possono dichiarare più metodi che hanno lo stesso nome ma possiedono parametri diversi. Il compilatore, comparando il numero e il tipo dei parametri, riesce ad individuare qual è il metodo corretto da invocare.

Un esempio di metodo che utilizza l'overloading è **println**, usato frequentemente negli esempi precedenti.
Questo metodo consente di stampare sul video i diversi tipi di parametri che vengono passati, tra cui stringhe, numeri interi e reali. Esistono quindi diversi metodi *println* con lo stesso nome, che vengono richiamati allo stesso modo, ma con parametri diversi, come mostrato dai seguenti esempi:

```
System.out.println("stringa di testo");
System.out.println(23);
System.out.println(5.896);
```

L'*overloading* fornisce un modo per far assumere allo stesso metodo dei comportamenti classificabili come **polimorfismo**: a seconda dei parametri che vengono passati, viene eseguito un comportamento diverso.
L'*overloading* è anche usato per offrire più di un costruttore a una classe.

Per esempio, la seguente classe *Numero* possiede due metodi costruttori.

```
class Numero
{
  private int num;

  public Numero()
  {
    num = 0;
  }

  public Numero(int num)
  {
    this.num = num;
  }
}
```

I due metodi sono stati definiti tramite *overloading*, infatti hanno lo stesso nome ma un numero diverso di parametri: un costruttore non ha parametri, mentre l'altro ha come parametro un numero intero.
La scelta di quale dei due metodi richiamare viene compiuta dal sistema in base alla corrispondenza dei parametri.
Per esempio, un oggetto di classe *Numero* può essere creato in due modi:

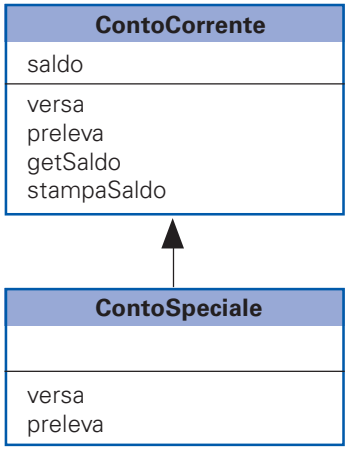
```
Numero n1 = new Numero();
Numero n2 = new Numero(100);
```

Nel primo caso l'oggetto *n1* viene creato usando il costruttore senza parametri e quindi l'attributo riceve il valore 0; nel secondo caso viene eseguito il costruttore che assegna all'attributo *num* il valore del parametro.

PROGETTO 9

Descrivere la classe per gestire un conto corrente che permette il versamento di assegni e il prelevamento di importi limitati, utilizzando la classe ContoCorrente già esistente.

Con riferimento alla classe presentata nel Progetto 3, a partire dalla classe base *ContoCorrente*, si deriva la classe *ContoSpeciale*.
Nella classe derivata *ContoSpeciale*, che eredita i metodi *versa*, *preleva*, *getSaldo* e *stampaSaldo* dalla classe *ContoCorrente*, ridefiniamo, usando il polimorfismo, i metodi *versa* e *preleva*.



Il metodo *versa* della classe base viene sottoposto a *overriding*, perché mantiene lo stesso nome e gli stessi parametri ma il suo comportamento viene sovrascritto con il codice inserito nella classe derivata. Il metodo sovrascritto esegue il prelevamento solo se l'importo non supera un limite costante e se c'è disponibilità nel saldo.

Il metodo *preleva* della classe base viene sottoposto a *overloading*, perché mantiene lo stesso nome ma cambia il tipo di parametro e il comportamento. Il nuovo metodo riceve come parametro una istanza della classe *Assegno* e ne versa l'importo.

L'attributo saldo della classe *ContoCorrente*, essendo dichiarato con la visibilità *private*, non viene ereditato nella classe derivata e la sua modifica può essere eseguita richiamando i metodi della classe base usando la parola chiave *super*.

Per esempio:

```
super.versa(p_assegno.importo);
```

Si noti che, sostituendo la precedente istruzione con il seguente assegnamento:

```
saldo += p_assegno.importo;
```

viene generato un errore di compilazione perché il *saldo* non è accessibile. Per rimuovere l'errore, si deve dichiarare l'attributo *saldo* con la visibilità **protected**. In questo modo, l'attributo viene ereditato e può essere modificato direttamente nella classe derivata.

Nel seguito sono riportate le dichiarazioni delle due classi e un programma che utilizza la classe derivata.

IMPLEMENTAZIONE DELLA CLASSE (*ContoCorrente.java*)

```
class ContoCorrente
{
  private double saldo;

  public void versa(double importo)
  {
    saldo += importo;
  }

  public void preleva(double importo)
  {
    saldo -= importo;
  }

  public double getSaldo()
  {
    return saldo;
  }

  public void stampaSaldo()
  {
    System.out.println("SALDO = " + saldo);
  }
}
```

IMPLEMENTAZIONE DELLA CLASSE (*ContoSpeciale.java*)

```
class ContoSpeciale extends ContoCorrente
{
  private final double LIMITE = 200.0;

  // overriding: metodo sovrascritto
  public void preleva(double importo)
  {
```

```
    // verifica il limite di prelevamento
    if (importo <= LIMITE)
    {
      // verifica la disponibilita' del saldo
      if (importo <= getSaldo())
      {
        super.preleva(importo);
      }
      else
      {
        System.out.println("Prelevamento non disponibile.");
      }
    }
    else
    {
      System.out.println("Prelevamento rifiutato.");
    }
  }

  // overloading: metodo sovraccaricato
  public void versa(Assegno p_assegno)
  {
    super.versa(p_assegno.importo);
  }
}
```

IMPLEMENTAZIONE DELLA CLASSE (*Assegno.java*)

```
class Assegno
{
  public double importo = 0;
}
```

PROGRAMMA JAVA (*ProgContoSpeciale.java*)

```
import java.io.*;

class ProgContoSpeciale
{
  public static void main(String argv[])
  {
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);

    // creazione degli oggetti
    ContoSpeciale conto = new ContoSpeciale();
    Assegno versamento = new Assegno();

    // dichiarazione delle variabili
    String valore = "";
    double importo = 0;
```

```
    System.out.print("Importo dell'assegno: ");
    try
    {
      valore = tastiera.readLine();
      versamento.importo = Double.valueOf(valore).doubleValue();
    }
    catch(IOException e) {}

    conto.versa(versamento);
    conto.stampaSaldo();

    System.out.print("Importo da prelevare: ");
    try
    {
      valore = tastiera.readLine();
      importo = Double.valueOf(valore).doubleValue();
    }
    catch(IOException e) {}

    conto.preleva(importo);
    conto.stampaSaldo();
  }
}
```

AUTOVERIFICA

Domande da 17 a 23 pag. 227-228
Problemi da 28 a 34 pag. 231

MATERIALI ON LINE

5. Le classi astratte
6. *Casting* tra le classi
7. Collegamento statico e collegamento dinamico

15 Le librerie

In Java, le **librerie** sono costituite da un insieme di classi già compilate che possono essere richiamate e usate all'interno dei programmi. Per riferirsi alle librerie, Java usa il termine **package**.
I package sono dei raggruppamenti che contengono un insieme di classi correlate. I programmi di grandi dimensioni che devono gestire decine o centinaia di classi, utilizzando i package, hanno la possibilità di organizzare le classi in unità logicamente coerenti.

Alcuni esempi di package disponibili nell'ambiente *JDK* sono:
- **java.applet**: raggruppa la classi per gestire le *applet*;
- **java.awt**: raggruppa le classi usate per creare le interfacce grafiche (pulsanti, finestre, menu) e per gestire le funzionalità grafiche (disegno, immagini, colori);
- **java.io**: raggruppa le classi per la gestione dei file e dell'I/O;
- **java.lang**, raggruppa le classi di base del linguaggio tra cui *System*, *Math* e *String*;
- **java.net**: raggruppa classi per lo sviluppo di applicazioni di rete;
- **java.util**: raggruppa classi di varia utilità per gestire la data, il calendario e i vettori.

I package di Java sono organizzati con una gerarchia nella quale i package sono contenuti all'interno del package chiamato **java**.

Le classi contenute nei package possono essere usate per creare oggetti oppure per creare nuove classi usando l'ereditarietà. Per riferirsi a una classe contenuta in un particolare package si usa la seguente notazione:

```
NomePackage.NomeClasse
```

Per esempio la classe *Button* contenuta nel package *java.awt* viene indicata con *java.awt.Button*. Per creare un oggetto dalla classe *Button*, si deve eseguire una dichiarazione e un'allocazione nel seguente modo:

```
java.awt.Button b;
b = new java.awt.Button();
```

L'oggetto *b* è un'istanza della classe *Button*: può quindi usare tutti gli attributi e i metodi pubblici della classe.

Per evitare di ripetere più volte il nome del package all'interno del codice, è possibile indicare una sola volta all'inizio quali sono le classi della libreria esterna che vengono utilizzate e importate nel programma. Per importare una classe bisogna usare il comando **import** nel seguente modo:

```
import java.awt.Button;
```

Dopo questa dichiarazione ci si può riferire alla classe *java.awt.Button* usando solo il nome *Button*. Grazie all'*import*, la creazione di un oggetto viene semplificata riducendo la lunghezza del nome nel seguente modo:

```
Button b;
b = new Button();
```

Se al nome del package si fa seguire un asterisco, si può fare riferimento a tutte le classi contenute nel package stesso. Per importare tutte le classi contenute nel package *java.io* si usa la seguente istruzione:

```
import java.io.*;
```

Questa istruzione è già stata usata in alcuni programmi, descritti nei paragrafi precedenti, per gestire l'input da tastiera.

Il package **java.lang** contiene la dichiarazione delle classi di base del linguaggio Java. Questo package viene automaticamente importato in tutti i programmi, quindi non è necessario usare il comando *import java.lang.**. Tra le classi contenute in questo package c'è *java.lang.System*, che è la classe utilizzata nei programmi precedenti per stampare i dati sul video.

L'insieme delle classi di Java formano la **API** (*Application Programming Interface*). Le classi contenute nell' API hanno una loro interfaccia costituita dagli attributi e dai metodi pubblici: per poter usare le classi incluse nelle librerie bisogna conoscere il loro nome, i loro attributi e i loro metodi.

La descrizione dell'API di Java si può trovare nella **documentazione** fornita insieme con il *JDK* oppure si può prelevare dal sito Internet di *Oracle* (*www.oracle.com/java*). La documentazione delle API è uno strumento molto importante per i programmatori in quanto fornisce informazioni utili per l'utilizzo delle classi di libreria.

Leggendo la documentazione è possibile conoscere:

- l'elenco dei package,
- l'elenco delle classi contenute in ogni package,
- l'elenco degli attributi di ogni classe e una loro descrizione,
- l'elenco dei metodi di ogni classe con l'indicazione e la spiegazione del numero/tipo dei parametri e l'eventuale valore di ritorno.

Per esempio, **java.lang.Math** indica la classe *Math* contenuta nel package *java.lang* che viene automaticamente importato in ogni programma. Contiene i metodi per l'esecuzione delle diverse operazioni matematiche.

Per dare una descrizione sintetica di questa classe si può usare il diagramma delle classi, limitandoci ad elencare solo i metodi più usati.

Math
E PI
abs (-) sin (double) cos (double) tan (double) exp (double) pow (double, double) log (double) sqrt (double) round (double) random

abs (-) significa che il metodo *abs* vuole un parametro, che può essere *double*, *float*, *int* o *long*.

Gli unici attributi presenti sono due costanti:

- **E** indica il numero *e*, la base dei logaritmi naturali.
- **PI**, indica il valore di *pigreco*.

Nei metodi vengono indicati tra parentesi i parametri che devono essere passati.

Quasi tutti richiedono un parametro di tipo *double*.

Il metodo **abs**, usando l'*overloading*, può ricevere come parametro qualunque tipo di dato numerico del quale calcola il valore assoluto.

I metodi **sin**, **cos**, **tan** calcolano le rispettive funzioni trigonometriche.

Il metodo **exp** calcola la funzione e^x usando come esponente il parametro.

Il metodo **pow** calcola l'elevamento a potenza in cui la base è il primo parametro e l'esponente è il secondo parametro.

Il metodo **log** calcola il logaritmo.

Il metodo **sqrt** calcola la radice quadrata.

Nella documentazione della classe *Math*, oltre alla descrizione del significato di ogni metodo, viene riportata l'intestazione. Per esempio per il metodo *abs* si può trovare la seguente intestazione:

```
public static int abs(int a)
```

dove:
- *public* indica che il metodo è accessibile
- *static* specifica che il metodo è richiamabile usando il nome della classe
- *int* indica che il valore di ritorno è di tipo intero.

Tra parentesi sono indicati il numero e il tipo di parametri. In questo caso c'è un solo parametro di tipo *int*.

Tutti gli attributi e i metodi della classe *Math* sono dichiarati con la parola chiave *static*, per richiamarli si deve quindi usare la seguente notazione:

```
Math.attributo
Math.metodo(parametri)
```

Per esempio:

```
double radice = Math.sqrt(9.0);   // radice = 3.0
double s = Math.sin(Math.PI/2);   // s = 1.0
double e = Math.E;                // e = 2.718281828459045
int a = Math.abs(-120);           // a = 120
```

16 Le stringhe

Le stringhe non sono un tipo di dato predefinito di Java: sono definite usando una classe chiamata **String** che è inclusa nel package *java.lang*. Questa classe serve per creare gli oggetti che rappresentano array di caratteri. Dopo aver fissato la dimensione della stringa, essa non può più essere modificata a meno di creare un nuovo oggetto. La classe *String* contiene vari metodi che consentono di manipolare le stringhe: è possibile contare il numero di caratteri, leggere i singoli caratteri, creare sottostringhe, convertire tutta la stringa in lettere minuscole o maiuscole.

Per usare le stringhe in Java si deve creare un oggetto di classe *String*.
Per esempio, la stringa contenente il nome di una persona viene creata dichiarando una variabile *nome* e allocando l'oggetto con l'operatore *new*. Nel parametro del costruttore si deve indicare la stringa da assegnare all'oggetto.

```
String nome = new String("Elena");
```

Esiste un altro modo per allocare le stringhe: invece di usare l'operatore *new*, si indica, tra virgolette, la stringa che si vuole assegnare all'oggetto.

```
String nome = "Laura";
```

Un ulteriore modo per generare le stringhe è tramite l'utilizzo dei metodi presenti nella classe *String*, che restituiscono un oggetto di classe *String*.

Per esempio, una nuova stringa composta dai primi tre caratteri del nome si ottiene con:

```
String abbreviazione = s.substring(0,3);
```

L'**operatore di concatenazione** tra stringhe (+) crea automaticamente un oggetto di classe *String*.
Esso consente di concatenare anche numeri effettuando automaticamente la conversione in stringa.

```
int a = 10;
String s = "Il numero " + a + " e' pari.";
```

La classe *String* non possiede un attributo accessibile. Il suo diagramma di classe, in cui vengono evidenziati solo i metodi principali, è il seguente:

String
charAt (int) equals (String) length substring (int, int) toLowerCase toUpperCase

Analizziamo il significato di questi metodi applicandoli al seguente oggetto:

```
String s = "Internet e' la Rete delle reti";
```

Il metodo **length** restituisce un numero che corrisponde alla lunghezza della stringa.
Per esempio, *s.length()* restituisce il valore 29.

Il metodo **charAt** restituisce il carattere che si trova nella posizione indicata dal parametro. Il primo carattere della stringa si trova in posizione 0, mentre l'ultimo è in posizione *s.length() - 1.*
Per esempio, *s.charAt(2)* restituisce il valore 't'.

I metodi **toLowerCase** e **toUpperCase** convertono tutti i caratteri della stringa rispettivamente in minuscole e in maiuscole. Per esempio, *s.toUpperCase()* crea la nuova stringa "INTERNET E' LA RETE DELLE RETI".

Il metodo **substring** restituisce una sottostringa della stringa a cui viene applicato. Il primo parametro indica la posizione iniziale della sottostringa, mentre il secondo è la posizione finale. Per esempio, dichiarando *String s1 = s.substring(15,19)*, si ottiene l'oggetto *s1* contenente la stringa "Rete".

Il metodo **equals** controlla se la stringa che riceve come parametro è uguale a quella a cui viene applicato. Restituisce un valore booleano: *true* se le stringhe sono uguali e *false* se sono diverse. Per esempio, *s.equals("Internet")* restituisce *false*.
Si noti che l'uguaglianza tra oggetti di classe *String* può anche essere verificata utilizzando l'operatore ==.

PROGETTO 10

Analizzare una frase inserita da tastiera calcolando la frequenza delle vocali e degli spazi bianchi.

Il problema chiede di contare il numero di vocali e di spazi a partire da una stringa di testo inserita da tastiera. Le informazioni sul testo, le vocali, gli spazi e la logica di calcolo vengono incapsulati nella classe *GestoreFrasi*. L'attributo *testo* contiene la stringa da analizzare, gli attributi *vocali* e *spazi* servono per i conteggi, il metodo *analizza* riceve come input il testo da analizzare e calcola il numero di vocali e spazi. Sono previsti inoltre due metodi identificati dal prefisso *getFreq* per restituire la frequenza di vocali e consonanti. Infine è stato previsto un metodo privato *arrotonda* che viene utilizzato dai due metodi pubblici per restituire un valore arrotondato della frequenza. Il diagramma di classe è il seguente:

GestoreFrasi
testo vocali spazi
analizza getFreqVocali getFreqSpazi arrotonda

Il metodo *analizza* contiene un ciclo che itera su tutti i caratteri della stringa di testo. Il singolo carattere viene memorizzato in una variabile di tipo *char* con la seguente istruzione:

```
carattere = testo.charAt(i);
```

Successivamente viene utilizzata la struttura di selezione *switch* per verificare se il carattere è una vocale o uno spazio e per incrementare il corrispondente attributo.
Nel metodo *main* la stringa da analizzare viene letta e memorizzata nell'oggetto *frase*, che viene creata come istanza della classe *String* e inizializzata con una stringa vuota tramite la seguente istruzione:

```
String frase = "";
```

Il codice completo del programma è riportano nel seguito.

IMPLEMENTAZIONE DELLA CLASSE (*GestoreFrasi.java*)

```
class GestoreFrasi
{
  private String testo;
  public int vocali;
  public int spazi;

  // costruttore: inizializza gli attributi
  public GestoreFrasi()
  {
    testo = "";
    vocali = 0;
    spazi = 0;
  }
```

```
  // calcola il numero di vocali e di spazi
  public void analizza(String t)
  {
    char carattere;

    this.testo = t;

    testo.toLowerCase();
    for(int i=0; i<testo.length(); i++)
    {
      carattere = testo.charAt(i);
      switch (carattere)
      {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u': vocali++;
                  break;
        case ' ': spazi++;
                  break;
      }
    }
  }

  // calcola la frequenza delle vocali
  public double getFreqVocali()
  {
    double freqVocali;

    freqVocali = (double) vocali / testo.length();

    return arrotonda(freqVocali);
  }

  // calcola la frequenza degli spazi
  public double getFreqSpazi()
  {
    double freqSpazi;

    freqSpazi = (double) spazi / testo.length();

    return arrotonda(freqSpazi);
  }

  // arrotonda a due cifre decimali
  private double arrotonda(double valore)
  {
    return (double) Math.round(valore * 100) / 100;
  }
}
```

PROGRAMMA JAVA (*ProgAnalisi.java*)

```
import java.io.*;

class ProgAnalisi
{
  public static void main(String argv[])
  {
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);

    String frase = "";
    GestoreFrasi gestore = new GestoreFrasi();

    System.out.println("Inserisci il testo da analizzare:");
    try
    {
      frase = tastiera.readLine();
    }
    catch(IOException e) {}

    gestore.analizza(frase);

    System.out.println("\nNumero di caratteri = " + frase.length());
    System.out.println("\nVOCALI");
    System.out.println("Numero    = " + gestore.vocali);
    System.out.println("Frequenza = " + gestore.getFreqVocali());

    System.out.println("\nSPAZI");
    System.out.println("Numero    = " + gestore.spazi);
    System.out.println("Frequenza = " + gestore.getFreqSpazi());
  }
}
```

AUTOVERIFICA

Domande da 24 a 26 pag. 228
Problemi da 35 a 39 pag. 231

MATERIALI ON LINE

8. Documentazione delle librerie Java

DOMANDE

Programmazione ad oggetti

1 Qual è il significato di OOP?

a) Object Oriented Paradigm
b) Object Operational Programming
c) Object Oriented Programming
d) Object Operational Paradigm

2 Quali di queste affermazioni, riferite al concetto di *oggetto*, sono vere (V) e quali false (F)?

a) Le caratteristiche di un oggetto descrivono il suo stato V F
b) I comportamenti di un oggetto descrivono il suo stato V F
c) I comportamenti si riferiscono alle funzionalità dell'oggetto V F
d) Un oggetto si può descrivere graficamente con un diagramma degli oggetti V F
e) L'oggetto è alla base della programmazione strutturata V F

3 Come si può rappresentare la definizione di una classe?

a) Elencando solo i suoi attributi
b) Elencando solo i suoi metodi
c) Elencando sia gli attributi che i metodi
d) Facendo alcuni esempi

4 Come viene rappresentata graficamente una classe?

a) Con il diagramma degli oggetti
b) Con il diagramma delle classi
c) Con il grafo delle classi
d) Con il grafo di gerarchia

Dichiarazione delle classi e uso degli oggetti

5 Quali di queste affermazioni, riferite alle classi, sono vere (V) e quali false (F)?

a) In ogni classe ci deve essere un metodo chiamato *main* V F
b) La dichiarazione di una classe inizia con la parola chiave *class* V F
c) Ogni classe viene memorizzata in un file con estensione *.class* V F
d) Le classi contengono solo la dichiarazione degli attributi V F

6 A che cosa serve l'operatore *new*?

a) A creare una classe
b) A creare un attributo
c) A creare un metodo
d) A creare un oggetto

7 Quale tra le seguenti rappresenta la corretta dichiarazione di un attributo?

a) tipo livelloDiVisibilità nomeAttributo;
b) livelloDiVisibilità tipo nomeAttributo;
c) livelloDiVisibilità nomeAttributo tipo;
d) nomeAttributo livelloDiVisibilità tipo;

AUTOVERIFICA

8 Per ognuno dei seguenti elementi presenti nella dichiarazione di un metodo, indica se è obbligatorio oppure facoltativo.

	Obbligatorio	Facoltativo
Nome		
Blocco di istruzioni		
Tipo di valore di ritorno		
Elenco di parametri		
Livello di visibilità		

9 Qual è la caratteristica del metodo *a*, la cui intestazione è *public void a()*?

a) Non restituisce nessun valore
b) Non contiene istruzioni
c) Può essere visto solo all'interno della classe in cui viene dichiarato
d) È un metodo statico

10 Quale tra queste istruzioni viene usata dai metodi per restituire un valore?

a) void
b) return
c) this
d) System.exit(1)

11 In quale altro modo si possono chiamare le variabili dichiarate all'interno dei metodi?

a) Attributi
b) Variabili di istanza
c) Variabili locali
d) Variabili globali

12 Quali di queste affermazioni, riferite ai metodi costruttori, sono vere (V) e quali false (F)?

a) Vengono eseguiti automaticamente quando si crea un nuovo oggetto V F
b) Restituiscono un valore di tipo *int* V F
c) Solitamente contengono le istruzioni di inizializzazione V F
d) Hanno lo stesso nome della classe a cui appartengono V F
e) In ogni classe deve essere dichiarato un costruttore V F

13 Qual è il meccanismo che fa interagire tra loro gli oggetti?

a) Lo scambio di attributi
b) Lo scambio di messaggi
c) L'ereditarietà
d) L'incapsulamento

14 Quali di queste affermazioni, riferite ai riferimenti nulli, sono vere (V) e quali false (F)?

a) Il riferimento *null* rappresenta un oggetto dichiarato ma non allocato V F
b) Non è possibile assegnare esplicitamente il valore *null* V F
c) Accedendo a un attributo di un oggetto *null* si genera l'eccezione *NullPointerException* V F
d) Il riferimento *null* è un valore numerico intero V F

Mascheramento dell'informazione e interazione tra oggetti

15 Quali di queste affermazioni, riferite al concetto di *information hiding*, sono vere (V) e quali false (F)?
a) Viene realizzato in pratica dall'interfaccia dell'oggetto V F
b) Viene realizzato in pratica dalla sezione privata dell'oggetto V F
c) Corrisponde al concetto di incapsulamento V F
d) Maschera la struttura dell'oggetto V F
e) Maschera l'oggetto rendendolo inutilizzabile V F

16 Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.
a) Il metodo ..... è usato per leggere il valore di un attributo.
b) Il metodo ..... è usato per modificare il valore di un attributo.
c) L'elenco dei metodi pubblici indica l' ..... di una classe.
d) La sezione ..... comprende gli attributi e i metodi non accessibili dall'esterno.
interfaccia, istanza, get, protected, pubblica, set, privata, hiding, costruttore

Ereditarietà e polimorfismo

17 Qual è la corretta dichiarazione per specificare che la classe *A* è sottoclasse della classe *B*?
a) class A
b) class B extends A
c) class A extends B
d) class A extend B

18 Quali di queste affermazioni, riferite all'ereditarietà in Java, sono vere (V) e quali false (F)?
a) Ogni classe può avere una sola sopraclasse (vale l'ereditarietà singola) V F
b) La sottoclasse si dichiara con la parola chiave *extends* V F
c) Gli oggetti creati dalle sottoclassi possono accedere solo agli attributi delle sopraclassi e non ai metodi V F
d) Gli attributi *private* di una classe sono ereditabili V F
e) Il riferimento speciale *super* serve per riferirsi alla sopraclasse V F
f) La classe *Object* è la sopraclasse di tutte le classi Java V F

19 Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.
a) La parola chiave ..... viene utilizzata quando una variabile non fa riferimento ad alcun oggetto.
b) ..... è un riferimento implicito all'oggetto stesso.
c) ..... è un riferimento implicito alla sopraclasse dell'oggetto.
d) La ridefinizione di un metodo (ereditato) con lo scopo di modificarne il comportamento è indicata con il termine .....
e) La possibilità di utilizzare lo stesso nome per compiere operazioni diverse è indicato con il termine di .....
this, new, public, private, null, overloading, super, void, overriding, information hiding

20 Con quale strumento viene offerta la possibilità di creare nuove classi estendendo classi già esistenti?
a) Incapsulamento
b) Ereditarietà
c) Polimorfismo
d) Astrazione

AUTOVERIFICA

21 Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.
a) La classe derivata da un'altra usando l'ereditarietà si chiama .....
b) La classe generatrice di una sottoclasse si chiama .....
c) Le relazioni sottoclasse-sopraclasse individuano una ..... di classi.
d) La gerarchia delle classi si può descrivere graficamente usando il .....
ereditarietà, sottoclasse, grafo di gerarchia, sopraclasse, base, diagramma delle classi, gerarchia, classe

22 Quali sono i tipi di ereditarietà?
a) semplice e complessa
b) singola e doppia
c) semplice e doppia
d) singola e multipla

23 Quale tra queste definizioni corrisponde al termine *polimorfismo*?
a) La possibilità per i metodi di assumere forme diverse
b) La possibilità per un oggetto di avere diversi metodi
c) La possibilità di poter creare più oggetti diversi
d) La possibilità che una classe sia derivata da più sopraclassi

Le librerie

24 Quale delle seguenti classi non è contenuta nel package *java.lang*?
a) System
b) Math
c) String
d) BufferedReader

25 Associa ad ogni metodo della classe *Math* della colonna di sinistra la corrispondente funzione calcolata scegliendola nella colonna di destra.

a) abs	1) elevamento a potenza
b) log	2) logaritmo
c) pow	3) radice quadrata
d) sqrt	4) valore assoluto

26 Quali di queste affermazioni, riferite alle stringhe, sono vere (V) e quali false (F)?
a) Fanno parte dei tipi di dato predefiniti di Java V F
b) Sono gestite da un'apposita classe V F
c) La classe *String* contiene vari metodi per manipolare le stringhe V F
d) Ogni stringa ha una dimensione fissa V F

PROBLEMI

Programmazione ad oggetti

1 Descrivere le caratteristiche dei seguenti oggetti: hard disk, computer, stampante.

2 Descrivere una radiosveglia indicando le sue caratteristiche e i suoi comportamenti.

3 Descrivere due o più oggetti *Televisore* usando il diagramma degli oggetti: indicare le caratteristiche e i valori associati.

4 Dall'elenco che segue, individuare gli oggetti e associare loro gli attributi e i metodi corretti: raggio; poligono; calcolaCirconferenza; numero lati; calcolaPerimetro; calcolaArea; cerchio.

5 Aggiungere altri attributi e altri metodi agli oggetti raggruppati nel problema precedente.

6 Data la seguente classe, creare tre istanze (oggetti) rappresentandole con il diagramma degli oggetti.

Pompa di carburante
tipo carburante prezzo al litro capacità cisterna carburante rimasto
modifica prezzo eroga carburante

7 Descrivere la classe *Televisore* usando il diagramma delle classi.

8 Usando il diagramma delle classi, descrivere un forno a microonde. Indicare tutti i possibili attributi e metodi completando il seguente diagramma.

Microonde
marca … …
impostaPotenza chiudiSportello … …

9 Usando il diagramma di classe *Microonde* descrivere graficamente il diagramma degli oggetti *forno1* e *forno2* attribuendo opportuni valori agli attributi.

10 Descrivere il diagramma di classe per modellare l'oggetto *mazzo di carte* pensando alle operazioni che potrebbe compiere un eventuale giocatore.

11 Utilizzando il diagramma delle classi, descrivere le caratteristiche di una figura geometrica piana e le trasformazioni elementari che possono essere ad essa applicate (affinità, omotetie, similitudini).

12 Utilizzando il diagramma delle classi, descrivere un dispositivo capace di trattare segnali elettrici (alimentatore, amplificatore, modulatore, campionatore, ecc.). Rappresentare graficamente, con il diagramma degli oggetti, due istanze della classe attribuendo opportuni valori agli attributi.

Dichiarazione delle classi e uso degli oggetti

13 Descrivere la classe per il programma che calcola l'area e la circonferenza di un cerchio conoscendone il diametro.

14 Descrivere la classe per il programma che calcola l'area e il perimetro di un quadrato conoscendone il lato.

15 Descrivere la classe per il programma che calcola l'ipotenusa di un triangolo rettangolo, conoscendo il valore dei cateti.

16 Descrivere la classe che simula una calcolatrice: deve essere in grado di memorizzare due numeri ed eseguire le operazioni elementari.

17 Descrivere la classe Orologio che registra le ore e i minuti. Definire tre costruttori: il primo per inizializzare l'orologio a un'ora di default, il secondo per inizializzare solo le ore, il terzo per richiedere sia le ore che i minuti.

18 Costruire una classe Atleta per rappresentare le informazioni di un giocatore.

19 Definire la classe Email con gli attributi: destinatario, mittente, oggetto e testo. Implementare i metodi per inserire i precedenti attributi e per visualizzare il messaggio completo.

Mascheramento dell'informazione e interazione tra oggetti

20 Per ogni attributo della classe Orologio (creata nel Problema 17), scrivere un metodo per leggere il valore e un metodo per modificarlo.

21 Scrivere un programma che memorizza il prezzo del carburante rilevato in cinque diversi distributori e successivamente calcola il prezzo massimo, minimo e medio.

22 Scrivere un programma che memorizza il prezzo e la capacità (espressa in MB) di diversi supporti di memorizzazione (CD, DVD, chiavi USB). Successivamente si deve stabilire quale è il supporto che offre il minor costo per MB.

23 Un compact disc può contenere dieci canzoni caratterizzate da un nome e una durata espressa in secondi. Scrivere una classe che descriva il CD e offra le seguenti operazioni:
- modifica il titolo di una canzone
- modifica la durata di una canzone
- dato il nome di una canzone, restituisce la sua durata.

24 Con riferimento alla classe creata nel problema precedente, scrivere un programma che, attraverso un menu, gestisce un CD consentendo di
- inserire il nome e la durata delle canzoni
- modificare il nome o la durata delle canzoni
- data una canzone restituire la durata della stessa.

25 Inserire in un vettore dieci libri, specificando per ognuno il titolo e il numero di pagine. Alla fine dell'inserimento, stampare il titolo di tutti i libri che hanno meno di 100 pagine.

26 Scrivere un programma che consenta di manovrare un'automobile. Le due operazioni possibili sono: accelerare (A) e frenare (F). Il programma resta in attesa che l'utente inserisca un comando. Se inserisce il carattere A, l'automobile aumenta la velocità di 5 km/h. Se inserisce il carattere F l'automobile rallenta la velocità di 10 km/h. Dopo l'inserimento di ogni comando si deve mostrare la velocità attuale dell'automobile. Se si superano i 90 km/h deve essere segnalato un avvertimento: "Vai troppo forte. Rallenta". La velocità non può essere inferiore a zero. Inizialmente l'automobile sta viaggiando a 50 km/h.

27 Ampliare il programma precedente aggiungendo la possibilità di spegnere (S) e accendere (I) la macchina. Se la macchina è spenta non è possibile né accelerare né frenare.

Ereditarietà e polimorfismo

28 Descrivere la classe per il programma che calcola il volume di un parallelepipedo a base quadrata estendendo la classe di base Quadrato (creata nel Problema 14).

29 Costruire una classe per rappresentare uno studente, estendendo la classe base Anagrafica (presentata nel Progetto 2) con un nuovo attributo per la matricola. Ridefinire il metodo per la stampa dei dati, visualizzando anche l'informazione sulla matricola.

30 Costruire una classe per rappresentare un dipendente, estendendo la classe base Anagrafica, con gli attributi per lo stipendio e il livello (da 1 a 7). Prevedere un metodo per gestire l'incremento del livello: ad ogni scatto deve corrispondere un aumento del 10% dello stipendio.

31 Con riferimento alla classe base creata nel problema precedente, derivare un nuovo dipendente specializzato. Per questo dipendente, il metodo per gestire l'incremento di livello viene sovrascritto considerando che, al raggiungimento del quinto livello, viene assegnato un premio di 1000 euro.

32 Con riferimento alla classe base creata nel problema 16, derivare una calcolatrice per calcolare le seguenti funzioni: la somma dei quadrati dei due numeri, la differenza tra il quadrato del numero maggiore e il quadrato del minore.

33 Con riferimento alla classe base creata nel problema 15, derivare una nuova classe Triangolo Colorato con gli attributi per memorizzare il colore del bordo e il colore dello sfondo del triangolo. Definire un metodo con due parametri per impostare i colori del triangolo. Usando l'overloading, creare un altro metodo, con lo stesso nome del precedente, ma con un solo parametro per impostare il colore del bordo.

34 Scrivere un programma per calcolare il costo di una telefonata da telefono fisso, conoscendo la sua durata in secondi e sapendo che il costo è di 15 centesimi al minuto. Definire un telefono cellulare, come estensione del telefono fisso, in cui il calcolo del costo è sovrascritto considerando una tariffa di 12 centesimi per lo scatto alla risposta e 0,35 centesimi al secondo.

Le librerie

35 Scrivere un programma che elenca le potenze di 2 a partire da 2^0 fino a 2^{20}.

36 Scrivere un metodo *arrotonda* che riceve come parametri un numero reale *r* e un numero intero *i*. Restituisce un numero reale che rappresenta l'arrotondamento del numero *r* alle prime *i* cifre decimali. Per esempio arrotonda(5.6794856,3) deve restituire il numero 5.679

37 Confrontare due stringhe che hanno lo stesso contenuto, per esempio "aaa" e "aaa", usando l'operatore ==. Rifare il confronto usando il metodo *equals* contenuto nella classe *String*.

38 Dati dieci nomi di persona, contare e visualizzare solo quelli che iniziano con una vocale.

39 Scrivere un programma che conta le lettere e i numeri contenuti in una stringa.

Focus notes

OBJECT ORIENTED PROGRAMMING

An *object* is a software construct that encapsulates state and behavior.
A *class* defines the common attributes and behaviors shared by a type of object.
Attributes (or properties) are the characteristics of a class.
A *behavior* is an action taken by an object when it is a message or in response to a change of state: it's what an object does.
Method is a function that defines the behavior of an object. An object can expose an attribute by providing an internal variable or by returning the value through a method.
A class is used to create object instances.
Constructors are methods used to initialize objects during their instantiation.
Objects communicate with one another through *messages*. Messages cause an object to do something. The four main principles of object-oriented programming are: data abstraction, encapsulation, inheritance and polymorphism.
Data abstraction is the development of a class as a model of real objects.
Encapsulation allows the programmer to build self-contained pieces of software by taking a functionality and hiding its implementation from the outside world.
Inheritance allows the programmer to define a new class upon the definition of an existing class. The new class inherits all of the attributes and behaviors (methods) from the existing class.
Polymorphism: methods having identical name, but on objects of different classes, and representing different code that expresses several different behaviors.
C++ and Java are the most popular examples of object-oriented computer languages.

DECLARING CLASSES

A class declaration starts with the *class* keyword and describes a list of attributes and methods.
An attribute is identified by a name, a type and an access modifier that controls whether attributes are visible and how subclasses inherit them.
There are three main access modifiers:
- *public*: the attribute is visible outside its class,
- *private*: the attribute is visible only inside its class,
- *protected*: the attribute is visible inside its class and within all its subclasses.

A method is identified by a name, a return type, a list of parameters, an access modifier and a list of instructions enclosed between braces.
There is a special method, called *constructor*, which is declared with the name of the class and without return type. The main goal of a constructor is to initialize the value of attributes.
When a class derives from another class, the *extends* keyword is used to mark this relationship.

CREATING OBJECTS

An object is an *istance* of a class and can be created with two steps: declaration and instantiation.
The *declaration* of an object creates a link between the name of the object and the relevant class. Declaring an object is like declaring a variable because, in the latter, a link is created between the name of the variable and its type.
The *istantiation* of an object allocates memory and creates a reference. The *new* keyword is responsible for carrying out the allocation and for invoking the constructor method.

Focus notes

Glossary

behavior
The way an object acts and reacts, the way it changes its state and passes the message.

class
A prototype describing the structure (attributes and methods) of an object.

class diagram
A graphical representation of the structure of classes and the relationships between them.

concatenation
The operation of connecting two strings.

constructor
The first method invoked when an object is created.

information hiding
Hiding the structure of an object, as well as the implementation of its methods.

inheritance
A mechanism to derive a class (*subclass*) from another class (*superclass*).

instance
An object of a specific class.

interface
The outside view of a class while hiding its structure and its behavior.

library
A set of well-known classes that programmers can reuse.

object
A software element consisting of a set of states and a set of behaviors.

polymorphism
A mechanism for the creation of new methods through overloading and overriding.

subclass
A class that inherits public attributes and methods from its superclass.

superclass
The class from which another class inherits.

unified modeling language (UML)
A standard notation used during object-oriented analysis and design to model classes and superclasses.

Acronyms

API	Application Programming Interface
AWT	Abstract Window Toolkit
I/O	Input/Output
JDK	Java Development Kit
OO	Object-Oriented
OOP	Object-Oriented Programming
RPM	Revolutions per minute
UML	Unified Modeling Language

SCHEDA DI AUTOVALUTAZIONE

CONOSCENZE

- ☐ Oggetti
- ☐ Attributi e metodi
- ☐ Definizione delle classi
- ☐ Incapsulamento
- ☐ Ereditarietà
- ☐ Polimorfismo
- ☐ Linguaggi di programmazione orientati agli oggetti
- ☐ Dichiarazione delle classi con attributi e metodi
- ☐ Livelli di visibilità
- ☐ Creazione di oggetti
- ☐ Riferimenti nulli
- ☐ Attributi e metodi statici
- ☐ Information hiding
- ☐ Interfaccia degli oggetti
- ☐ Applicazione dell'ereditarietà e del polimorfismo agli oggetti
- ☐ Array di oggetti
- ☐ Sottoclasse
- ☐ Gerarchia delle classi
- ☐ Librerie del linguaggio Java
- ☐ Manipolazione di stringhe

ABILITÀ

- ☐ Descrivere i concetti di base della programmazione ad oggetti
- ☐ Individuare gli aspetti della metodologia orientata agli oggetti
- ☐ Rappresentare una classe usando il diagramma delle classi
- ☐ Descrivere la gerarchia delle classi con il grafo di gerarchia
- ☐ Dichiarare una classe con attributi e metodi
- ☐ Creare un oggetto
- ☐ Utilizzare i livelli di visibilità di attributi e metodi
- ☐ Applicare l'information hiding
- ☐ Creare sottoclassi applicando l'ereditarietà
- ☐ Applicare l'overriding e l'overloading ai metodi
- ☐ Utilizzare i metodi delle librerie del linguaggio Java
- ☐ Manipolare le stringhe

SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 539

5

parte seconda

Programmazione ad oggetti

Strutture di dati e file

OBIETTIVI DI APPRENDIMENTO

In questo capitolo comprenderai la differenza tra gestione statica e gestione dinamica della memoria e sarai in grado di individuare le soluzioni dei problemi basate sull'uso di liste di dati dinamiche.
Imparerai anche a distinguere i diversi tipi di file e le diverse modalità di accesso ai file.

Strutture di dati dinamiche
Array dinamici
Pila
Coda
Lista concatenata
Albero
I flussi di input/output
File strutturati
File di testo

1 Strutture di dati dinamiche

Le strutture di dati dinamiche sono un modo per organizzare i dati di cui a priori non è nota la dimensione. Il problema nel gestire questi dati sorge perché non esiste un numero prefissato indicante la loro quantità e durante l'esecuzione questo numero può aumentare e diminuire continuamente. Per poter trattare questo tipo di dati servono le strutture che vengono definite **strutture di dati dinamiche**.
Le strutture di dati dinamiche hanno in comune la possibilità di adattarsi al variare della dimensione dei dati da trattare e tra di loro si differenziano per il tipo di operazioni che vengono eseguite sui dati. Le operazioni tipiche di queste strutture si dividono in:

- **operazioni di modifica**: inserimento ed eliminazione di un elemento dalla struttura;
- **operazioni di interrogazione**: ricerca di un certo elemento nella struttura, prelevamento di un elemento con particolari caratteristiche (per esempio il primo elemento inserito), conteggio del numero di elementi inseriti nella struttura.

Le strutture di dati dinamiche possono essere pensate in termini di oggetti, in quanto sono composte da dati e da un insieme di operazioni che agiscono sui dati. Per implementare le strutture di dati dinamiche non possono essere usate strutture statiche come gli array, in quanto la loro dimensione è predeterminata e limita il numero massimo di elementi che possono essere trattati. Si potrebbe pensare di fissare una dimensione grande per l'array in modo da comprendere il caso peggiore, ma in questo modo si ottiene un inutile spreco di memoria. La realizzazione di strutture dinamiche comporta un'interazione continua con il gestore della memoria per allocare nuove porzioni di memoria quando la struttura di dati cresce e per deallocarle quando la struttura di dati si contrae.
Il paragrafo successivo introduce la classe *Vector*, contenuta nelle librerie standard di Java, che supporta le caratteristiche di dinamicità e che risulta molto utile per implementare le strutture di dati dinamiche. Questa classe implementa un array di oggetti dinamico nel senso che è basata su un array le cui dimensioni possono aumentare o diminuire in base ai dati che contiene. L'array si adatta dinamicamente, aumentando quando vengono aggiunti nuovi dati e diminuendo quando i dati vengono eliminati. L'operazione di adattamento è eseguita automaticamente: viene allocato nuovo spazio quando serve e deallocato quando non è più utilizzato.
La classe *Vector* mette a disposizione i metodi per aggiungere gli elementi in fondo all'array e per prelevare o eliminare gli elementi in una certa posizione. Questi metodi sono quelli utilizzati per implementare le operazioni tipiche delle strutture di dati dinamiche.
Utilizzeremo questa classe per implementare due strutture di dati dinamiche: la *pila* e la *coda*. Entrambe le strutture possono crescere dinamicamente inserendo gli elementi da un lato del vettore, mentre si differenziano per il modo con cui avviene il prelevamento dei dati. In una pila, l'elemento che viene prelevato è quello inserito più di recente, mentre, in una coda, l'elemento che viene prelevato è quello presente nella struttura da più tempo.
Oltre a queste strutture verrà illustrata la struttura di *lista concatenata* realizzabile tramite i riferimenti ad oggetti, che sono stati introdotti in Java per mascherare l'uso dei puntatori.

2 Array dinamici

I *vettori* sono una lista di elementi tutti dello stesso tipo. Per creare un vettore occorre indicare la sua dimensione che, una volta fissata, non può più essere modificata: questo rappresenta una grande limitazione, perché spesso non si conosce a priori il numero degli elementi che verranno inseriti nel vettore. L'uso di un array in queste situazioni costringe quindi a scegliere arbitrariamente una dimensione: stabilendo una dimensione troppo grande si rischia di occupare memoria che non verrà utilizzata, al contrario una dimensione troppo piccola può non essere sufficiente per tutti gli elementi.

Per risolvere questi problemi serve una struttura di dati più complessa che consenta di variare le dimensioni a seconda delle necessità, aumentando o diminuendo l'occupazione di memoria.

L'**array dinamico** è un vettore di oggetti senza una dimensione prefissata, può aumentare o diminuire in base alle necessità.

Java rende disponibile una struttura dati di questo tipo attraverso la classe **Vector**, inclusa nel package **java.util** che deve essere importata nel programma.
Gli elementi di un *Vector* sono accessibili attraverso un indice, come avviene per gli array. Il primo elemento del vettore ha indice 0.
Ogni vettore ha una sua capacità che è data dal numero di oggetti che può memorizzare. Quando il vettore si riempie e c'è bisogno di ulteriore spazio, la capacità del vettore viene incrementata automaticamente.

Per dichiarare un vettore si possono usare tre diversi costruttori:

```
Vector v = new Vector();
```

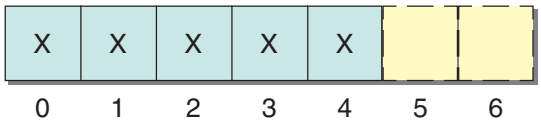
crea un vettore senza specificare la capacità;

```
Vector v = new Vector(5);
```

crea un vettore specificando la capacità iniziale attraverso il parametro;

```
Vector v = new Vector(5,2);
```

crea un vettore specificando la capacità iniziale con il primo parametro e il valore di incremento con il secondo parametro.
Questo significa che quando il vettore è pieno, viene aggiunto lo spazio per memorizzare altri due oggetti.

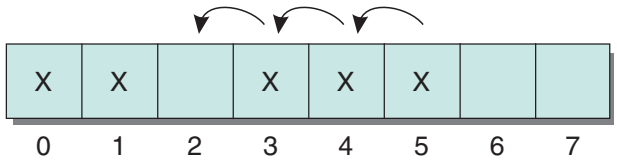


In figura è mostrato un vettore di dimensione 5. Con X si indica la presenza di un oggetto in quella posizione. Quando si deve inserire il sesto oggetto, la dimensione del vettore viene incrementata di 2 come specificato dal costruttore.
Allo stesso modo, se gli elementi sono più di sette, il vettore viene incrementato ancora di due posizioni. La dinamicità del vettore si manifesta anche in senso contrario: quando gli elementi vengono eliminati, le dimensioni del vettore vengono ridotte di conseguenza.
La classe *Vector* può essere utilizzata in tutte le situazioni in cui si usano gli array. Il vantaggio che si ottiene con un oggetto *Vector* è una migliore gestione della memoria.

I metodi principali della classe *Vector* che consentono di manipolare un vettore sono i seguenti:

- **addElement**(Object obj), riceve come parametro un oggetto di qualunque classe e lo aggiunge in fondo al vettore; all'interno del vettore gli oggetti subiscono un casting verso la classe *Object*.
- **removeElementAt**(**int** index), toglie dal vettore l'elemento nella posizione indicata dall'indice; la sua posizione non resta vuota perché tutti gli elementi con un indice più grande vengono fatti scalare per occupare la posizione e ottimizzare la memoria.

Questo significa che gli oggetti non hanno un indice fissato. L'indice con il quale vengono indicati può variare a causa delle eliminazioni di oggetti.



La figura mostra cosa succede dopo l'eliminazione dell'oggetto in posizione 2. L'oggetto che era in posizione 3 passa ad occupare la posizione 2, l'oggetto in 4 passa in 3 e quello in 5 passa in 4.

Se il parametro fa riferimento a un indice non corretto, cioè un indice negativo o maggiore della dimensione del vettore, viene generata un'eccezione.

- **size()**, restituisce un numero intero che indica il numero di oggetti effettivamente memorizzati nel vettore; è un numero minore o uguale alla capacità del vettore.
- **elementAt**(**int** index), restituisce un oggetto di classe *Object* che si trova nella posizione indicata dall'indice; per convertirlo in un oggetto della classe originaria va effettuato il casting. Il parametro deve essere un indice consentito, in caso contrario viene generata un'eccezione.

Vediamo come sono utilizzati in pratica i metodi elencati precedentemente.
Supponiamo di utilizzare un array dinamico per memorizzare un insieme di coordinate cartesiane che rappresentano i vertici di un poligono, di cui a priori non si conosce la cardinalità. L'array dinamico viene implementato usando l'oggetto *poligono* di classe *Vector* definito nel seguente modo:

```
Vector poligono = new Vector(1,1);
```

Il vettore così definito ha una capacità iniziale unitaria e un valore di incremento unitario. Un vettore dinamico con queste caratteristiche ha il vantaggio di occupare solo la memoria effettivamente utilizzata, ma non è efficiente se vengono continuamente aggiunti nuovi elementi. Il motivo è legato al valore di incremento: se il valore è piccolo saranno frequenti le operazioni per adeguare la memoria alle nuove dimensioni, se il valore è grande ci saranno meno operazioni a scapito di un'occupazione maggiore di memoria.
Le operazioni per adeguare la memoria richiedono tempo, quindi più sono le operazioni e più tempo sarà necessario.

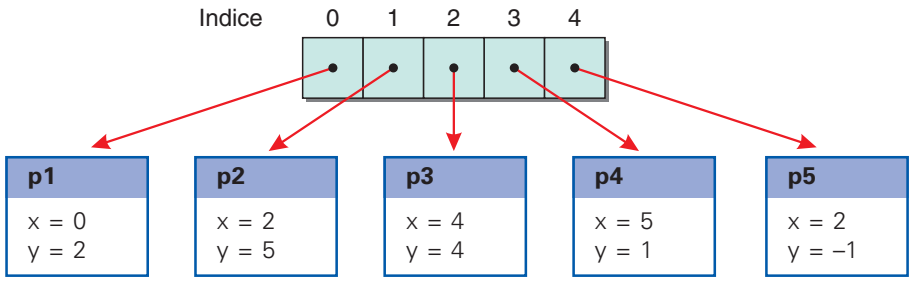
Al vettore precedentemente dichiarato vengono aggiunti gli oggetti generati come istanze della classe *Punto* composta dagli attributi pubblici *x* e *y* e definita nel seguente modo:

```
class Punto
{
  public int x, y;

  public Punto(int x, int y)
  {
    this.x = x;
    this.y = y;
  }
}
```

L'aggiunta di un oggetto al vettore *poligono* avviene creando prima l'oggetto da aggiungere e successivamente usando il metodo *addElement*. Il seguente metodo mostra come si possono aggiungere cinque coordinate create all'interno del metodo stesso:

```
public void aggiungi()
{
  Punto p1 = new Punto(0,2);
  Punto p2 = new Punto(2,5);
  Punto p3 = new Punto(4,4);
  Punto p4 = new Punto(5,1);
  Punto p5 = new Punto(2,-1);

  poligono.addElement(p1);
  poligono.addElement(p2);
  poligono.addElement(p3);
  poligono.addElement(p4);
  poligono.addElement(p5);
}
```

Dopo aver eseguito il metodo precedente, il vettore poligono si presenta composto da cinque elementi:



Quasi tutte le elaborazioni applicate ai singoli elementi del vettore hanno bisogno di scorrere il vettore dal primo all'ultimo elemento. Per eseguire questa operazione si utilizza un ciclo che parte dall'elemento in posizione 0 e scorre tutti gli elementi. La dimensione del vettore è ottenuta con il metodo *size*. Si utilizza una variabile temporanea, nel nostro caso *p*, che serve per memorizzare l'oggetto che viene recuperato dal vettore. L'oggetto restituito dal metodo *elementAt* è di classe *Object* e per questo motivo si deve applicare il casting verso la classe corretta.

Il ciclo per il vettore *poligono* è il seguente:

```
Punto p;
for(int i=0; i<poligono.size(); i++)
{
  p = (Punto) poligono.elementAt(i);

  // elaborazioni sull'elemento p
}
```

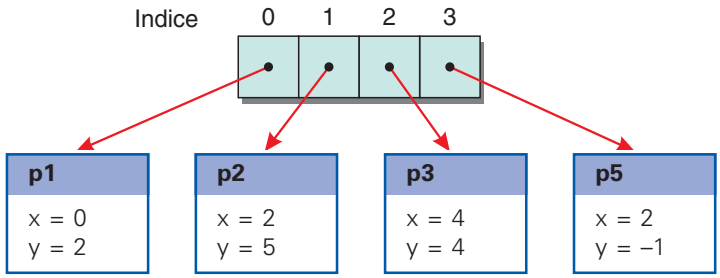
L'operazione che stampa tutti gli elementi del vettore sfrutta il ciclo appena presentato ed è implementata col seguente metodo:

```
public void stampa()
{
  Punto p;

  System.out.println("Elenco punti");
  for(int i=0; i<poligono.size(); i++)
  {
    p = (Punto) poligono.elementAt(i);
    System.out.println("x= "+p.x+" y= "+p.y);
  }
}
```

Un oggetto si elimina dal vettore *poligono* usando il metodo *removeElementAt*, specificando come parametro la posizione dell'oggetto da eliminare. Il seguente metodo mostra come si può eliminare un oggetto verificando preventivamente che l'indice sia nell'intervallo corretto:

```
public void togli(int i)
{
  if ((i >= 0) && (i < poligono.size()))
  {
    poligono.removeElementAt(i);
  }
}
```

Dopo aver eseguito il metodo *togli(3)*, il vettore *poligono* si presenta composto da quattro elementi e il punto *p5* si è spostato nella posizione del punto eliminato:



Il seguente esempio rappresenta un programma completo che sfrutta i vettori per gestire una rubrica telefonica.

PROGETTO 1

Costruire un programma per gestire una rubrica telefonica.

La rubrica che si vuole gestire contiene un numero variabile di elementi; ogni voce dalla rubrica memorizza un nome e un numero di telefono.
Il programma si presenta agli utenti con un menu per la gestione della rubrica, composto dalle seguenti opzioni: aggiunta di una voce, eliminazione di una voce, visualizzazione della rubrica.

La classe *Voce* racchiude le informazioni di un singolo nominativo, mentre la classe *Rubrica* contiene l'elenco dei nominativi in un vettore, dichiarato come attributo privato. Le operazioni di modifica della rubrica sono incapsulate nei metodi della classe *Rubrica* mentre la classe *Menu* viene utilizzata per controllare le scelte dell'utente.
Queste tre classi sono descritte nei seguenti diagrammi di classe:

Voce
nome telefono
stampa

Menu
mostraMenu scelta leggiIndice

Rubrica
elenco
aggiungiVoce eliminaVoce visualizza

Il programma viene avviato creando i due oggetti *miaRubrica* e *mioMenu* nel seguente modo:

```
Rubrica miaRubrica = new Rubrica();
Menu mioMenu = new Menu();
```

Successivamente, tramite un ciclo, viene letta la scelta dell'utente, usando i metodi di *mioMenu*, e vengono attivati i metodi di *miaRubrica* per eseguire le operazioni richieste. Per esempio, nel caso di aggiunta di una nuova voce nella rubrica:
- viene letto il valore inserito dall'utente con il metodo *scelta*:

```
scelta = mioMenu.scelta();
```

- viene creato l'oggetto *v*:

```
Voce v = new Voce();
```

- viene richiamato il metodo *aggiungiVoce*, passando la voce creata:

```
miaRubrica.aggiungiVoce(v);
```

Il programma e l'implementazione delle classi sono descritte di seguito.

IMPLEMENTAZIONE DELLA CLASSE (*Voce.java*)

```
import java.io.*;

class Voce
{
  private String nome = new String();
  private String telefono = new String();

  public Voce()
  {
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);

    System.out.print("Nome: ");
    try
    {
      nome = tastiera.readLine();
    }
    catch(IOException e) {}
```

```
    System.out.print("Telefono: ");
    try
    {
      telefono = tastiera.readLine();
    }
    catch(IOException e) {}
  }

  public void stampa()
  {
    System.out.println(nome + " tel.: " + telefono);
  }
}
```

IMPLEMENTAZIONE DELLA CLASSE (*Menu.java*)

```
import java.io.*;

class Menu
{
  private void mostraMenu()
  {
    System.out.println();
    System.out.println("1) Aggiungi voce");
    System.out.println("2) Elimina voce");
    System.out.println("3) Visualizza rubrica");
    System.out.println("4) Esci");
  }

  public int scelta()
  {
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);
    int scelta;

    mostraMenu();
    System.out.print("\n-> ");
    try
    {
      String numeroLetto = tastiera.readLine();
      scelta = Integer.valueOf(numeroLetto).intValue();
    }
    catch(Exception e)
    {
      scelta = 0;
    }

    return scelta;
  }
```

```
  public int leggiIndice()
  {
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);
    int indice;

    System.out.print("\nVoce da eliminare: ");
    try
    {
      String numeroLetto = tastiera.readLine();
      indice = Integer.valueOf(numeroLetto).intValue();
    }
    catch(Exception e)
    {
      indice = -1;
    }

    return indice;
  }
}
```

IMPLEMENTAZIONE DELLA CLASSE (*Rubrica.java*)

```
import java.util.Vector;

class Rubrica
{
  private Vector elenco;

  public Rubrica()
  {
    elenco = new Vector(1,1);
  }

  public void aggiungiVoce(Voce v)
  {
    elenco.addElement(v);
  }

  public void eliminaVoce(int indice)
  {
    try
    {
      elenco.removeElementAt(indice);
    }
    catch (Exception e)
    {
      System.out.println("Eliminazione non possibile.");
      return;
    }

    System.out.println("Eliminazione effettuata.");
  }
```

```
  public void visualizza()
  {
    Voce v;

    System.out.println("\nRUBRICA");
    for(int i=0; i<elenco.size(); i++)
    {
      System.out.print("posizione " + i + " -> ");
      v = (Voce) elenco.elementAt(i);
      v.stampa();
    }
  }
}
```

PROGRAMMA JAVA (*ProgRub.java*)

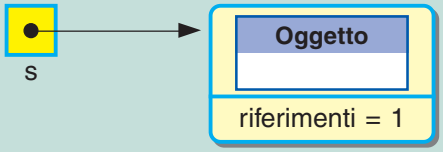
```
class ProgRub
{
  public static void main(String argv[])
  {
    Rubrica miaRubrica = new Rubrica();
    Menu mioMenu = new Menu();
    int scelta;

    scelta = mioMenu.scelta();

    while (scelta != 4)
    {
      if (scelta == 1)
      {
        Voce v = new Voce();
        miaRubrica.aggiungiVoce(v);
      }
      else if (scelta == 2)
      {
        int indice = mioMenu.leggiIndice();
        miaRubrica.eliminaVoce(indice);
      }
      else if (scelta == 3)
      {
        miaRubrica.visualizza();
      }

      scelta = mioMenu.scelta();
    }

    System.out.println("Fine programma.");
  }
}
```

GESTIONE AUTOMATICA DELLA MEMORIA

La gestione della memoria si occupa di tutti i problemi legati all'assegnazione della memoria per le variabili e gli oggetti. I linguaggi di programmazione gestiscono la memoria in modi diversi: ci sono linguaggi che lasciano questo compito al programmatore e altri che rendono il più possibile trasparente la gestione della memoria.
Tra i compiti di un gestore della memoria, oltre all'allocazione, vi è la *deallocazione*. Un buon gestore deve essere in grado di mantenere libera più memoria possibile per assegnarla quando ne verrà fatta richiesta.
Java mette a disposizione un **gestore automatico della memoria** che si occupa di tutti i compiti legati all'allocazione e deallocazione della memoria. Il programmatore interagisce con questo gestore solo quando usa l'operatore **new** per creare nuovi oggetti.
Quando un oggetto viene creato, viene allocata la memoria di cui ha bisogno. Questa memoria varia a seconda del numero di attributi di cui è composto. Mentre l'oggetto viene utilizzato, la memoria resta riservata. Quando un oggetto non è più usato, la memoria dovrebbe essere deallocata e resa disponibile per altri oggetti.
In Java la deallocazione è eseguita in modo automatico.
Una parte importante dell'interprete Java è il **Garbage Collector** (letteralmente, raccoglitore di spazzatura): è un sistema automatico che si preoccupa di recuperare le parti di memoria inutilizzate per metterle a disposizione del programma. Il *Garbage Collector* tiene traccia dello stato di tutti gli oggetti: quando si accorge che un certo oggetto non viene più usato, interviene per liberare la memoria occupata e renderla disponibile per usi futuri.
Questo modo di gestire la memoria presuppone che il *Garbage Collector* resti attivo durante tutta l'esecuzione di un'applicazione Java. È quindi un esempio di programma concorrente, nel senso che è eseguito contemporaneamente ad un altro programma.
Il vantaggio di avere un gestore di memoria automatico si traduce in una semplificazione dell'attività di programmazione e nell'eliminazione degli errori di programmazione dovuti alla gestione dei puntatori.
Il gestore della memoria può stabilire che un oggetto non è più utilizzato contando il numero di riferimenti a quell'oggetto.
Si consideri per esempio l'istruzione:

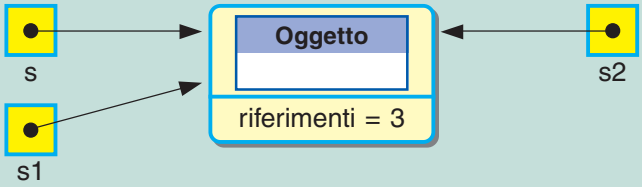
```
String s = new String();
```

Essa crea un riferimento a un oggetto di classe *String*. L'oggetto ha un solo riferimento contenuto in *s*.



Le seguenti istruzioni creano ulteriori riferimenti all'oggetto:

```
String s1 = s;
String s2 = s;
```



Un oggetto che possiede riferimenti può essere utilizzato e quindi deve restare allocato in memoria. Quando il numero di riferimenti a quell'oggetto diventa zero non c'è più modo per poterlo utilizzare: diventa un oggetto inaccessibile e può quindi essere trattato dal *Garbage Collector*.
La memoria occupata da tutti gli oggetti non più accessibili può essere liberata per altri riutilizzi.
Il *Garbage Collector* controlla costantemente gli oggetti per verificare se il numero dei riferimenti è diventato zero.
Gli oggetti non possono essere distrutti esplicitamente dal programmatore: la deallocazione viene fatta in modo automatico dal *Garbage Collector*.
Quello che può fare il programmatore è impostare tutti i riferimenti a un particolare oggetto con il valore *null*. In questo modo non distrugge l'oggetto ma lo candida a essere gestito dal *Garbage Collector*.
I metodi chiamati *costruttori* vengono eseguiti quando un nuovo oggetto viene creato. Esistono altri metodi, che vengono richiamati automaticamente prima che un oggetto venga eliminato.
Quando un oggetto non è più utilizzato e prima che il *Garbage Collector* provveda alla sua eliminazione, viene eseguito un metodo particolare, il **finalizzatore**. Questo metodo è utile per eseguire le azioni che evitino una brusca eliminazione dell'oggetto. Tra queste ci possono essere le operazioni di chiusura di eventuali file aperti dall'oggetto.
La dichiarazione di un finalizzatore avviene sovrascrivendo il metodo **finalize**, che è un metodo della classe *Object* e quindi viene ereditato da tutti gli oggetti. Il metodo deve essere dichiarato in questo modo:

```
protected void finalize()
{
  // operazioni di chiusura
}
```

AUTOVERIFICA

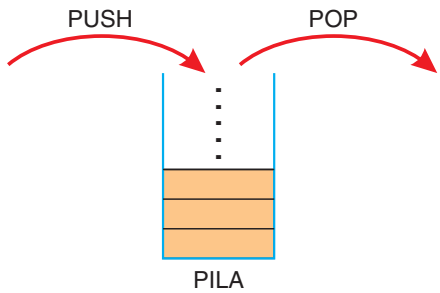
Domande da 1 a 4 pag. 287
Problemi da 1 a 5 pag. 289-290

MATERIALI ON LINE

1. Accesso alle strutture dinamiche usando gli iteratori
2. Struttura di dati dinamica per gestire le *Hash Table*

3 Pila

La **pila** (in inglese **stack**) è una struttura di dati dinamica gestita usando la modalità **LIFO** (*Last-In First-Out*). Ha la caratteristica che l'inserimento e l'estrazione dei dati avviene da un'unica estremità.



Gli elementi vengono aggiunti da un'estremità (**push**) e sono posizionati uno sopra l'altro, formando idealmente una pila di oggetti. Il prelevamento di un elemento (**pop**) avviene a partire da quello che si trova in cima alla pila ed è stato inserito per ultimo.

Questa struttura di dati viene usata quando si vuole tenere traccia delle informazioni per poi recuperarle procedendo in ordine inverso rispetto all'inserimento. Come esempio pratico di uso di una pila si può pensare alla funzionalità offerta da molti programmi applicativi che permette di annullare le ultime operazioni eseguite. In questo caso, tutte le operazioni eseguite dall'utente vengono memorizzate in una pila. Quando viene richiesto di annullare l'ultima operazione, il programma preleva l'informazione dalla pila e agisce di conseguenza. Il numero di operazioni che l'utente può annullare dipende dalla dimensione riservata alla pila.

La pila viene schematizzata con il seguente diagramma di classe:

Pila
array dinamico
push pop top vuota size

Le operazioni di modifica sono:

- *push*: inserisce un elemento in cima alla pila,
- *pop*: preleva un elemento eliminandolo dalla cima della pila.

Le operazioni di interrogazione sono:

- *top*: restituisce l'elemento in cima alla pila senza eliminarlo,
- *vuota*: segnala, con un valore booleano, se la pila non contiene elementi,
- *size*: restituisce il numero di elementi presenti nella pila.

Le soluzioni dei progetti seguenti per le strutture dati *pila* e *coda* sono basate, per una migliore comprensione, sull'uso della classe *Vector*. Nei *Materiali on line* si possono trovare le soluzioni con uso delle classi predefinite nella libreria *java.util*.

Il prossimo progetto mostra come implementare in Java una classe che offre le funzionalità della pila. Questa classe potrà essere riutilizzata tutte le volte che serve una struttura di dati dinamica con le caratteristiche della pila.

PROGETTO 2

Implementare la pila come struttura di dati generica adatta a trattare qualunque tipo di oggetto.

La classe *Pila* contiene un attributo privato in cui verranno memorizzati i dati. Questo attributo è un oggetto di classe *Vector* dichiarato nel seguente modo:

```
private Vector elementi;
```

Il numero degli elementi nella pila viene controllato con il metodo *size* e il valore viene memorizzato in una variabile intera chiamata *size*:

```
int size = elementi.size();
```

L'operazione *pop* legge l'elemento che si trova in posizione *size – 1*, lo cancella e lo restituisce al chiamante.

La cosa importante da sottolineare in questa implementazione della classe *Pila* è il tipo di dato associato al singolo elemento che può essere aggiunto. I singoli elementi sono gestiti usando la classe *Object*. Questo significa che qualunque oggetto può essere aggiunto alla pila, in quanto la classe *Object* è la sopraclasse di ogni altra classe.

Al momento di eseguire l'operazione *push*, viene automaticamente applicato il casting verso la classe *Object*. Quando viene prelevato un elemento, la pila lo restituisce come oggetto di classe *Object*: sarà compito di chi riceve l'oggetto convertirlo tramite un ulteriore casting verso la classe originaria.
Una pila realizzata in questo modo è una struttura di dati generica che può essere riutilizzata per trattare qualunque tipo di oggetto.

Il codice completo della classe *Pila* è riportato nel seguito.

IMPLEMENTAZIONE DELLA CLASSE (*Pila.java*)

```
import java.util.*;

class Pila
{
  private Vector elementi;

  public Pila()
  {
    elementi = new Vector();
  }

  public void push(Object obj)
  {
    elementi.addElement(obj);
  }

  public Object pop()
  {
    Object obj = null;
    int size = elementi.size();

    if (size > 0)
    {
      obj = elementi.elementAt(size-1);
      elementi.removeElementAt(size-1);
    }
    return obj;
  }
```

```
  public Object top()
  {
    Object obj = null;
    int size = elementi.size();

    if (size > 0)
    {
      obj = elementi.elementAt(size-1);
    }
    return obj;
  }

  public boolean vuota()
  {
    if (elementi.size() > 0)
    {
      return false;
    }
    else
    {
      return true;
    }
  }

  public int size()
  {
    return elementi.size();
  }
}
```

Vediamo ora come utilizzare la classe precedentemente definita all'interno di un semplice programma che mostra come vengono inseriti ed eliminati gli elementi da una pila.

PROGETTO 3

Popolare una pila con dieci numeri casuali e svuotarla mostrando che l'ordine degli elementi viene invertito, dall'ultimo al primo.

I numeri interi non possono essere inseriti direttamente nella pila perché quest'ultima accetta solo gli oggetti. Per superare questo problema si deve creare un'istanza della classe *Integer* che incorpora il valore di un numero intero all'interno di un oggetto.

```
numObj = new Integer(num);
```

A questo punto si può utilizzare questo oggetto per le operazioni con la pila. Nel momento in cui l'oggetto viene prelevato dalla pila, tramite l'operazione *pop*, bisogna effettuare il casting dell'oggetto verso la classe originaria che in questo caso è la classe *Integer*.

```
numObj = (Integer) pila.pop();
```

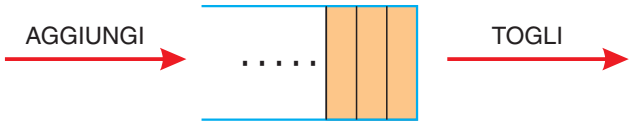
Il codice completo del programma è riportato di seguito.

PROGRAMMA JAVA (*ProgPila.java*)

```
class ProgPila
{
  public static void main(String argv[])
  {
    // crea una pila vuota
    Pila pila = new Pila();

    int num;
    Integer numObj;

    // aggiunge elementi alla pila
    for(int i=0; i<10; i++)
    {
      num = (int) (Math.random()*100);
      numObj = new Integer(num);
      System.out.print(numObj + " ");
      pila.push(numObj);
    }
    System.out.println("\nElementi nella pila: " + pila.size());

    // toglie gli elementi e li visualizza
    while (!pila.vuota())
    {
      numObj = (Integer) pila.pop();
      System.out.print(numObj + " ");
    }
    System.out.println();
  }
}
```

4 Coda

La **coda** (in inglese **queue**) è una struttura dinamica di dati gestita usando la modalità **FIFO** (*First-In First-Out*). Ha la caratteristica che l'inserimento avviene da un'estremità e l'estrazione dall'altra.



Gli elementi vengono aggiunti da un'estremità e sono posizionati uno di seguito all'altro, formando idealmente una coda di oggetti. Il prelevamento di un elemento avviene a partire da quello che è stato inserito per primo e si trova all'inizio della coda.
Questa struttura di dati viene usata quando si vuole tenere traccia delle informazioni per poi recuperarle nello stesso ordine nel quale sono state inserite.

Le code servono per gestire quelle situazioni in cui il primo che arriva è il primo ad essere servito. Una tipica situazione è quella che si verifica nei ristoranti in cui le ordinazioni vengono registrate e posizionate in coda in ordine temporale per essere soddisfatte una dopo l'altra a partire dalla prima ordinazione ricevuta.
La coda viene schematizzata con il seguente diagramma di classe:

Coda
array dinamico
aggiungi togli vuota size

Le operazioni di modifica sono:
- *aggiungi*: inserisce un elemento alla fine della coda,
- *togli*: preleva un elemento all'inizio della coda.

Le operazioni di interrogazione sono:
- *vuota*: segnala, con un valore booleano, se la coda non contiene elementi,
- *size*: restituisce il numero di elementi presenti nella coda.

Il seguente esempio mostra come implementare in Java una classe che offre le funzionalità della coda. Questa classe potrà essere riutilizzata tutte le volte che serve una struttura di dati dinamica con le caratteristiche della coda.

PROGETTO 4

Implementare la coda come struttura di dati generica adatta a trattare qualunque tipo di oggetto.

Come per la pila, anche la classe *Coda* contiene un attributo privato di classe *Vector* in cui verranno memorizzati i dati. I singoli elementi sono gestiti usando la classe *Object*.
Gli elementi aggiunti alla coda vengono posizionati nel vettore dopo l'ultimo elemento inserito.
Al contrario vengono tolti dalla coda sempre dalla posizione 0, con la seguente istruzione:

```
obj = elementi.elementAt(0);
```

Il codice completo della classe *Coda* è riportato di seguito.

IMPLEMENTAZIONE DELLA CLASSE (*Coda.java*)

```
import java.util.*;

class Coda
{
  private Vector elementi;

  public Coda()
  {
    elementi = new Vector();
  }
```

```
  public void aggiungi(Object obj)
  {
    elementi.addElement(obj);
  }

  public Object togli()
  {
    Object obj = null;
    int size = elementi.size();

    if (size > 0)
    {
      obj = elementi.elementAt(0);
      elementi.removeElementAt(0);
    }
    return obj;
  }

  public boolean vuota()
  {
    if (elementi.size() > 0)
    {
      return false;
    }
    else
    {
      return true;
    }
  }

  public int size()
  {
    return elementi.size();
  }
}
```

Il seguente esempio mostra come può essere inclusa la precedente classe in un programma Java che utilizza le code.

PROGETTO 5

Creare un programma che simula il funzionamento di un bar.

Un bar può essere immaginato in modo semplificato come un servizio in cui i clienti generano delle ordinazioni e i gestori soddisfano queste ordinazioni.
Ogni ordinazione è rappresentata dal numero del tavolo e dalla descrizione dell'ordine e può essere descritta con il seguente diagramma di classe.

Ordine
tavolo ordine
stampa

Le ordinazioni vengono organizzate in una coda, dichiarata come un oggetto di classe *Coda*, nel seguente modo:

```
Coda codaOrdinazioni = new Coda();
```

Un ordine viene aggiunto alla coda delle ordinazioni nel momento in cui viene generato dal cliente. Successivamente i gestori del bar prelevano le ordinazioni dalla coda e le soddisfano. Il seguente programma simula la sequenza di operazioni: ordina, ordina, ordina, soddisfa, ordina, soddisfa, soddisfa, soddisfa.

IMPLEMENTAZIONE DELLA CLASSE (*Ordine.java*)

```
class Ordine
{
  private String tavolo;
  private String ordine;

  public Ordine(String tav, String ord)
  {
    tavolo = tav;
    ordine = ord;
  }

  public void stampa()
  {
    System.out.println("Tavolo: " + tavolo);
    System.out.println("Ordine: " + ordine);
  }
}
```

PROGRAMMA JAVA (*ProgBar.java*)

```
class ProgBar
{
  public static void main(String argv[])
  {
    Coda codaOrdinazioni = new Coda();
    Ordine ord;

    ord = new Ordine("05", "4 caffe'");
    codaOrdinazioni.aggiungi(ord);
    System.out.println("--Aggiunto ordine");

    ord = new Ordine("03", "2 bibite");
    codaOrdinazioni.aggiungi(ord);
    System.out.println("--Aggiunto ordine");

    ord = new Ordine("11", "1 caffe' e 2 bibite");
    codaOrdinazioni.aggiungi(ord);
    System.out.println("--Aggiunto ordine");
```

```
    ord = (Ordine) codaOrdinazioni.togli();
    System.out.println("Soddisfatto ordine:");
    ord.stampa();

    ord = new Ordine("15", "1 cappuccino");
    codaOrdinazioni.aggiungi(ord);
    System.out.println("--Aggiunto ordine");

    ord = (Ordine) codaOrdinazioni.togli();
    System.out.println("Soddisfatto ordine:");
    ord.stampa();

    ord = (Ordine) codaOrdinazioni.togli();
    System.out.println("Soddisfatto ordine:");
    ord.stampa();

    ord = (Ordine) codaOrdinazioni.togli();
    System.out.println("Soddisfatto ordine:");
    ord.stampa();
  }
}
```

Nel precedente esempio viene usata un'unica variabile *ord* per creare e riferirsi a vari oggetti di classe *Ordine*. Il riutilizzo della stessa variabile non comporta l'eliminazione dell'oggetto precedentemente creato in quanto la variabile contiene solo il riferimento all'oggetto. Quando l'oggetto viene inserito nella coda, viene generato un duplicato di questo riferimento all'interno del vettore che gestisce la coda. Diventa così possibile riutilizzare la variabile *ord* per referenziare altri oggetti, senza che l'ordine precedente venga perso.

AUTOVERIFICA

Domande da 5 a 7 pag. 287-288
Problemi da 6 a 12 pag. 290

MATERIALI ON LINE

3. La classe *Stack* e l'interfaccia *Queue* nel package *java.util*

5 Lista concatenata

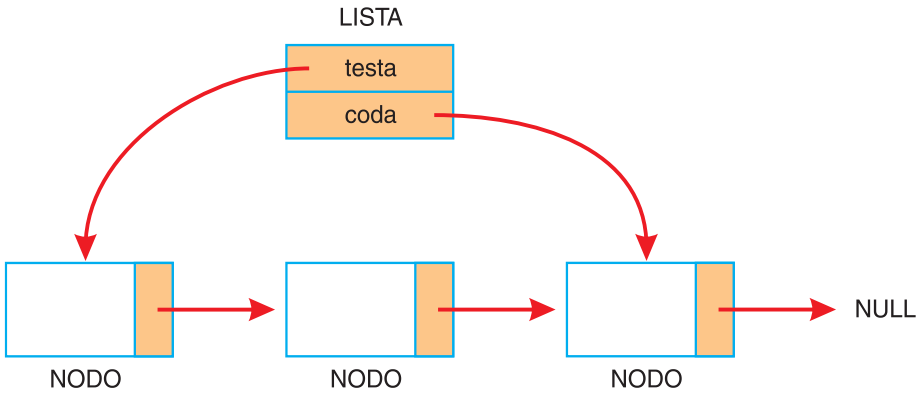
La **lista concatenata** (in inglese *linked list*) è una struttura di dati in cui gli elementi sono ordinati linearmente. Ogni elemento conosce qual è il suo successore e in questo modo, a partire dal primo elemento è possibile ricostruire tutti gli elementi presenti nella lista.

Oltre a gestire strutture ordinate, la lista ha il vantaggio di inserire e cancellare i dati in modo più veloce rispetto ad altre soluzioni. Con le liste concatenate, dopo che è stata trovata la posizione, l'inserimento è immediato. La stessa operazione, eseguita con un array, risulta molto più costosa in quanto, per fare spazio al nuovo elemento, si devono spostare tutti gli elementi di un posto.

Una lista può essere vista come una successione di nodi, in cui ogni nodo è costituito dai dati relativi all'elemento più un riferimento al nodo successivo. Per tenere traccia di questa successione è necessario conoscere il riferimento al primo elemento della lista (*testa*) e all'ultimo elemento (*coda*). Il riferimento alla testa è utile perché, partendo da questo, è possibile scorrere l'intera lista, eseguendo un'operazione chiamata di **attraversamento**. Il riferimento alla coda è utile per aggiungere rapidamente un elemento in fondo alla lista senza doverla scorrere per recuperare l'ultimo nodo.

In altri linguaggi di programmazione, le liste concatenate vengono gestite utilizzando i puntatori. In Java vengono invece utilizzati i **riferimenti agli oggetti** che in pratica svolgono lo stesso ruolo dei puntatori, ma la cui gestione è in buona parte demandata al modulo *run-time* di Java.

Graficamente, una lista concatenata può essere rappresentata nel seguente modo:



La lista concatenata viene schematizzata con il seguente diagramma di classe:

Lista concatenata
testa coda
inserisci elimina contiene

Le operazioni di modifica sono:
- *inserisci*: inserisce un elemento nella lista,
- *elimina*: preleva un elemento specifico dalla lista.

L'operazione di inserimento può essere a sua volta di vari tipi a seconda di come si vuole aggiungere l'elemento alla lista. Gli elementi potrebbero essere aggiunti in testa, in coda oppure all'interno della lista, se occorre rispettare un particolare ordine nella generazione della lista. Ogni operazione di inserimento deve essere implementata con un metodo diverso.

L'operazione di interrogazione è:
- *contiene*: controlla se nella lista è presente un particolare elemento.

Esistono anche altre operazioni minori di interrogazione, per esempio un'operazione che controlla se la lista è vuota e un'operazione che conta gli elementi nella lista.

Il seguente esempio mostra come viene costruita una lista concatenata i cui elementi contengono un dato di tipo stringa.

PROGETTO 6

Costruire una lista di parole organizzate in ordine alfabetico.

Il nodo della lista contiene due attributi:
- *dato*: è l'informazione che viene memorizzata nel nodo, in questo caso si tratta di una parola,
- *successivo*: è un riferimento al nodo successivo; l'ultimo nodo della lista assegna il valore *null* all'attributo successivo.

Nodo
dato successivo
setSuccessivo getSuccessivo getDato

Le operazioni *set* e *get* sono utilizzate per accedere e modificare gli attributi del nodo che sono dichiarati con il modificatore *private*.
La lista che si vuole realizzare consente le seguenti operazioni:
- inserimento di un nuovo elemento in ordine alfabetico,
- eliminazione di un elemento specificato,
- controllo circa la presenza di un elemento specificato nella lista,
- stampa della lista in ordine alfabetico.

Lista
testa coda
inserisci elimina contiene stampa

Gli attributi *testa* e *coda* sono oggetti di classe *Nodo* che vengono dichiarati e inizializzati nel seguente modo:

```
private Nodo testa = null;
private Nodo coda = null;
```

Il metodo *inserisci* viene implementato considerando quattro situazioni:
1. il nuovo elemento viene aggiunto in testa e la lista è vuota
2. il nuovo elemento viene aggiunto in testa e la lista contiene elementi
3. il nuovo elemento viene aggiunto all'interno della lista
4. il nuovo elemento viene aggiunto in coda alla lista.

La prima istruzione contenuta nel metodo *inserisci* è la creazione di un nuovo nodo, poi in base alla situazione si effettuano altre operazioni.
L'inserimento in ordine alfabetico viene gestito utilizzando il metodo **compareTo** contenuto nella classe *String*. Questo metodo esegue un confronto lessicografico tra due stringhe, la prima è quella da cui viene invocato il metodo, mentre la seconda stringa è indicata dal parametro.

Il metodo *compareTo* restituisce il valore zero se le stringhe sono uguali, restituisce un valore minore di zero se la prima stringa è minore della seconda in senso lessicografico, restituisce un valore maggiore di zero se la prima stringa è maggiore della seconda.

Il metodo *elimina* viene implementato considerando tre situazioni:
1. l'elemento da eliminare si trova in testa alla lista
2. l'elemento da eliminare si trova all'interno della lista
3. l'elemento da eliminare si trova in coda alla lista.

Nella situazione 1 viene modificato il riferimento dell'attributo *testa* facendolo puntare al secondo elemento della lista.
Nella situazione 2 viene modificato il riferimento dell'attributo *successivo* relativo al nodo che precede l'elemento da eliminare.
Nella situazione 3 si ripete l'operazione effettuata nella situazione 2 e in più si aggiorna il riferimento dell'attributo *coda* facendolo puntare al penultimo elemento della lista.

Il metodo *contiene* scorre la lista partendo dalla testa e arrestandosi quando trova l'elemento cercato oppure quando termina la lista.
Il metodo *stampa* scorre la lista dalla testa fino alla coda visualizzando tutti gli elementi.
Entrambi questi metodi si basano su un ciclo che possiede la seguente struttura:

```
Nodo temp = testa;
while (temp != null)
{
  // operazioni sull'elemento temp

  temp = temp.getSuccessivo();
}
```

Il programma completo per la gestione della lista concatenata viene riportato di seguito.

IMPLEMENTAZIONE DELLA CLASSE (*Nodo.java*)

```
class Nodo
{
  private String dato;
  private Nodo successivo;

  public Nodo(String dato)
  {
    this.dato = dato;
    successivo = null;
  }

  public void setSuccessivo(Nodo succ)
  {
    successivo = succ;
  }

  public Nodo getSuccessivo()
  {
    return successivo;
  }
```

```
  public String getDato()
  {
    return dato;
  }
}
```

IMPLEMENTAZIONE DELLA CLASSE (*Lista.java*)

```
class Lista
{
  private Nodo testa = null;
  private Nodo coda = null;

  // costruttore vuoto
  public Lista() {}

  public void inserisci(String str)
  {
    Nodo n = new Nodo(str);
    Nodo temp, succ;
    boolean aggiunto;

    if (testa == null)
    {
      testa = coda = n;
    }
    else
    {
      // controlla se va inserito in testa
      if (str.compareTo(testa.getDato()) < 0)
      {
        n.setSuccessivo(testa);
        testa = n;
      }
      else
      {
        // controlla all'interno della lista
        aggiunto = false;
        temp = testa;
        succ = testa.getSuccessivo();
        while (succ != null)
        {
          if (str.compareTo(succ.getDato()) < 0)
          {
            temp.setSuccessivo(n);
            n.setSuccessivo(succ);
            aggiunto = true;
            break;
          }
          temp = succ;
          succ = succ.getSuccessivo();
        }
```

```
        // controlla se va inserito in coda
        if (!aggiunto)
        {
          coda.setSuccessivo(n);
          coda = n;
        }
      }
    }
  }

  public void elimina(String str)
  {
    Nodo temp, succ;

    if (str.equals(testa.getDato()))
    {
      // elimina l'elemento in testa
      testa = testa.getSuccessivo();
    }
    else
    {
      temp = testa;
      succ = testa.getSuccessivo();
      while (succ != null)
      {
        if (str.equals(succ.getDato()))
        {
          temp.setSuccessivo(succ.getSuccessivo());
          break;
        }
        temp = succ;
        succ = succ.getSuccessivo();
      }

      // controlla se e' stata modificata la coda della lista
      if (temp.getSuccessivo() == null)
      {
        coda = temp;
      }
    }
  }

  public boolean contiene(String str)
  {
    Nodo temp;

    temp = testa;
    while (temp != null)
    {
      if (str.equals(temp.getDato()))
```

```
      {
        return true;
      }
      temp = temp.getSuccessivo();
    }
    return false;
  }

  public void stampa()
  {
    Nodo temp;

    System.out.println("Contenuto lista:");
    temp = testa;
    while (temp != null)
    {
      System.out.println(temp.getDato());
      temp = temp.getSuccessivo();
    }
  }
}
```

Il seguente esempio serve per testare il funzionamento delle struttura di dati dinamica precedentemente realizzata.

PROGETTO 7

Inserire nella lista cinque parole e stampare l'elenco in ordine alfabetico.

Avendo a disposizione la classe *Lista*, il programma risulta molto conciso e utilizza le operazioni *inserisci* e *stampa* implementate nella stessa classe.

PROGRAMMA JAVA *(ProgLista.java)*

```
class ProgLista
{
  public static void main(String argv[])
  {
    Lista l = new Lista();
    l.inserisci("rosso");
    l.inserisci("indaco");
    l.inserisci("nero");
    l.inserisci("azzurro");
    l.inserisci("verde");
    l.stampa();
  }
}
```

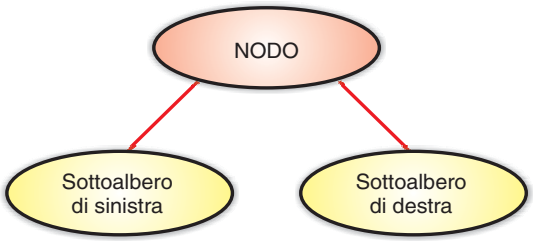
6 Albero

L'**albero** è una struttura di dati dinamica definita come un insieme di nodi connessi tra loro in modo che non esistano cicli. Questo significa che non ci sono percorsi chiusi che permettono, partendo da un nodo, di tornare allo stesso senza ripercorrere lo stesso tratto due volte.

Tra i nodi che compongono un albero si distinguono:
- la **radice**: è il nodo da cui partono le connessioni che generano l'albero,
- le **foglie**: sono i nodi esterni che non hanno nodi figli,
- i **nodi interni**: sono i nodi che non sono né radice né foglie.

Un esempio di albero è l'organizzazione dei file su disco gestita dal *file system* del sistema operativo. La radice è la directory principale, i nodi interni sono le altre directory e le foglie sono i file.
Un caso particolare di albero è costituito dall'albero binario, nel quale ogni nodo ha al massimo due nodi figli chiamati **sottoalbero sinistro** e **sottoalbero destro**. I due sottoalberi sono ancora alberi binari e quindi potranno essere vuoti oppure essere costituiti da nodi con due sottoalberi. La definizione di albero binario può essere espressa in modo sintetico con la ricorsione, cioè usando il concetto di albero binario per spiegare che cos'è l'albero stesso.

Un **albero binario** è un insieme vuoto oppure è costituito da un nodo a cui sono collegati al massimo due alberi binari.



L'albero binario viene schematizzato con il seguente diagramma di classe:

Albero
radice
inserisci elimina contiene

Le operazioni di modifica sono:
- *inserisci*: inserisce un elemento nell'albero,
- *elimina*: preleva un elemento specifico dall'albero.

L'operazione di interrogazione è:
- *contiene*: controlla se nell'albero è presente un particolare elemento.

Esistono anche altre operazioni di interrogazione, per esempio quella che stampa il contenuto dell'albero. Gli alberi binari sono strutture efficienti per la gestione delle liste su cui è definito un ordinamento. I dati vengono immessi nell'albero seguendo questo criterio: i dati che precedono il dato nel nodo o nel suo sottoalbero di sinistra, quelli che seguono il dato nel nodo, nel suo sottoalbero di destra.

Il seguente esempio mostra come usare un albero binario per gestire i dati che possono essere ordinati attraverso un codice numerico.

PROGETTO 8

Costruire il programma per gestire le operazioni di modifica e interrogazione su un albero binario.

Il nodo dell'albero contiene cinque attributi:
- *codice*, *descrizione*: rappresentano l'informazione associata al singolo nodo dell'albero,
- *eliminato*: è un attributo booleano che specifica se il nodo è stato eliminato,
- *sinistro*, *destro*: sono i riferimenti ai nodi che identificano il sottoalbero di sinistra e il sottoalbero di destra.

L'operazione *stampa* visualizza sullo schermo il codice e la descrizione, indicando anche se il nodo è stato eliminato.

I riferimenti al sottoalbero sinistro e destro sono dichiarati come oggetti pubblici di classe *Nodo* nel seguente modo:

```
public Nodo sinistro;
public Nodo destro;
```

Si noti come questa modalità di dichiarazione introduce una situazione ricorsiva, in cui gli attributi della classe *Nodo* fanno riferimento alla classe che stanno definendo.

Il diagramma della classe *Nodo* è mostrato nella figura:

Nodo
codice descrizione eliminato sinistro destro
stampa

Le operazioni previste consentono di aggiungere e togliere nodi, ricercare un nodo all'interno dell'albero e visualizzare i dati contenuti nella struttura.

Albero
radice
inserisci elimina contiene stampaInordine stampaPreordine stampaPostordine

L'albero ha un unico attributo *radice* che fa riferimento al nodo radice dell'albero e viene dichiarato con l'istruzione:

```
public Nodo radice;
```

Oltre alle classiche operazioni di inserimento (*inserisci*), eliminazione (*elimina*) e ricerca (*contiene*) sono presenti tre tipi di operazioni di stampa. Rappresentano i tre modi in cui può essere attraversato (o *visitato*) un albero:

- *stampaInordine*: l'attraversamento è eseguito in **ordine simmetrico** (prima il sottoalbero di sinistra, poi la radice, poi il sottoalbero di destra)
- *stampaPreordine*: l'attraversamento è eseguito in **ordine anticipato** (prima la radice, poi il sottoalbero di sinistra e il sottoalbero di destra)
- *stampaPostordine*: l'attraversamento è eseguito in **ordine posticipato** (prima il sottoalbero di sinistra e il sottoalbero di destra, poi la radice).

Se si vuole ottenere una stampa in ordine di codice si deve utilizzare l'operazione *stampaInordine*.

IMPLEMENTAZIONE DELLA CLASSE (*Nodo.java*)

```
class Nodo
{
  // informazioni del nodo
  public int codice;
  public String descrizione;
  public boolean eliminato;

  // sottoalberi
  public Nodo sinistro;
  public Nodo destro;

  public Nodo(int codice, String descrizione)
  {
    this.codice = codice;
    this.descrizione = descrizione;

    eliminato = false;
    sinistro = null;
    destro = null;
  }

  public void stampa()
  {
    System.out.print("-" + codice + "- " + descrizione);
    if (eliminato)
    {
      System.out.print(" (eliminato)");
    }
    System.out.println();
  }
}
```

PROGRAMMA JAVA (*Albero.java*)

```
class Albero
{
  public Nodo radice;

  public Albero()
  {
    radice = null;
  }

  public void inserisci(Nodo nuovo)
  {
    if (radice == null)
    {
      radice = nuovo;
    }
    else
    {
      inserisci(nuovo, radice);
    }
  }

  private void inserisci(Nodo nuovo, Nodo n)
  {
    if (nuovo.codice == n.codice)
    {
      n.eliminato = false;
    }
    else if (nuovo.codice < n.codice)
    {
      if (n.sinistro == null)
      {
        n.sinistro = nuovo;
      }
      else
      {
        inserisci(nuovo, n.sinistro);
      }
    }
    else
    {
      if (n.destro == null)
      {
        n.destro = nuovo;
      }
      else
      {
        inserisci(nuovo, n.destro);
      }
    }
  }
```

```
public void elimina(int elim)
{
  elimina(elim, radice);
}

private void elimina(int elim, Nodo n)
{
  if (n != null)
  {
    if (elim == n.codice)
    {
      n.eliminato = true;
      System.out.println("Nodo eliminato.");
    }
    else if (elim < n.codice)
    {
      elimina(elim, n.sinistro);
    }
    else
    {
      elimina(elim, n.destro);
    }
  }
  else
  {
    System.out.println("Nodo non trovato!");
  }
}

public Nodo contiene(int cerca)
{
  return contiene(cerca, radice);
}

private Nodo contiene(int cerca, Nodo n)
{
  if (n != null)
  {
    if (cerca == n.codice)
    {
      if (n.eliminato)
      {
        return null;
      }
      else
      {
        return n;
      }
    }
    else if (cerca < n.codice)
    {
      return contiene(cerca, n.sinistro);
    }
    else
```

```
      {
        return contiene(cerca, n.destro);
      }
    }
    else
    {
      return null;
    }
  }

  public void stampaInordine()
  {
    stampaInordine(radice);
  }

  private void stampaInordine(Nodo n)
  {
    if (n != null)
    {
      stampaInordine(n.sinistro);
      n.stampa();
      stampaInordine(n.destro);
    }
  }

  public void stampaPreordine()
  {
    stampaPreordine(radice);
  }

  private void stampaPreordine(Nodo n)
  {
    if (n != null)
    {
      n.stampa();
      stampaPreordine(n.sinistro);
      stampaPreordine(n.destro);
    }
  }

  public void stampaPostordine()
  {
    stampaPostordine(radice);
  }

  private void stampaPostordine(Nodo n)
  {
    if (n != null)
    {
      stampaPostordine(n.sinistro);
      stampaPostordine(n.destro);
      n.stampa();
    }
  }
}
```

Tutti i metodi della classe *Albero* sono presenti in doppia versione: una pubblica e una privata. In questa implementazione, i metodi privati sono **metodi ricorsivi**, infatti al loro interno è contenuta una chiamata al metodo stesso. Questi metodi vengono attivati dai metodi pubblici e alla prima chiamata ricevono come parametro il riferimento *radice*.
L'eliminazione di un nodo avviene impostando a *true* il valore dell'attributo *eliminato*. Il nodo non viene tolto dalla struttura e questo va tenuto in considerazione durante la stampa degli elementi dell'albero e durante l'inserimento di un nuovo nodo che ha lo stesso codice di quello eliminato.

La sezione esecutiva del programma contiene un menu di scelte per l'utente e viene implementata con la seguente classe.

PROGRAMMA JAVA (*ProgAlbero.java*)

```
import java.io.*;

class ProgAlbero
{
  InputStreamReader input = new InputStreamReader(System.in);
  BufferedReader tastiera = new BufferedReader(input);
  Albero a;

  public ProgAlbero()
  {
    int scelta;

    // crea l'albero vuoto
    a = new Albero();

    do
    {
      System.out.println();
      System.out.println("***** MENU *****");
      System.out.println("1) Aggiunta di un dato");
      System.out.println("2) Eliminazione di un dato");
      System.out.println("3) Ricerca di un dato");
      System.out.println("4) Stampa ordinata");
      System.out.println("5) Stampa in preordine");
      System.out.println("6) Stampa in postordine");
      System.out.println("\n0) Fine\n");
      System.out.print("Scelta: ");

      try
      {
        scelta = Integer.valueOf(tastiera.readLine()).intValue();
      }
      catch(Exception e)
      {
        scelta = 10;
      }
```

```
      switch(scelta)
      {
        case 0:
          break;
        case 1:
          aggiunta();
          break;
        case 2:
          eliminazione();
          break;
        case 3:
          ricerca();
          break;
        case 4:
          a.stampaInordine();
          break;
        case 5:
          a.stampaPreordine();
          break;
        case 6:
          a.stampaPostordine();
          break;
        default:
          System.out.println("Scelta non corretta.");
          break;
      }
    }
    while (scelta != 0);
  }

  public void aggiunta()
  {
    Nodo n;
    int codice;
    String descrizione;

    try
    {
      System.out.print("Codice: ");
      codice = Integer.valueOf(tastiera.readLine()).intValue();
      System.out.print("Descrizione: ");
      descrizione = tastiera.readLine();
      n = new Nodo(codice, descrizione);
      a.inserisci(n);
    }
    catch(Exception e) {}
  }

  public void eliminazione()
  {
    int codice;
```

```
    try
    {
      System.out.print("Codice: ");
      codice = Integer.valueOf(tastiera.readLine()).intValue();
      a.elimina(codice);
    }
    catch(Exception e) {}
  }

  public void ricerca()
  {
    Nodo n;
    int codice;

    try
    {
      System.out.print("Codice: ");
      codice = Integer.valueOf(tastiera.readLine()).intValue();
      n = a.contiene(codice);
    }
    catch(Exception e)
    {
      n = null;
    }

    if (n != null)
    {
      System.out.println("Nodo trovato:");
      n.stampa();
    }
    else
    {
      System.out.println("Codice inesistente.");
    }
  }

  // avvia il programma chiamando il costruttore
  public static void main(String argv[])
  {
    new ProgAlbero();
  }
}
```

Nei progetti precedenti, tutte le istruzioni del programma erano contenute nel metodo *main*. In questo esempio le istruzioni sono state spostate nel costruttore della classe *ProgAlbero*. Per attivare il programma, a partire dal metodo *main*, si crea soltanto un'istanza della classe usando l'operatore *new*.

AUTOVERIFICA

Domande da 8 a 10 pag. 288
Problemi da 13 a 20 pag. 290-291

MATERIALI ON LINE

4. Il *Java Collections Framework* per gestire gli insiemi di dati

7 I flussi di input/output

Gli archivi di dati sono rappresentati da un insieme di informazioni registrate su un supporto di memorizzazione secondaria, quale può essere un disco o un nastro. Le informazioni presenti nella memoria secondaria sono indicate con il termine generico di **file**.
Si hanno due principali vantaggi nel memorizzare i dati all'interno di file. Il primo è la persistenza dei dati, in quanto non vengono persi quando si chiude l'applicazione oppure si spegne il computer. Si possono utilizzare i file per memorizzare le informazioni che si vogliono mantenere e recuperare per varie esecuzioni del programma. Il secondo vantaggio è rappresentato dalla grande quantità di dati che può essere memorizzata in un file. Le memorie di massa hanno dimensioni che superano quelle della memoria principale e sono utili per gestire grandi quantità di informazioni.

Le operazioni fondamentali riguardanti il trattamento di un file sono:

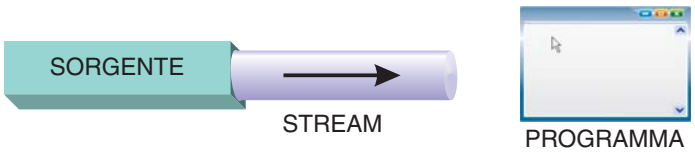
- **apertura** del file: si stabilisce un collegamento tra memoria centrale e l'unità di memoria di massa che contiene il file per svolgere su di esso delle operazioni;
- **lettura** (*read*) dal file: le operazioni che trasferiscono dati dal file alla memoria centrale;
- **scrittura** (*write*) sul file: le operazioni che trasferiscono dati dalla memoria centrale al file;
- **chiusura** del file: si chiude il collegamento tra memoria centrale e file.

La lettura si indica anche con *input* e la scrittura con *output*, e le operazioni che trasferiscono informazioni dalla memoria centrale al file e viceversa si chiamano, in generale, **operazioni di I/O** (*Input/Output*) sul file.

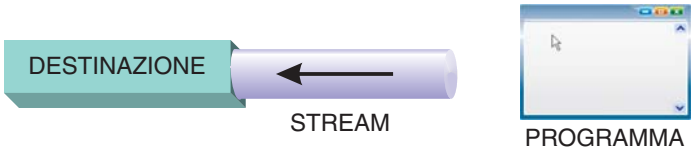
Le classi e i metodi che gestiscono l'I/O sono contenuti nel package *java.io*.

Java gestisce tutte le operazioni di I/O, e quindi anche i file, con il concetto di **stream**, cioè di flusso di dati. Gli stream costituiscono in pratica sequenze ordinate di dati e si dividono in:

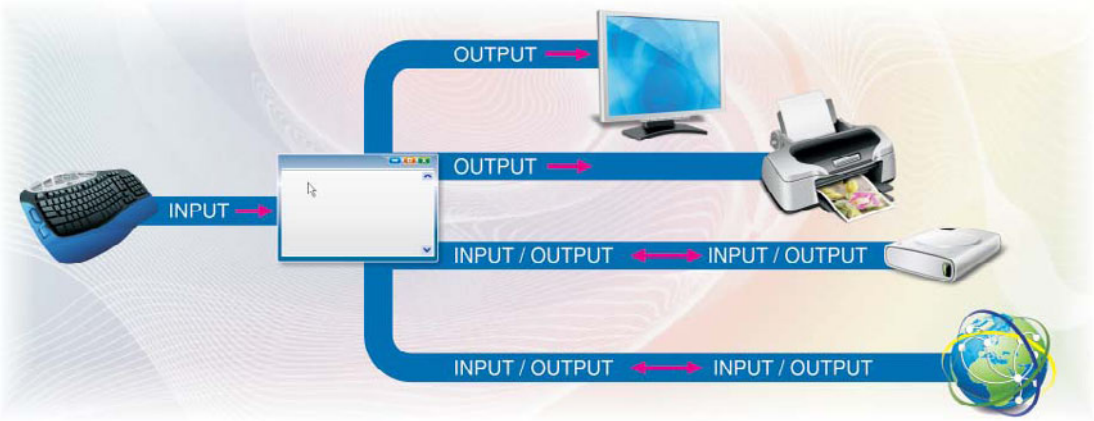
- *stream di input*, in cui i dati vengono presi da una sorgente e trasferiti al programma che li richiede;



- *stream di output*, in cui i dati vengono generati dal programma e diretti verso una destinazione.



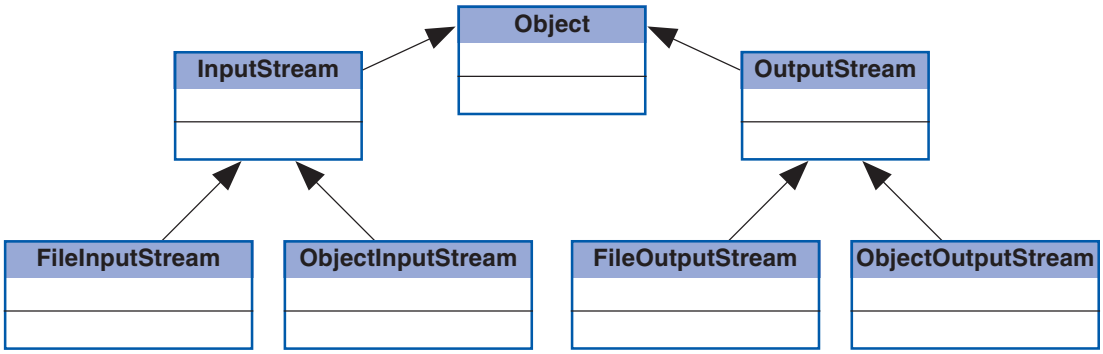
La sorgente degli *stream di input* può essere un file su disco, da cui si leggono dati, oppure un dispositivo di input come la tastiera da cui si ricevono i caratteri digitati dall'utente. La destinazione degli *stream di output* può essere un file su cui si scrive oppure un dispositivo di output come il video a cui vengono inviati i dati per essere visualizzati. Gli *stream di input* e *output* sono utilizzati anche per la ricezione e l'invio di informazioni in rete.



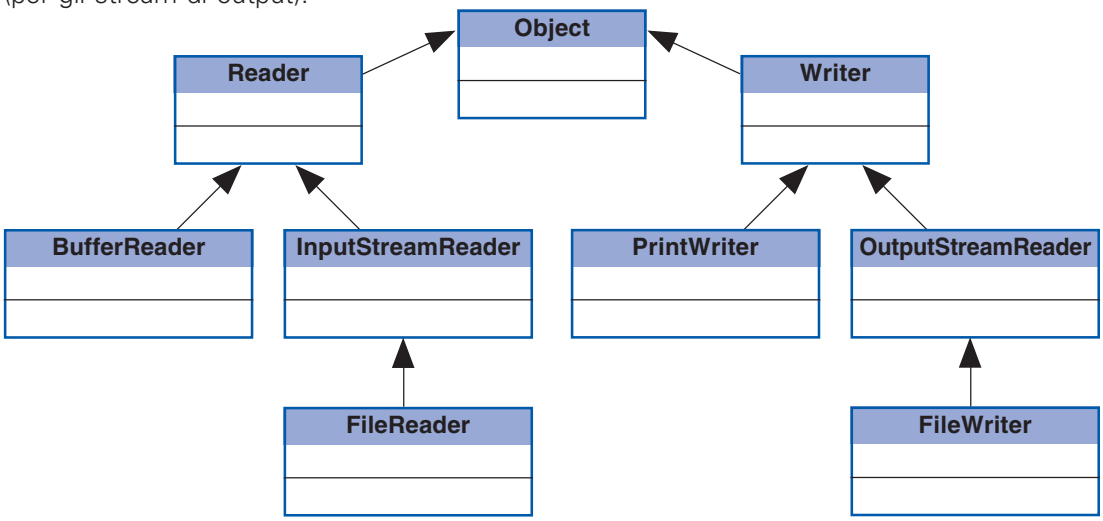
Gli *stream* vengono creati in Java istanziando le classi opportune contenute nel package *java.io*. È possibile classificare queste classi in due categorie principali:

- classi basate sui byte
- classi basate sui caratteri.

In Java questa differenza è ulteriormente marcata dal fatto che i byte sono formati da 8 bit mentre i caratteri sono gestiti con 16 bit.
Le classi basate sui byte hanno come sopraclasse **InputStream** (per gli stream di input) e **OutputStream** (per gli stream di output).



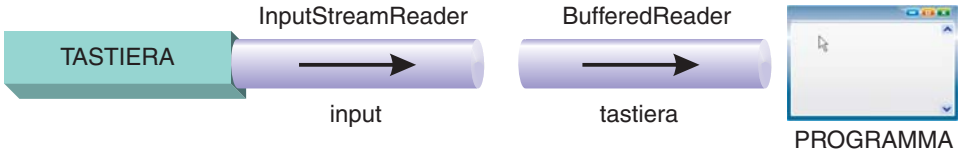
Le classi basate sui caratteri hanno come sopraclasse **Reader** (per gli stream di input) e **Writer** (per gli stream di output).



Una delle caratteristiche peculiari dei flussi di I/O in Java è la possibilità di concatenare tra loro più stream. Questo risulta utile perché ogni stream consente di osservare i dati in un certo modo, accedendo agli stessi attraverso dei metodi tipici del singolo stream.
Un esempio di concatenazione di stream è già stato visto con la lettura di dati da tastiera. In questo caso si leggono i caratteri, quindi si devono utilizzare le classi basate sui caratteri; inoltre, essendo un'operazione di input, si utilizzano gli stream di input. Le classi Java vanno scelte tra le sottoclassi della classe *Reader*. Le seguenti istruzioni mostrano come vengono impostati i due stream:

```
InputStreamReader input = new InputStreamReader(System.in);
BufferedReader tastiera = new BufferedReader(input);
```

La prima istruzione crea uno stream chiamato *input* dall'oggetto *System.in* che rappresenta lo standard input. La seconda istruzione crea un secondo stream chiamato *tastiera* che si collega al primo e permette di accedere ai dati utilizzando il metodo *readLine()*. Questo metodo è presente nella classe *BufferedReader*, ma non nella classe *InputStreamReader*.



I file trattati da Java possono essere classificati in due categorie:
- file strutturati
- file di testo.

I **file strutturati** sono composti da una struttura definita nel programma: per interpretare il contenuto di un file strutturato se ne deve conoscere la struttura. Per esempio un file composto da sequenze di due numeri interi, un numero reale e una stringa, può essere letto solo conoscendo questo ordine, altrimenti si ottengono dei risultati imprevisti.
Al contrario un **file di testo** è composto da sequenze di linee di caratteri, in cui ogni linea termina con un carattere speciale di fine riga. Il contenuto di questi file può essere letto anche usando un editor di testi.
I file strutturati utilizzano le classi basate sui byte per le operazioni di I/O, mentre i file di testo utilizzano le classi basate sui caratteri.

MATERIALI ON LINE

5. Lettura dei dati da tastiera con la classe *Console*

8 File strutturati

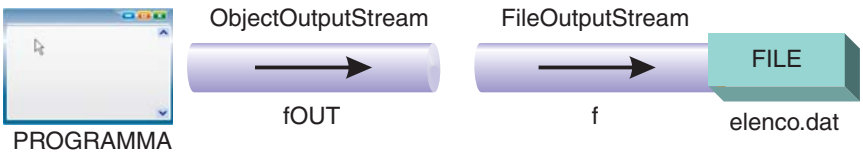
Esaminiamo il modo con cui Java tratta i file strutturati, spesso indicato con il termine di **serializzazione**.

• Apertura e chiusura
L'apertura di un file strutturato per le operazioni di output, cioè per la scrittura, viene eseguita con le dichiarazioni dei seguenti oggetti:

```
FileOutputStream f = new FileOutputStream("elenco.dat");
ObjectOutputStream fOUT = new ObjectOutputStream(f);
```

La prima riga crea uno stream *f* per scrivere sul file *elenco.dat*. Le operazioni di scrittura non vengono fatte direttamente su questo stream, ma sul secondo stream creato dalla classe **ObjectOutputStream**. È in questa classe che sono contenuti i metodi per la scrittura.

Rappresentando graficamente gli stream si ottiene la seguente figura:



Usando i precedenti comandi, se il file *elenco.dat* esiste viene sovrascritto e viene cancellato tutto il suo contenuto. Se si vuole aprire un file esistente per accodare dei valori si deve sostituire la creazione del *FileOutputStream* nel seguente modo:

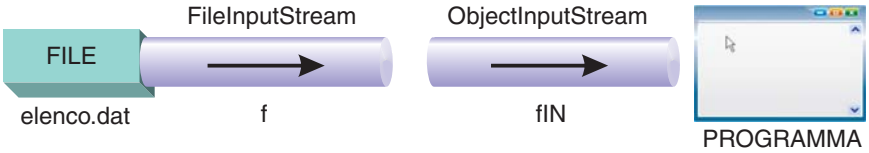
```
FileOutputStream f = new FileOutputStream("elenco.dat", true);
```

Il valore del secondo parametro indica la condizione di accodamento (*append*): se è *true* il file viene aperto per aggiungere i dati in coda a quelli preesistenti, altrimenti se non è presente o è *false*, il file viene aperto in scrittura e ciò comporta la cancellazione di un eventuale archivio preesistente.
L'apertura di un file strutturato per le operazioni di input, cioè per la lettura, viene eseguita con le dichiarazioni dei seguenti oggetti:

```
FileInputStream f = new FileInputStream("elenco.dat");
ObjectInputStream fIN = new ObjectInputStream(f);
```

La prima riga crea uno stream *f* per leggere dal file *elenco.dat*. Le operazioni di lettura non vengono fatte direttamente su questo stream, ma sul secondo stream creato dalla classe *ObjectInputStream*. È in questa classe che sono contenuti i metodi per la lettura dei dati memorizzati.

Rappresentando graficamente gli stream si ottiene la seguente figura:



La chiusura di uno stream, sia di input che di output, viene fatta richiamando il metodo *close* nel seguente modo:

```
f.close();
```

Le operazioni che gestiscono l'I/O possono generare diverse eccezioni, per esempio quando il metodo *close* riscontra una situazione anomala, oppure quando l'apertura di uno stream di input specifica un nome di file non esistente. Per questi motivi si devono racchiudere tutte le istruzioni che gestiscono i file in un blocco *try…catch* che cattura le eccezioni. Per controllare esattamente dove si verifica l'eccezione, si possono specificare tanti blocchi di controllo, uno per ogni istruzione, secondo la struttura a pagina seguente.

```
try
{
  // istruzione di I/O
}
catch(Exception e)
{
  System.out.println("Eccezione: " + e.getMessage());
}
```

• **Scrittura**
Dopo l'apertura dello stream di output, si possono utilizzare diversi metodi per la scrittura. Ogni metodo memorizza sul file un particolare tipo di dato e assume la forma **write*Dato***. Al posto di *Dato* si sostituisce il tipo di dato che si vuole memorizzare.
Per esempio:

```
fOUT.writeInt(25400);
fOUT.writeDouble(12.36);
```

Prima di chiudere il file è importante eseguire il metodo **flush**, che serve per scrivere su disco tutti i dati che sono attualmente contenuti nel buffer dello stream. Il metodo di scrittura più importante è **writeObject**. Con questo metodo è possibile salvare su di un file anche gli oggetti: è Java che si preoccupa di salvare la struttura dell'oggetto, memorizzando tutti i suoi attributi non statici. Un oggetto salvato su disco diventa **persistente**, cioè sopravvive alla singola esecuzione del programma e può essere richiamato in una successiva esecuzione. Una classe, le cui istanze si vogliono rendere persistenti, deve implementare l'interfaccia **Serializable**.

PROGETTO 9

Memorizzare sul file *elenco.dat* le informazioni di 3 persone.

Le informazioni che si vogliono memorizzare sul file sono contenute negli attributi *nome* e *eta* dalla classe *Persona*, descritta nel seguente diagramma di classe.

Persona
nome eta

La dichiarazione della classe *Persona* mette in evidenza l'uso dell'interfaccia *Serializable*, necessaria per poter trasferire le sue istanze sugli stream di input e output.

```
class Persona implements Serializable
```

Nel programma vengono create tre istanze *p1*, *p2* e *p3* che verranno scritte nel file usando il metodo *writeObject* nel seguente modo:

```
fOUT.writeObject(p1);
```

Si noti l'uso del metodo *flush*, eseguito subito dopo le tre operazioni di scrittura, per rendere effettivo il trasferimento dei dati verso la destinazione.

Il programma completo è riportato di seguito.

IMPLEMENTAZIONE DELLA CLASSE (*Persona.java*)

```
import java.io.*;

class Persona implements Serializable
{
  public String nome;
  public int eta;

  // costruttore
  public Persona(String nome, int eta)
  {
    this.nome = nome;
    this.eta = eta;
  }
}
```

PROGRAMMA JAVA (*ScriviDati.java*)

```
import java.io.*;

class ScriviDati
{
  public static void main(String argv[])
  {
    Persona p1, p2, p3;
    p1 = new Persona("Mario", 31);
    p2 = new Persona("Anna", 25);
    p3 = new Persona("Luigi", 57);

    try
    {
      FileOutputStream f = new FileOutputStream("elenco.dat");
      ObjectOutputStream fOUT = new ObjectOutputStream(f);

      fOUT.writeObject(p1);
      fOUT.writeObject(p2);
      fOUT.writeObject(p3);
      fOUT.flush();
      f.close();
    }
    catch(Exception e)
    {
      System.out.println("Eccezione: " + e.getMessage());
    }
  }
}
```

• **Lettura**
Per leggere le informazioni contenute in un file strutturato, si usano i metodi della classe **ObjectInputStream**. Questi metodi assumono la forma **read*Dato***, dove al posto di *Dato* si sostituisce il tipo di dato che si vuole leggere. Per evitare di ottenere inconsistenza, i dati devono essere letti nello stesso ordine in cui sono stati salvati. Tra i metodi di lettura c'è il metodo **readObject** che consente di recuperare un oggetto precedentemente salvato. Questo metodo restituisce un oggetto di classe *Object* che attraverso il casting può essere riportato alla sua classe originaria. La lettura di un oggetto comporta la creazione di una nuova istanza della classe, in cui a ogni attributo viene assegnato il valore letto dal file.
Se non conosce quanti sono i dati contenuti nel file, si può usare un ciclo infinito per continuare a leggere finchè viene generata l'eccezione **EOFException**. Questa eccezione segnala che si è raggiunta la fine del file e non ci sono più dati da leggere; si può quindi interrompere il ciclo.

PROGETTO 10

Leggere dal file *elenco.dat* e stampare le informazioni delle persone memorizzate.

Questo programma di lettura da file utilizza la classe *Persona* definita nel progetto precedente.
Dopo aver aperto il file in lettura, viene attivato un ciclo infinito con la seguente struttura di ripetizione:

```
while (true)
{
}
```

Il ciclo si interrompe con l'esecuzione del comando *break*, dopo che è stata rilevata l'eccezione *EOFException*.

All'interno del ciclo, i dati vengono letti e assegnati all'oggetto *p* con la seguente istruzione, che si occupa di eseguire il casting dalla classe *Object* alla classe *Persona*.

```
p = (Persona) fIN.readObject();
```

Il programma completo è riportato di seguito.

PROGRAMMA JAVA (*LeggiDati.java*)

```
import java.io.*;

class LeggiDati
{
  public static void main(String argv[])
  {
    Persona p;

    try
    {
      FileInputStream f = new FileInputStream("elenco.dat");
      ObjectInputStream fIN = new ObjectInputStream(f);
```

```
      // ciclo infinito per leggere i dati
      while (true)
      {
        try
        {
          p = (Persona) fIN.readObject();
          System.out.println("\nNome: " + p.nome);
          System.out.println("Eta': " + p.eta);
        }
        catch(EOFException e)
        {
          // interrompe il ciclo
          break;
        }
      }
      f.close();
    }
    catch(Exception e)
    {
      System.out.println("Eccezione: " + e.getMessage());
    }
  }
}
```

AUTOVERIFICA

Domande da 11 a 14 pag. 288-289
Problemi da 21 a 28 pag. 291

MATERIALI ON LINE

6. Gestione dei file ad accesso casuale

9 File di testo

Analizziamo ora le principali operazioni sui file di testo che sono molto simili a quanto già visto per i file strutturati.

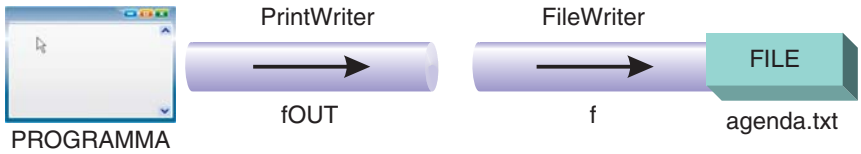
• Apertura e chiusura

L'apertura di un file di testo per le operazioni di output, cioè per la scrittura, viene eseguita con le dichiarazioni dei seguenti oggetti:

```
FileWriter f = new FileWriter("agenda.txt");
PrintWriter fOUT = new PrintWriter(f);
```

La prima riga crea uno stream *f* per scrivere sul file *agenda.txt*. Le operazioni di scrittura non vengono fatte direttamente su questo stream, ma sul secondo stream creato dalla classe *PrintWriter*. È in questa classe che sono contenuti i metodi per la scrittura.

Rappresentando graficamente gli stream si ottiene la seguente figura:



Usando i precedenti comandi, se il file *agenda.txt* esiste viene sovrascritto e viene cancellato tutto il suo contenuto. Se si vuole aprire un file esistente, per accodare dei valori, si deve sostituire la creazione del *FileWriter* nel seguente modo:

```
FileWriter f = new FileWriter("agenda.txt", true);
```

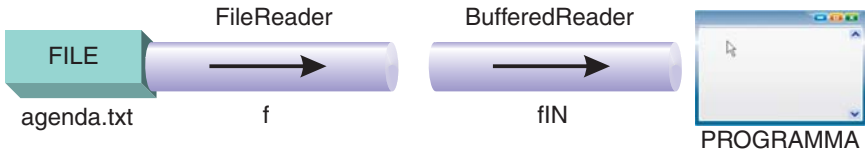
Il valore del secondo parametro indica la condizione di accodamento (*append*): se è *true* il file viene aperto per aggiungere dei dati in coda a quelli preesistenti, altrimenti, se non è presente o è *false,* il file viene aperto in scrittura e ciò comporta la cancellazione di un eventuale archivio preesistente.

L'apertura di un file di testo per le operazioni di input, cioè per la lettura, viene eseguita con le dichiarazioni dei seguenti oggetti:

```
FileReader f = new FileInputStream("agenda.txt");
BufferedReader fIN = new BufferedReader(f);
```

La prima riga crea uno stream *f* per leggere dal file *agenda.txt*. Le operazioni di lettura non vengono fatte direttamente su questo stream, ma sul secondo stream creato dalla classe *BufferedReader*. È in questa classe che sono contenuti i metodi per la lettura dei dati memorizzati.

Rappresentando graficamente gli stream si ottiene la seguente figura:



La chiusura di uno stream, sia di input che di output, viene fatta richiamando il metodo *close* nel seguente modo:

```
f.close();
```

Come per i file strutturati, le operazioni che gestiscono l'I/O dei file di testo possono generare diverse eccezioni, che devono essere gestite racchiudendo le istruzioni in blocchi *try…catch*.

- **Scrittura**

La scrittura di un file di testo che utilizza lo stream *PrintWriter* viene eseguita con i metodi **print** e **println**. Questi metodi sono simili a quelli richiamati con *System.out* e usati per scrivere sullo standard output. Entrambi i metodi scrivono caratteri, la differenza è che il metodo *println* aggiunge alla fine un carattere speciale indicante il ritorno a capo (\n).

PROGETTO 11

Creare un archivio per memorizzare un'agenda telefonica.

Si vuole creare un archivio su file che permetta di memorizzare cognome, nome e numero di telefono. Questi dati vengono inseriti da tastiera attraverso un'iterazione che si arresta quando al posto del cognome viene digitato il carattere '*'. La memorizzazione di questi dati viene effettuata su un file di testo in modo da rendere visibile l'archivio anche con un programma per l'editazione dei testi. Il cognome, il nome e il numero di telefono vengono memorizzati su una riga di testo e ognuno viene separato con il carattere ';'.

Gli attributi e i metodi della classe *Contatto* sono descritti nel seguente diagramma.

Contatto
cognome nome telefono
leggi stampa

Si noti che, a differenza dei file strutturati, le istanze della classe *Contatto* non verranno scritte direttamente nel file e quindi non implementano l'interfaccia *Serializable*.

Le operazioni di apertura del file sono racchiuse da un blocco *try...catch* per gestire una eventuale eccezione *IOException*. In questo caso il programma si interrompe con codice di errore 1, come mostra il seguente frammento di codice.

```
try
{
  f = new FileWriter("agenda.txt");
  fOUT = new PrintWriter(f);
}
catch (IOException e)
{
  System.out.println("Errore nell'apertura del file.");
  System.exit(1);
}
```

Il programma completo è riportato di seguito.

IMPLEMENTAZIONE DELLA CLASSE (*Contatto.java*)

```
import java.io.*;

class Contatto
{
  public String cognome;
  public String nome;
  public String telefono;

  public Contatto() {}
```

```
  public Contatto leggi()
  {
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);

    try
    {
      System.out.print("Inserisci il cognome (* per finire): ");
      cognome = tastiera.readLine();

      // controlla la fine dell'inserimento
      if (cognome.equals("*"))
      {
        return null;
      }

      System.out.print("Inserisci il nome: ");
      nome = tastiera.readLine();

      System.out.print("Inserisci il telefono: ");
      telefono = tastiera.readLine();
    }
    catch(IOException e) {}
    return this;
  }

  public void stampa()
  {
    System.out.println(cognome + "\t" + nome + "\tTel. " + telefono);
  }
}
```

PROGRAMMA JAVA (*ScriviAgenda.java*)

```
import java.io.*;

class ScriviAgenda
{
  // costruttore vuoto
  public ScriviAgenda() {}

  public void scrivi()
  {
    Contatto c = null;
    FileWriter f = null;
    PrintWriter fOUT = null;
```

```
    try
    {
      f = new FileWriter("agenda.txt");
      fOUT = new PrintWriter(f);
    }
    catch (IOException e)
    {
      System.out.println("Errore nell'apertura del file.");
      System.exit(1);
    }

    c = new Contatto();

    while (c.leggi() != null)
    {
      // scrive sul file
      fOUT.println(c.cognome + ";" + c.nome + ";" + c.telefono);
      fOUT.flush();
    }

    try
    {
      f.close();
    }
    catch (IOException e)
    {
      System.out.println("Errore nella chiusura del file.");
      System.exit(1);
    }
  }

  public static void main(String argv[])
  {
    ScriviAgenda a = new ScriviAgenda();
    a.scrivi();
  }
}
```

• **Lettura**

Per leggere le informazioni contenute in un file di testo si usano due metodi della classe *BufferedReader*: **read** e **readLine**.

Il metodo *read* legge un singolo carattere, ma come valore di ritorno restituisce un intero. Se si vuole ottenere il carattere letto si deve effettuare la conversione al tipo di dato *char* nel seguente modo:

```
char c = (char) fIN.read();
```

Il metodo *read* restituisce il valore -1 quando viene raggiunta la fine del file.

Il metodo *readLine* legge una riga di testo e come valore di ritorno restituisce una stringa. Se è stata raggiunta la fine del file il metodo restituisce un valore *null*.

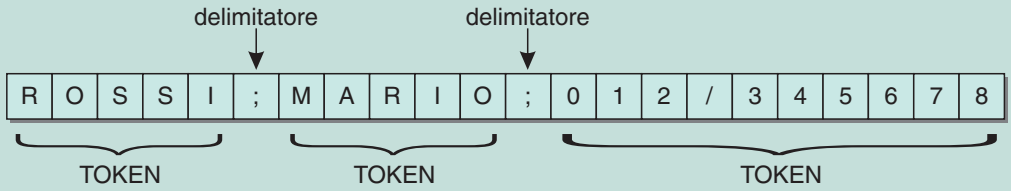
Per effettuare la lettura sequenziale di ogni riga di un file di testo si esegue un ciclo come il seguente:

```
String s;
try
{
  s = fIN.readLine();
  while (s != null)
  {
    // operazioni sulla stringa s

    s = fIN.readLine();
  }
}
catch (IOException e)
{
  System.out.println("Errore nella lettura del file.");
  System.exit(1);
}
```

Questo ciclo utilizza una variabile temporanea *s* nella quale viene memorizzata la riga letta. Il ciclo viene ripetuto finchè la variabile *s* assume il valore *null*. All'interno del ciclo vanno definite le operazioni da applicare alle singole righe.

LA CLASSE *StringTokenizer*

Una classe molto utile per la gestione delle stringhe è **StringTokenizer**, contenuta nel package *java.util*. Questa classe consente di dividere una stringa in pezzi (*token*) in base a caratteri speciali che vengono usati come delimitatori di questi token. Se il delimitatore è lo spazio (' ') allora i token corrispondono alle parole che formano la stringa.
Il seguente esempio mostra come una stringa può essere interpretata usando i token. In questo caso il delimitatore è il carattere punto e virgola (;).



Un oggetto di classe *StringTokenizer* viene costruito con la seguente dichiarazione:

```
StringTokenizer st = new StringTokenizer(stringa, delimitatore);
```

Il costruttore di questa classe riceve come parametri due stringhe: la prima stringa è quella che viene manipolata, la seconda contiene l'elenco di caratteri che vengono usati come delimitatori. I metodi principali della classe *StringTokenizer* sono:

- **hasMoreTokens()**: restituisce il valore *true* se ci sono token disponibili per essere estratti dalla stringa;
- **nextToken()**: restituisce il token successivo rappresentato da una stringa.

L'iterazione usata per scorrere tutti i token generati dall'oggetto *st* di classe *StringTokenizer* è la seguente:

```
while (st.hasMoreTokens())
{
  token = st.nextToken();

  // operazioni sul token
}
```

PROGETTO 12

Stampa il contenuto dell'agenda telefonica.

Questo programma di lettura da file utilizza la classe *Contatto* definita nel progetto precedente. Per stampare il contenuto dell'agenda telefonica si procede con un'iterazione che comprende la lettura sequenziale di ogni riga, la scomposizione della riga in campi (cognome, nome e telefono) e la sua stampa.
La stringa letta viene passata all'oggetto *st* di classe *StringTokenizer* che divide la stringa usando come delimitatore il carattere punto e virgola. Gli attributi del contatto vengono impostati leggendo i singoli token nel seguente modo:

```
c.cognome    = st.nextToken();
c.nome       = st.nextToken();
c.telefono   = st.nextToken();
```

Si noti che l'oggetto *c* di classe *Contatto* viene sovrascritto ad ogni nuova riga letta dal file. Se il programma avesse richiesto di mantenere i contatti in memoria, invece di un unico oggetto, si sarebbe dovuta creare una struttura dinamica basata sulla classe *Vector*.

Il programma completo è riportato di seguito.

PROGRAMMA JAVA (*LeggiAgenda.java*)

```
import java.io.*;
import java.util.*;

class LeggiAgenda
{
  // costruttore vuoto
  public LeggiAgenda() {}

  public void leggeToken()
  {
    FileReader f = null;
    BufferedReader fIN = null;
    Contatto c;
    String s;
    StringTokenizer st;
```

```
    try
    {
      f = new FileReader("agenda.txt");
      fIN = new BufferedReader(f);
    }
    catch (IOException e)
    {
      System.out.println("Errore nell'apertura del file.");
      System.exit(1);
    }

    c = new Contatto();

    try
    {
      s = fIN.readLine();

      // legge e stampa una riga
      while (s != null)
      {
        st = new StringTokenizer(s, ";");
        c.cognome  = st.nextToken();
        c.nome     = st.nextToken();
        c.telefono = st.nextToken();

        c.stampa();
        s = fIN.readLine();
      }
    }
    catch (IOException e)
    {
      System.out.println("Errore nella lettura del file.");
      System.exit(1);
    }

    try
    {
      f.close();
    }
    catch (IOException e)
    {
      System.out.println("Errore nella chiusura del file.");
      System.exit(1);
    }
  }

  public static void main(String argv[])
  {
    LeggiAgenda a = new LeggiAgenda();
    a.leggeToken();
  }
}
```

Il seguente esempio mostra come utilizzare contemporaneamente uno stream di input e uno di output.

PROGETTO 13

Eseguire la copia di backup di un file di testo.

La copia avviene tra un file sorgente, aperto in lettura, e un file di destinazione aperto in scrittura. Si esegue un'iterazione sul file sorgente leggendo una riga per volta che viene immediatamente scritta nel file di destinazione. Il nome assegnato al file di destinazione viene generato aggiungendo l'estensione *.bak* al nome del file sorgente.
La rappresentazione degli stream coinvolti in questo processo è la seguente:

PROGRAMMA JAVA (*Copia.java*)

```
import java.io.*;

class Copia
{
  public static void main(String argv[])
  {
    String filename = "readme.txt";
    String s;
    FileReader f1 = null;
    BufferedReader fIN = null;
    FileWriter f2 = null;
    PrintWriter fOUT = null;

    try
    {
      // sorgente
      f1 = new FileReader(filename);
      fIN = new BufferedReader(f1);
      // destinazione
      f2 = new FileWriter(filename + ".bak");
      fOUT = new PrintWriter(f2);
    }
    catch (IOException e)
    {
      System.out.println("Errore nell'apertura del file.");
      System.exit(1);
    }

    try
    {
      s = fIN.readLine();
```

```
      while (s != null)
      {
        fOUT.println(s);
        fOUT.flush();
        s = fIN.readLine();
      }
    }
    catch (IOException e)
    {
      System.out.println("Errore nella lettura del file.");
      System.exit(1);
    }

    try
    {
      f1.close();
      f2.close();
    }
    catch (IOException e)
    {
      System.out.println("Errore nella chiusura del file.");
      System.exit(1);
    }
  }
}
```

AUTOVERIFICA

Domande da 15 a 16 pag. 289
Problemi da 29 a 34 pag. 291

MATERIALI ON LINE

7. Operazioni sulle directory e sui file

DOMANDE

Array dinamici

1 Quali di queste affermazioni, riferite alla classe *Vector*, sono vere (V) e quali false (F)?

a) Rappresenta un array di oggetti dinamico V F
b) È inclusa nel package *java.io* V F
c) Le sue dimensioni sono fissate con il costruttore V F
d) Il primo elemento del vettore ha indice 0 V F
e) La sua capacità si modifica dinamicamente V F

2 Quale tra queste dichiarazioni *non* usa correttamente il costruttore della classe *Vector*?

a) Vector v = new Vector();
b) Vector v = new Vector(5);
c) Vector v = new Vector(5,2);
d) Vector v = new Vector(true);

3 A quale classe appartiene l'oggetto restituito dal metodo *elementAt*?

a) Integer
b) Object
c) String
d) Vector

4 Quali di queste affermazioni, riferite alla gestione della memoria, sono vere (V) e quali false (F)?

a) In Java la deallocazione deve essere eseguita dal programmatore V F
b) Il gestore della memoria può stabilire che un oggetto non è più utilizzato contando il numero di riferimenti a quell'oggetto V F
c) Prima che il Garbage Collector provveda all'eliminazione di un oggetto, viene eseguito un metodo particolare detto finalizzatore V F
d) Il Garbage Collector resta attivo durante tutta l'esecuzione di un'applicazione Java V F

Pila e Coda

5 Completa la seguente tabella, che mette a confronto la pila e la coda, utilizzando le parole elencate alla fine della domanda.

	Pila	Coda
Modalità di gestione		
Operazioni di modifica		
Operazioni di interrogazione		

FIFO, push, vuota, pop, togli, LIFO, size, top, aggiungi

6 Quale delle seguenti istruzioni viene usata per realizzare l'operazione di *pop* in una pila?

a) obj = elementi.elementAt(size);
b) obj = elementi.elementAt(size-1);
c) obj = elementi.elementAt(1);
d) obj = elementi.elementAt(0);

AUTOVERIFICA

AUTOVERIFICA

7 Quale delle seguenti istruzioni viene usata per realizzare l'operazione di *togli* in una coda?
a) obj = elementi.elementAt(size);
b) obj = elementi.elementAt(size-1);
c) obj = elementi.elementAt(1);
d) obj = elementi.elementAt(0);

Lista e Albero

8 Quali di queste affermazioni, riferite alla lista concatenata, sono vere (V) e quali false (F)?
a) Ogni elemento conosce qual è il suo successore V F
b) Il riferimento al nodo *testa* è superfluo V F
c) Viene gestita utilizzando i riferimenti agli oggetti V F
d) Se la lista contiene un solo elemento, il riferimento del nodo *testa* è uguale al riferimento del nodo *coda* V F
e) Le operazioni di inserimento ed eliminazione sono veloci V F

9 Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.
a) Il riferimento al primo elemento di una lista è detto .....
b) Il riferimento all'ultimo elemento di una lista è detto .....
c) Con l'operazione di ..... si naviga tra gli oggetti della lista.
d) Ogni nodo della lista contiene un ..... al nodo successivo.
attraversamento, inizio, testa, fine, riferimento, coda, visita, ultimo, nodo

10 Associa gli elementi della colonna di sinistra alle corrette definizioni scegliendo nella colonna di destra.

a) la radice	1) è il nodo terminale di un albero
b) le foglie	2) è il nodo da cui partono le connessioni che generano l'albero
	3) sono i nodi esterni che non hanno nodi figli
	4) sono i nodi interni dell'albero

File strutturati

11 In quale package sono contenute le classi e i metodi che gestiscono le operazioni di Input/Output?
a) java.awt
b) java.io
c) java.util
d) java.input

12 Quali di queste affermazioni, riferite agli stream, sono vere (V) e quali false (F)?
a) Sono delle sequenze ordinate di dati V F
b) Si dividono in stream di input e stream di output V F
c) Possono essere concatenati tra loro V F
d) Sono utilizzati solo per i file di testo V F
e) Le sottoclassi della classe *Reader* gestiscono gli stream di input V F

13 Associa le classi a sinistra alle corrette sottoclassi di destra.

a) InputStream	1) FileOutputStream, ObjectOutputStream
b) OutputStream	2) PrintWriter, OutputStreamWriter, FileWriter
c) Reader	3) FileInputStream, ObjectInputStream
d) Writer	4) BufferedReader, InputStreamReader, FileReader

14 Quale delle seguenti istruzioni viene usata per aprire un file strutturato per l'accodamento?
a) FileWriter f = new FileWriter("xyz");
b) FileOutputStream f = new FileOutputStream("xyz");
c) FileWriter f = new FileWriter("xyz", true);
d) FileOutputStream f = new FileOutputStream("xyz", true);

File di testo

15 Associa ciascun metodo della colonna a sinistra con la corrispondente classe scegliendo nella colonna a destra.

a) writeInt	1) ObjectOutputStream
b) close	2) ObjectInputStream
c) readObject	3) PrintWriter
d) print	4) BufferedReader
e) readLine	
f) flush	
g) println	
h) writeObject	

16 Quale delle seguenti iterazioni viene usata per scorrere tutti i token dall'oggetto *st* di classe *StringTokenizer*?

```
a) while (st.hasMore)
   {
      token = st.next;
   }

b) while (st.hasMoreTokens())
   {
      token = st.nextToken();
   }

c) while (hasMoreTokens())
   {
      token = nextToken();
   }

d) while (st.nextToken())
   {
      token = st.hasMoreTokens();
   }
```

PROBLEMI

Array dinamici

1 Il seguente frammento di codice crea un vettore contenente dei nomi e lo stampa a video. Correggere gli errori presenti aiutandosi anche con il compilatore.

```
Vector v = new Vector(10, 5, 1);
String s;
v.addElement("Maria");
v.addElement("Elena");
for(int i=0; i<=v.size(); i--)
{
  s = v.elementAt(i);
  System.out.println(s);
}
```

2 Dato un elenco di articoli (memorizzato in un vettore dinamico) definito da codice, descrizione e prezzo, si vuole gestire l'inserimento, la cancellazione e la stampa degli articoli tramite un menu.

3 Raccogliere in un vettore i nomi degli studenti e le loro votazioni in una prova. Calcolare la media dei voti e il numero di prove sufficienti.

4 Dopo aver caricato in un vettore le temperature rilevate in una settimana, cancellare la temperatura massima e la minima.

5 Le dimensioni, espresse in pollici, e i prezzi dei televisori venduti in un negozio sono letti da tastiera e inseriti in un vettore. Calcolare, in percentuale, il numero di televisori da 40 pollici e la media del loro prezzo.

Pila e Coda

6 Usando una rappresentazione grafica, mostrare come cambia una pila di numeri interi in seguito alle seguenti operazione: *push(8)*, *push(3)*, *push(5)*, *pop()*, *pop()*, *push(1)*. Dopo aver eseguito queste operazioni che risultati restituirebbero le operazioni *conta()*, *top()* e *vuota()*?

7 Implementare un metodo *stampa* da aggiungere alla classe *Pila* con il compito di mostrare a video tutti gli elementi contenuti nella pila partendo dal fondo.

8 Rappresentare la struttura a pila con un array statico.

9 Utilizzare la struttura di dati pila per implementare la conversione di un numero dalla base 10 alla base 2. Il risultato è ottenuto prendendo i resti della divisione per 2 in ordine opposto a come sono stati generati. (Suggerimento: se si aggiungono i resti a una pila, il successivo svuotamento della stessa permette di ottenere i resti in ordine opposto).

10 Rappresentare la struttura a coda con un array statico.

11 Simulare le operazioni (*ordina* e *soddisfa*), relative alla gestione delle ordinazioni eseguite da un bar, creando un menu attraverso il quale, da tastiera, si specifica quale tipo di operazione effettuare. È necessario effettuare il controllo che non possa essere eseguita l'operazione *soddisfa* se prima non è stata eseguita almeno un'operazione *ordina*.

12 Realizzare la struttura di dati di nome *doppia coda*. Questa struttura consente l'inserimento e l'eliminazione degli elementi da entrambi i lati che vengono identificati come lato destro e lato sinistro. Descrivere questa nuova struttura usando il diagramma di classe e realizzare un'implementazione in Java.

Lista e Albero

13 Implementare le seguenti operazioni di interrogazione per una lista concatenata:
- *conta*: restituisce il numero di elementi presenti nella lista,
- *vuota*: indica se la lista è vuota o meno.

14 Implementare le seguenti operazioni di inserimento per una lista concatenata:
- *inseriscilnTesta:* aggiunge l'elemento specificato come parametro in testa alla lista,
- *inseriscilnCoda:* aggiunge l'elemento specificato come parametro in coda alla lista.

15 Realizzare una pila tramite una lista concatenata.

16 Realizzare una coda tramite una lista concatenata.

17 Scrivere un metodo che, date come input due liste, concatena la seconda lista con la prima.

18 Realizzare una lista doppiamente concatenata. In questa struttura, ogni nodo ha due riferimenti, uno al nodo successivo e uno al nodo precedente. Le operazioni da implementare sono: l'inserimento di nuovi oggetti in fondo alla lista, la stampa ordinata (dal primo all'ultimo elemento) e la stampa inversa (dall'ultimo elemento al primo).

19 Data la sequenza di numeri 40, 25, 56, 12, 31, 77, 43, 89, 2, 18, costruire su un foglio lo schema per l'albero binario che consente di mantenere ordinata la lista di numeri. Qual è il risultato che si ottiene visitando l'albero in ordine simmetrico, in ordine anticipato e in ordine posticipato?

20 Gestire tramite un albero binario una sequenza ordinata di nomi. Le operazioni consentite sono l'inserimento di un nuovo nome, la ricerca con controllo se un nome è presente nell'albero e la stampa in ordine alfabetico.

File strutturati

21 Realizzare un programma per memorizzare su un file strutturato il nome di una città, la provincia e il numero di abitanti.

22 Realizzare un programma che legge il precedente file strutturato e calcola il numero medio di abitanti delle città memorizzate.

23 Realizzare un programma che memorizzi su un file le informazioni relative a un insieme di libri introdotte da tastiera. Un libro è composto da titolo, autore e prezzo. Si consiglia di creare una classe per descrivere l'oggetto libro.

24 Realizzare un programma che visualizzi tutti i libri, precedentemente memorizzati in un file, il cui prezzo è maggiore di un valore introdotto da tastiera.

25 Realizzare un programma che memorizzi su un file 100 numeri interi. Scrivere un secondo programma che legge il file di dati calcolando la somma, la media e la varianza dell'insieme di numeri.

26 In un file vengono registrate, acquisendole dalla tastiera, le informazioni sui giocatori, e le rispettive squadre, iscritti ad un campionato; successivamente, con un programma di lettura del file, si vogliono visualizzare i nomi dei giocatori di una squadra richiesta.

27 I prodotti venduti in un grande magazzino sono suddivisi in tre reparti; un primo programma registra sul file i dati con le informazioni sui prodotti (descrizione, numero del reparto, prezzo); un secondo programma, leggendo il file creato, conta quanti sono i prodotti venduti per ciascun reparto.

28 Da tastiera vengono inseriti per ogni contribuente il nome, il codice fiscale e il reddito imponibile. Questi dati vengono registrati in un file con un programma di scrittura. Con un secondo programma di lettura, si deve calcolare l'imposta del 5.5% sul reddito, producendo in stampa nome, codice fiscale e ammontare dell'imposta per ciascun contribuente.

File di testo

29 Scrivere il frammento di codice che divide le seguenti stringhe in *token* e li stampa a video:
"1400-515-298000-33-160-75"
"c:/documenti/word/lettere/personali/auguri.doc"
"do re mi fa sol la si"

30 Scrivere su un file di testo la tavola pitagorica.

31 Leggere e visualizzare a video il contenuto del file *readme.txt*, precedentemente preparato con un editor di testi.

32 Realizzare un programma che legga un file di testo e conti il numero di frasi, usando come delimitatore il carattere punto.

33 Realizzare un programma che esegua la copia di un file di testo inserendo un'interlinea doppia nel file di destinazione. Ogni riga copiata dal file originale deve essere seguita da una riga vuota.

34 Dato un file di testo, scriverne il contenuto su video presentando 20 righe per volta e permettendo all'utente di passare alla "pagina" successiva o di interrompere la visualizzazione.

Focus notes

DYNAMIC DATA STRUCTURES

Dynamic data structures are data structures that grow and shrink during execution by allocating and deallocating memory.

Two data types for manipulating large collections of objects are *stack* and *queue*. Each is defined by two basic operations: *insert* a new item (named *push*), and *remove* an item (*pop*).

The rule used for a queue is to always remove the oldest item in the data structure. This policy is known as *first-in first-out* or *FIFO*.

The rule used for a stack is to always remove the most recent item of the data structure. This policy is known as *last-in first-out* or LIFO.

A *linked list* is a set of nodes: each node contains a unit of data and a reference to the next node in the list.

An example of queue application is the printer scheduler in a multitasking operating system where many users request the same printer: the jobs must be batched up and then sent to the printer according to a FIFO policy.

The most important application of stack is to implement function calls. With each call, the return address is pushed in the stack. When the function finishes executing, it returns control to the main program and the last address is removed (*pop*) from stack (LIFO).

FILE MANAGEMENT

A file is a set of persistent data that is stored on a disk. Files can be created and modified by programs but their data remains available among different program executions.

A file can be managed by means of four main operations:

- *opening*: a link to the file is created;
- *reading*, data is transferred from the disk to the primary memory (RAM);
- *writing*, data is moved from the RAM to the secondary memory (disk);
- *closing*, the link to the file is dropped.

Generally, there are two kinds of computer files: text files and binary files.

Text files contain a sequence of characters and are usually divided into lines. On *Windows*, every line of a text file ends with two characters: CR (*carriage return*) and LF (*line feed*).

Binary files contain data in binary form and cannot be read by text editors.

Focus notes

Glossary

binary tree
A dynamic data structure made of nodes, a node can have two child nodes (left and right).

flush
An operation that empties a buffer, moving all the data to its destination.

garbage collector
An automatic process that deallocates unused objects and frees memory.

linked list
A dynamic data structure made of items, an item is linked only to another item.

pop
An operation that removes an element from the top of a stack.

push
An operation that adds a new element at the top of a stack.

queue
A dynamic data structure with two main operations: *enqueue* and *dequeue*.

serialization
The process of converting objects into a portable format to ease their storage and transmission.

stack
A dynamic data structure with two main operations: *push* and *pop*.

stack overflow
An error occurring when memory is used excessively, usually by infinite recursion.

token
An item resulting from the operation of splitting a string.

Acronyms

CR	Carriage Return
EOF	End Of File
FIFO	First-In First-Out
I/O	Input/Output
LF	Line Feed
LIFO	Last-In First-Out

SCHEDA DI AUTOVALUTAZIONE

CONOSCENZE

- ☐ Creazione dinamica di aree di memoria
- ☐ Array dinamici
- ☐ Gestione automatica della memoria
- ☐ Pila
- ☐ Coda
- ☐ Liste concatenate
- ☐ Albero
- ☐ Flussi di Input/Output
- ☐ Gestione di file strutturati
- ☐ Gestione di file di testo

ABILITÀ

- ☐ Implementare le classi per array dinamici
- ☐ Implementare le classi per pila, coda, albero
- ☐ Utilizzare i metodi per la gestione delle strutture dati dinamiche
- ☐ Implementare le liste concatenate
- ☐ Utilizzare i metodi per la gestione di liste concatenate
- ☐ Utilizzare le istruzioni appropriate di lettura e scrittura dei flussi di dati
- ☐ Implementare le operazioni per la gestione dei file strutturati
- ☐ Utilizzare le istruzioni appropriate di lettura e scrittura dei file di testo

SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 539

Programmazione guidata dagli eventi e interfaccia grafica

1 L'interfaccia per l'utente

I programmi eseguiti da un elaboratore possono essere divisi in due grandi categorie: i programmi che eseguono i loro compiti senza richiedere l'intervento degli utenti, e altri che hanno bisogno di interagire con gli utenti stessi.
Per esempio, un programma che deve tenere aggiornato un orologio, non ha bisogno di alcun input per eseguire i suoi compiti. Al contrario, un programma che esegue i calcoli sui valori inseriti da un utente e ne restituisce il risultato, ha bisogno di interagire con l'utente. Il programma deve essere in grado di chiedere all'utente l'inserimento dei valori e nello stesso tempo deve prevedere le modalità con cui mostrare i risultati dell'elaborazione.
La parte di un'applicazione che interagisce con l'utente prende il nome di **interfaccia utente**.
La presenza di questa interfaccia distingue i programmi interattivi da quelli non interattivi.
L'interfaccia utente indica tutto ciò che serve per la comunicazione tra un programma e il suo utilizzatore (apparecchiature hardware e risorse software). In questa definizione rientra quello che l'utente vede sullo schermo, quello che sente oppure quello che può toccare. Il mouse è una parte dell'interfaccia utente, come pure uno schermo sensibile al tatto (*touchscreen*). Questi rientrano sia nella parte visiva che tattile di un'interfaccia. Il microfono e i programmi che gestiscono la voce fanno parte di un'interfaccia sonora.

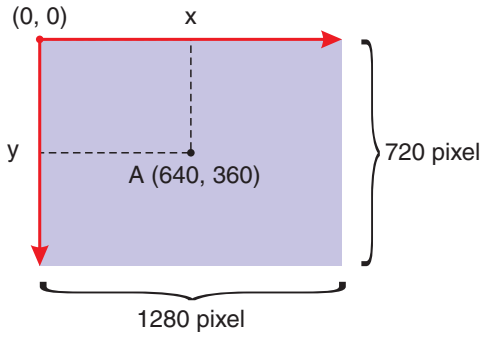
È possibile classificare le interfacce utente in base agli strumenti che usano per interagire con gli utenti. Supponendo di considerare lo schermo come l'elemento principale di interazione tra le persone e l'elaboratore, le interfacce si possono distinguere in interfacce *a caratteri* e interfacce *grafiche*.

Le **interfacce a caratteri** sono molto semplici perché possono visualizzare solo i caratteri. Solitamente, l'interazione avviene inserendo i comandi attraverso la tastiera.
Le **interfacce grafiche** sono più complesse da realizzare ma rendono più piacevole e semplice il modo con cui l'utente interagisce. L'interfaccia grafica è composta da immagini, icone, finestre, pulsanti e molti altri elementi. Le interfacce grafiche si indicano con il termine **GUI**, acronimo di *Graphical User Interface*.
Le pagine seguenti sono dedicate all'interfaccia di tipo GUI e al modo con il quale può essere realizzata nel linguaggio Java.

Le interfacce grafiche sono un'evoluzione delle interfacce a caratteri. Questo passaggio è stato reso possibile dallo sviluppo dei terminali video e dei dispositivi di puntamento oltre che da una riduzione del loro costo. Si è passati da schermi che visualizzavano solo caratteri a schermi grafici.

La videata degli schermi grafici è composta da una matrice di punti, chiamati **pixel** (*picture element*). A ogni pixel può essere associato un colore e la loro unione forma l'immagine completa. Le dimensioni della matrice indicano la risoluzione del particolare schermo. Dimensioni comuni in pixel per gli attuali monitor sono 1366x768, 1280x720 (*HD*), 1920x1080 (*Full HD*) dove il primo numero indica il numero di pixel che compongono una linea orizzontale, mente il secondo numero indica i pixel presenti su una linea verticale.
Queste dimensioni sono indicate come **risoluzione** dello schermo.

Un punto all'interno di una videata può essere indicato usando una coppia di numeri (x,y). Queste rappresentano le coordinate del punto.
Il sistema di riferimento che si adotta con lo schermo ha come origine il punto (0,0) che si trova in alto a sinistra. Spostandosi verso destra viene incrementato il valore delle *x*, mentre spostandosi in basso viene incrementato il valore delle *y*.



In figura viene mostrato il sistema di riferimento di uno schermo di dimensione 1280x720.
Il punto A, che si trova nel centro dello schermo, ha come coordinate (640,360).

Il principale dispositivo di puntamento che viene usato con le interfacce grafiche è il **mouse**. Le interfacce che supportano questo dispositivo vengono anche chiamate *point and click* (*punta e fai clic* sul tasto del mouse). Questo dispositivo consente infatti di muovere un puntatore che viene visualizzato sullo schermo. Il puntatore si trova, in ogni istante, in un certo punto dello schermo individuato dalla sue coordinate lungo i due assi. Dopo aver posizionato il puntatore, si possono premere i tasti del mouse per attivare particolari operazioni: il mouse può essere usato per spostarsi da una finestra all'altra, per indicare o selezionare gli oggetti oppure per attivare i programmi o le azioni dell'utente.

2 Gli elementi dell'interfaccia grafica

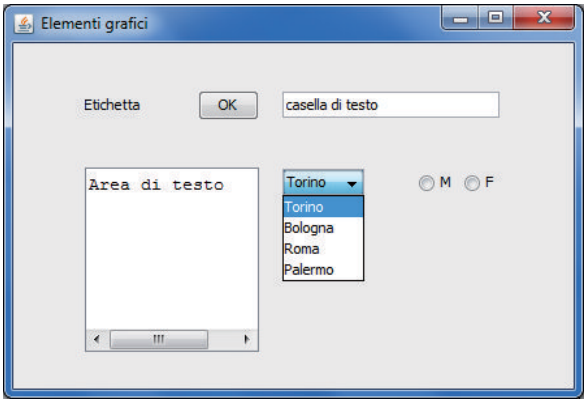
L'elemento principale di un'interfaccia grafica è la **finestra**. Rappresenta un'area dello schermo all'interno della quale vengono eseguite le applicazioni. Può coincidere o meno con l'intera videata dello schermo a seconda di come vengono impostate le sue dimensioni. La finestra è usata per contenere e posizionare gli altri elementi grafici. Tra i compiti della progettazione di un'interfaccia grafica c'è la scelta della disposizione (in inglese **layout**) degli elementi grafici all'interno di una finestra. Questa scelta non è banale e da questa può dipendere il successo o meno dell'applicazione.
Una finestra possiede alcune funzionalità caratteristiche: si può ingrandire a pieno schermo, ridimensionare oppure ridurre a icona; la sua chiusura corrisponde solitamente alla terminazione del programma.

Un altro elemento grafico è il **pulsante** (o *bottone*). Può essere etichettato con una stringa oppure con un'immagine. Le applicazioni associano ai pulsanti le particolari azioni che l'utente è abilitato a svolgere. L'attivazione di queste azioni avviene con un clic del mouse sul pulsante.

Gli elementi grafici che gestiscono l'inserimento del testo sono le *caselle di testo* e le *aree di testo*.
La **casella di testo** è formata da una cella in cui è possibile inserire solo una riga, è usata dalle applicazioni per richiedere all'utente l'inserimento di numeri o stringhe.

L'**area di testo**, invece, è formata da più linee, viene utilizzata per inserire o per mostrare le righe di testi particolarmente lunghi.

Un'applicazione può interagire con l'utente offrendo la possibilità di scegliere tra un numero prefissato di alternative. Questa scelta si può realizzare all'interno di un'interfaccia grafica usando diversi elementi grafici.
La **casella combinata** mostra le alternative tramite una casella di testo che incorpora un menu a tendina. L'utente, usando il mouse, può aprire e scorrere la tendina per indicare una delle alternative.
Un altro modo molto comune per effettuare le scelte, soprattutto quando sono legate all'esecuzione di particolari operazioni, è rappresentato dai **menu**. I menu sono solitamente posizionati nella parte alta della finestra. Si attivano con un clic del mouse che mostra l'elenco di voci contenute nel menu: scegliendo una voce si attiva l'azione corrispondente.
Il **pulsante di opzione** (avente la forma di un piccolo cerchio) permette di effettuare la scelta tra valori booleani, cioè gestisce solo due stati, corrispondenti al valore *true* se il cerchio è annerito e al valore *false* in caso contrario. Si possono raggruppare più pulsanti di opzione in un unico gruppo: in questo modo può essere effettuata una scelta esclusiva tra gli elementi del gruppo.
La figura mostra, a partire dall'alto a sinistra, un'etichetta, un pulsante, una casella di testo, un'area di testo, una casella combinata e due pulsanti di opzione.



Altri elementi grafici che possono far parte di un'interfaccia grafica sono: le barre di scorrimento, le immagini e i menu *popup*, cioè i menu di scelte che vengono aperti all'interno della finestra (come i menu di scelta rapida attivabili in Windows con il tasto destro del mouse).

Avendo a disposizione tutte le componenti precedentemente elencate, è possibile combinarle per **progettare un'interfaccia grafica**.
La realizzazione di una GUI dipende molto dal tipo di programma a cui è destinata. Il numero di menu, di pulsanti, di caselle di testo va scelto in base a quello che l'applicazione deve fare. Se per esempio è richiesto l'inserimento di due valori, sarà necessario predisporre due caselle di testo.
Anche la disposizione degli elementi grafici ha una sua importanza che è sempre legata agli scopi del programma. Un programma per l'elaborazione dei testi riserva la parte centrale a un'area di testo, disponendo i menu e i pulsanti nella parte superiore o inferiore della finestra.

Per costruire un'applicazione grafica bisogna quindi eseguire due operazioni:
- individuare gli elementi grafici che servono
- disporre questi elementi all'interno di una finestra.

Queste operazioni devono essere svolte durante la fase di analisi del problema. È utile dare una descrizione dell'interfaccia grafica che si vuole utilizzare, indicando quali sono gli elementi e quale posizione essi devono occupare.

Un linguaggio di programmazione che gestisce le interfacce grafiche deve consentire l'utilizzo degli elementi grafici precedentemente descritti. Il linguaggio Java consente al programmatore di creare gli oggetti da inserire nella GUI; offre anche diversi modi per disporre e dimensionare questi oggetti all'interno di una finestra.
I programmi Java, presentati nei capitoli precedenti, utilizzano un'interfaccia a caratteri. L'input e l'output vengono eseguiti dal *Prompt dei comandi*. Questo modo di interagire con il programma è semplice e molto limitato. Le uniche cose che si possono fare sono la visualizzazione di messaggi testuali e l'inserimento di stringhe utilizzando la tastiera. Non è permesso l'utilizzo del colore, del mouse, di immagini, dei pulsanti e di tutti gli altri elementi grafici.
Le applicazioni che usano una GUI risultano più complesse e di dimensioni maggiori rispetto a quelle realizzate con un'interfaccia a caratteri. Si ha però il vantaggio di creare programmi *user friendly* che facilitano l'interazione con l'utente.

I paragrafi successivi mostrano come costruire un'interfaccia grafica usando l'ambiente di sviluppo grafico **NetBeans**.

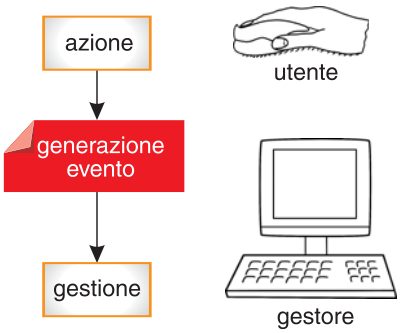
GLI ELEMENTI GRAFICI COME OGGETTI DELLA OOP

I diversi elementi grafici della GUI possono essere rappresentati in modo naturale utilizzando gli **oggetti**.
Infatti una finestra può essere vista come un oggetto i cui *attributi* sono la dimensione, la posizione e il titolo. I *metodi* di una finestra consentono di modificarla, cambiandone le dimensioni, spostandola in primo o in secondo piano, rendendola visibile o nascondendola.
Anche le caselle di testo possono essere considerate degli oggetti: hanno un *attributo* che è usato per memorizzare il valore digitato nella casella; i *metodi* consentono di cancellare il contenuto della casella, rendere la casella editabile o meno, oppure impostare il numero di caratteri che possono essere inseriti nella casella.
In generale tutti gli elementi grafici possono essere considerati oggetti. All'interno di un programma che usa una GUI, le operazioni che coinvolgono gli oggetti grafici vengono attivate con i metodi degli oggetti coinvolti. Per esempio, l'operazione di chiusura di una finestra viene eseguita con il metodo di chiusura presente nell'oggetto che rappresenta la finestra.
La programmazione ad oggetti è quindi l'ambiente ideale per lo sviluppo di interfacce utente di tipo grafico. Ogni elemento grafico è un oggetto e ha una sua classe corrispondente: istanziando queste classi si possono creare gli oggetti da inserire nell'interfaccia grafica per l'utente.

3 Programmazione guidata dagli eventi

Un'interfaccia grafica non può essere realizzata usando soltanto gli elementi grafici. Questi rappresentano solo figure, linee colorate, scritte che vengono visualizzate sullo schermo. L'utente le può vedere e può usare il mouse o la tastiera per interagire con esse. Occorre però aggiungere le modalità con le quali viene gestita l'interazione tra l'utente e l'interfaccia.
La OOP, per gestire le interfacce grafiche, usa la **programmazione guidata dagli eventi**: a ogni interazione dell'utente viene associato un evento che deve essere gestito richiamando un'azione appropriata.

Un **evento** è un avvenimento asincrono, cioè può manifestarsi senza rispettare tempi prefissati.
Per esempio l'accensione del televisore o l'apertura del frigorifero sono eventi, mentre non possiamo considerare come evento il compleanno di una persona, perché si verifica a una scadenza precisa.
Un evento, riferito a una GUI sul computer, si verifica in seguito all'interazione dell'utente: possibili eventi sono l'apertura o la chiusura di una finestra, il clic su un pulsante oppure la pressione di un tasto del mouse.



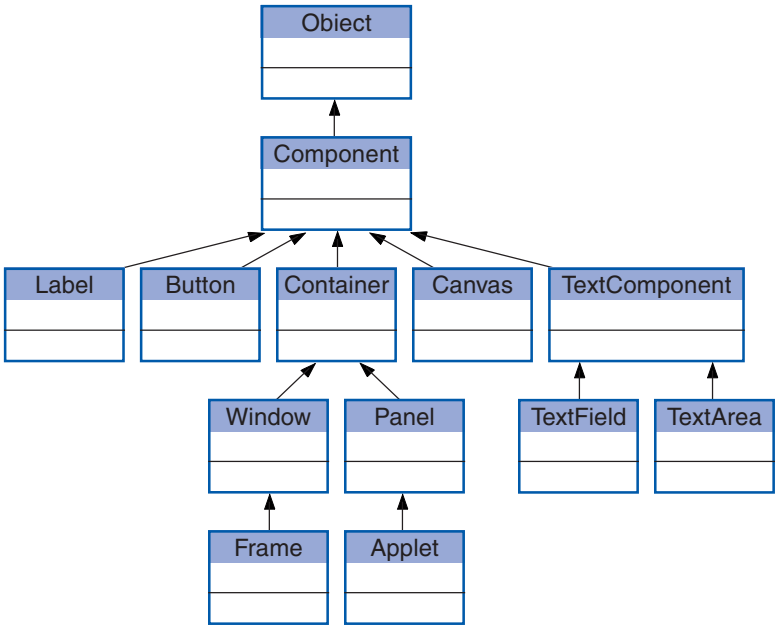
La generazione di un evento è causata da un'azione effettuata dall'utente. L'evento generato contiene quindi tutte le informazioni riguardanti questa azione. Una volta generato un evento, si deve provvedere alla sua gestione, che viene demandata ad un **gestore di eventi**.
Per esempio, l'evento *accensione del televisore* viene generato in seguito all'azione compiuta premendo un particolare tasto. La risposta a questo evento, viene gestita dal televisore che si attiva per visualizzare un determinato canale televisivo.
Un utente che preme un tasto del mouse causa la generazione dell'evento *pressione del tasto del mouse*: esso viene gestito eseguendo un insieme di operazioni associate a quell'evento.
In un programma che usa una GUI gli eventi possono nascere dall'interazione dell'utente con gli elementi grafici. La finestra può generare eventi quando viene chiusa, ridotta a icona oppure ingrandita. I pulsanti generano un evento se si fa clic su di essi.

La parte di un'applicazione, destinata alla gestione degli eventi, viene esplicitamente dichiarata. Se da un lato la generazione degli eventi è causata dall'interazione dell'utente, dall'altro la gestione è affidata a un gestore di eventi.
Un **gestore**, detto anche *handler*, è la parte dell'applicazione software che si preoccupa di dare una risposta in base al tipo di evento ricevuto.
Poiché ogni elemento grafico può generare un evento, bisogna associare ad ognuno un gestore. Se non si vuole gestire l'evento, basta non specificare il gestore. I gestori di un particolare elemento grafico sono *registrati* per gestire quell'elemento e si preoccupano di eseguire le operazioni in risposta ai vari eventi che ricevono.
Ogni volta che un evento viene generato, l'elemento grafico che lo genera richiama un metodo particolare del gestore che è stato registrato per esso.
L'obiettivo fondamentale di un gestore degli eventi, e in generale di un sistema di gestione degli eventi, è di rispondere istantaneamente all'azione di un utente. Se il gestore è occupato a gestire un evento, gli eventi che vengono successivamente generati non devono essere scartati, devono aspettare il loro turno per essere gestiti quando il gestore si libera. Questa strategia viene implementata con una *coda*: gli eventi, man mano che vengono generati, vengono inseriti in modo ordinato in una coda di eventi. Da questa vengono poi prelevati e assegnati al loro gestore. La gestione di questa coda deve essere fatta in modo efficiente per rispondere tempestivamente alle interazioni degli utenti.

4 Le librerie grafiche AWT e Swing

La parte grafica di Java è contenuta nei package **java.awt** e **javax.swing**. È quindi necessario importare entrambe le librerie in ogni programma in cui viene utilizzata la grafica per realizzare una GUI.
L'**AWT** (*Abstract Window Toolkit*) è il sistema utilizzato da Java per definire un insieme di elementi grafici indipendenti dalla piattaforma su cui vengono visualizzati. Questo meccanismo garantisce la portabilità delle applicazioni Java tra piattaforme diverse.

Windows ha un suo modo di visualizzare le finestre, i pulsanti e gli altri elementi grafici che è diverso dal modo usato da *Unix/Linux* oppure da *Mac OS*, ma per tutte queste piattaforme, la costruzione dell'interfaccia grafica prevede l'utilizzo delle stesse classi definite nella AWT.
Swing rappresenta l'evoluzione di AWT nella costruzione delle interfacce grafiche. Il package *javax.swing* mette a disposizione contenitori e componenti, come il package AWT, ma si distingue da quest'ultimo per l'aggiunta di nuove componenti, metodi e funzionalità: per esempio, i pulsanti Swing possono contenere un'immagine anziché una sola scritta, come per i pulsanti AWT. In generale, le componenti Swing sono molto più numerose e sono da preferirsi nella realizzazione delle interfacce utente rispetto ad AWT.
In Java, l'interfaccia utente è formata da due elementi principali: le *componenti* e i *contenitori*.
Una **componente** è un oggetto con una rappresentazione grafica: la sua caratteristica principale è di offrire un'interazione con l'utente. Esempi di componenti sono: i pulsanti, i menu, le caselle per l'inserimento del testo, le barre di scorrimento e così via. Un pulsante, sul quale viene premuto il tasto del mouse, interagisce cambiando forma e facendo capire che è stato azionato. Un campo di testo, a seguito dei tasti digitati sulla tastiera, interagisce mostrando i caratteri corrispondenti ai tasti.
Ogni componente gestisce il modo con cui viene visualizzata, conosce il suo contenuto e reagisce alle eventuali interazioni con l'utente.
Unendo varie componenti si costruisce un'interfaccia utente grafica.
Un **contenitore** è un oggetto che può contenere le componenti. Usando opportuni metodi, si possono aggiungere o togliere le componenti dal contenitore. Il compito del contenitore è quello di posizionare e dimensionare le componenti all'interno del contenitore stesso (**layout** delle componenti). Esistono vari modi con cui questa operazione viene eseguita e dipende dal tipo di gestore del *layout* (in inglese *Layout Manager*) che viene assegnato a quel particolare contenitore.
Un contenitore comunemente usato è la **finestra**: è rappresentata da un'area rettangolare, con un titolo, i pulsanti che gestiscono la chiusura e il ridimensionamento e può contenere varie componenti, per esempio un'area di testo oppure altri pulsanti.
Tutte le componenti, insieme al contenitore, possono essere viste come un'unica entità e considerate come una componente speciale. In questo modo, il contenitore diventa a sua volta una componente e può essere inserita in altri contenitori.
Le classi di Java che realizzano le componenti e i contenitori, sono organizzate in una **gerarchia delle componenti** che ha come padre la classe **Component**.



In figura, per semplificare la rappresentazione, sono mostrate solo le componenti della libreria AWT: *Label, Button*, *Canvas*, *Frame*, *Panel*, *TextField* e *TextArea*.
La classe *Component* è una classe astratta e da questa derivano tutte le classi che realizzano concretamente una componente. Ogni componente viene realizzata creando un'istanza di queste classi.

Si noti che la classe **Container** è una sottoclasse astratta derivata dalla classe *Component* e rappresenta i contenitori. Le sue sottoclassi implementano concretamente i contenitori che possono essere utilizzati per costruire un'interfaccia grafica. Tra i contenitori è presente anche la classe *Applet* che sarà presentata nel prossimo capitolo.

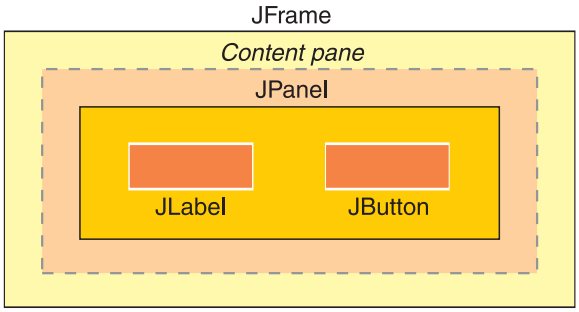
Nella libreria Swing i nomi delle componenti e dei contenitori iniziano con la lettera J e si possono classificare in:

- **contenitori principali** (*top-level container*): finestra principale (*JFrame*), finestra di dialogo (*JDialog*) e finestra di applet (*JApplet*);
- **contenitori intermedi** (*intermediate container*, indicati anche con il termine *pane*), cioè contenitori di altre componenti: pannello (*JPanel*), pannello con barre di scorrimento (*JScrollPane*) o scheda con etichette (*JTabbedPane*);
- **componenti atomiche**: etichetta (*JLabel*), pulsante (*JButton*), casella di testo (*JTextField*), area di testo (*JTextArea*), combo box (*JComboBox*), tabelle (*JTable*).

I **pannelli** (*pane*) hanno lo scopo di organizzare le componenti atomiche nella finestra, semplificandone il posizionamento.

A differenza di AWT, ogni contenitore *top-level* di Swing contiene in senso figurato un contenitore intermedio, detto **content pane** (pannello del contenuto).
Le componenti dell'interfaccia non possono essere inserite direttamente nel contenitore principale, ma solo nel contenitore intermedio. Fa eccezione la barra dei menu che, per convenzione, è collocata in una posizione predefinita della finestra principale, al di fuori del contenitore intermedio.

Lo schema seguente rappresenta l'organizzazione delle componenti Swing per una generica finestra contenente un pannello, al cui interno sono collocati un'etichetta e un pulsante.



La costruzione delle interfacce utente, tramite le componenti e i contenitori AWT e Swing, può essere fatta in modo visuale, usando l'ambiente di programmazione *NetBeans*, oppure in modo testuale scrivendo direttamente il codice Java.
La realizzazione di interfacce utente con *NetBeans* sarà presentata nel paragrafo successivo.
Il progetto seguente, invece, mostra come utilizzare le classi AWT e Swing per istanziare gli oggetti grafici e realizzare un'applicazione scrivendo direttamente il codice Java.

PROGETTO 1

Creare una finestra grafica composta da un'etichetta e un pulsante.

L'interfaccia grafica da creare è quella rappresentata con lo schema precedente, e cioè un'etichetta e un pulsante all'interno di un pannello, inserito all'interno di una finestra.
Per utilizzare le classi grafiche Swing bisogna importare entrambi i package Swing e AWT nel seguente modo:

```
import javax.swing.*;
import java.awt.*;
```

Per ogni elemento grafico, viene creato un oggetto come istanza della relativa classe.
In particolare i contenitori vengono costruiti con le seguenti istruzioni:

```
JFrame f = new JFrame();
JPanel p = new JPanel();
```

Le componenti invece sono costruite istanziando la classe e indicando come parametro il testo da visualizzare nella componente, come mostrano le seguenti istruzioni:

```
JLabel l = new JLabel("Etichetta");
JButton b = new JButton("Bottone");
```

Per aggiungere una componente al contenitore, si usa il metodo **add.** In sequenza, le componenti atomiche (*l* e *b*) vengono prima aggiunte al contenitore intermedio *p*, che a sua volta è aggiunto al contenitore principale *f*.

In Swing, l'aggiunta al contenitore principale viene eseguita tramite un ulteriore contenitore intermedio (oggetto **Container**) che rappresenta il *content pane*, pannello del contenuto della finestra *JFrame*. Il metodo **getContentPane** rende disponibile questo contenitore, al cui interno poi vengono aggiunte le altre componenti nel seguente modo:

```
Container c = f.getContentPane();
c.add(p);
```

In AWT, l'aggiunta al contenitore principale viene eseguita direttamente con l'istruzione:

```
f.add(p);
```

I codici completi per realizzare la finestra con le componenti Swing e con le AWT sono riportati di seguito.

PROGRAMMA JAVA con Swing (*FinestraSwing.java*)

```
import javax.swing.*;
import java.awt.*;

class FinestraSwing
{
  public static void main(String argv[])
  {
    JFrame f = new JFrame();
    JPanel p = new JPanel();
```

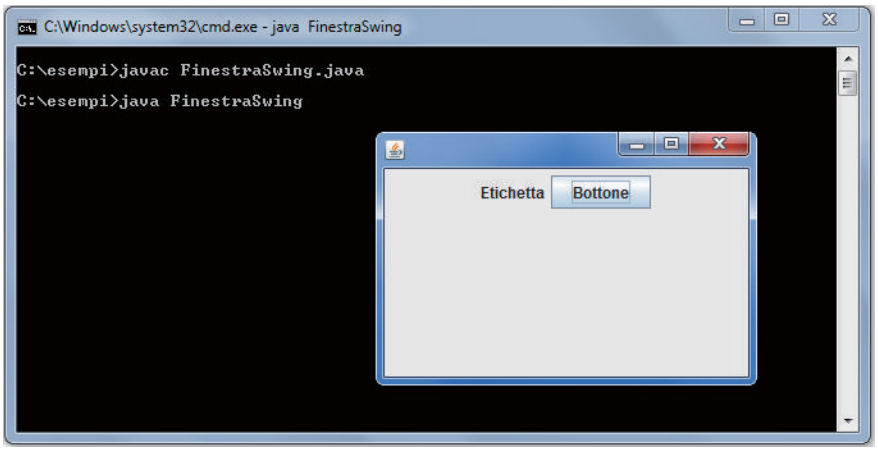
```
    JLabel l = new JLabel("Etichetta");
    JButton b = new JButton("Bottone");
    p.add(l);
    p.add(b);

    Container c = f.getContentPane();
    c.add(p);
    f.setSize(300,200);
    f.setVisible(true);
  }
}
```

PROGRAMMA JAVA con AWT (*FinestraAwt.java*)

```
import java.awt.*;

class FinestraAwt
{
  public static void main(String argv[])
  {
    Frame f = new Frame();
    Panel p = new Panel();
    Label l = new Label("Etichetta");
    Button b = new Button("Bottone");
    p.add(l);
    p.add(b);

    f.add(p);
    f.setSize(300,200);
    f.setVisible(true);
  }
}
```

Si noti che il metodo **setSize** imposta le dimensioni della finestra: i due numeri, passati come parametro, rappresentano la larghezza e l'altezza espresse in *pixel*.

La finestra che viene creata rimane non visibile finché viene richiamato il metodo **setVisible*(boolean)***. Con il valore booleano *true*, la finestra viene resa visibile, con il valore *false* la finestra viene nascosta.



Il programma genera una finestra con le dimensioni indicate, con un'etichetta e un pulsante. La finestra può essere ridimensionata, ingrandita o ridotta a icona, però non può essere chiusa. Per chiudere la finestra, occorre gestire l'evento di chiusura associando ad esso un'istruzione che causa la fine del programma. Non avendo previsto nel programma la gestione di questa operazione, per chiudere la finestra si deve interrompere l'esecuzione del programma premendo la combinazione di tasti **Ctrl + C** dal *Prompt dei comandi*.

AUTOVERIFICA
Problemi da 1 a 4 pag. 347

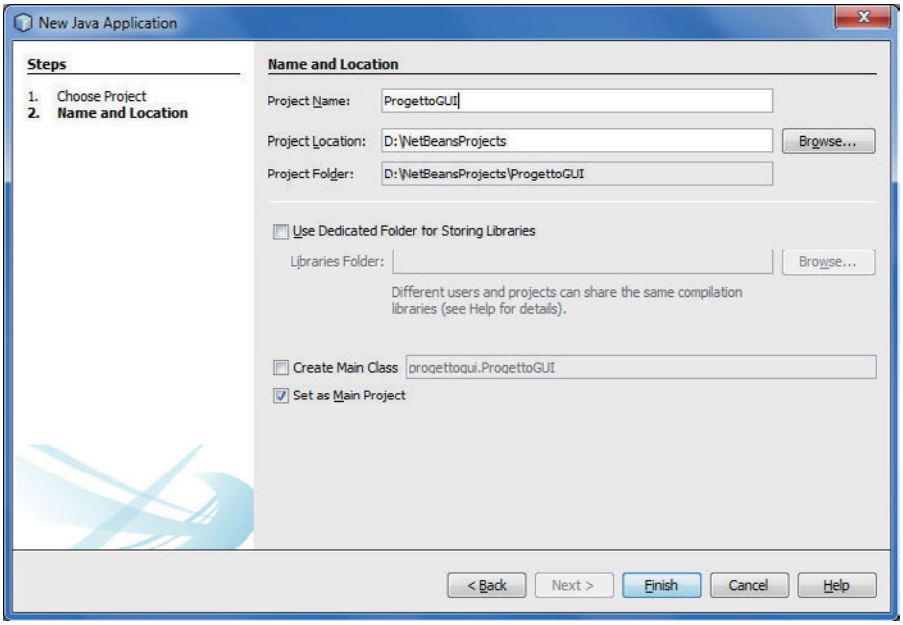
1. I contenitori in AWT

5 L'ambiente di programmazione

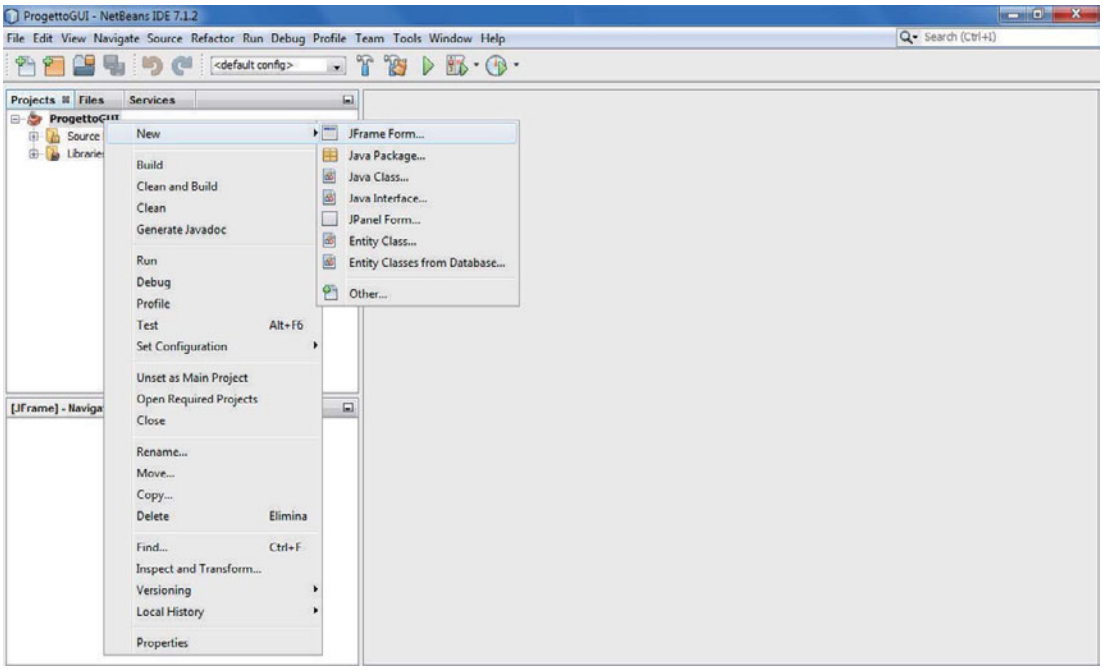
La programmazione delle interfacce grafiche richiede che il programmatore definisca gli elementi grafici e li disponga all'interno di una finestra. L'**ambiente di programmazione e di sviluppo grafico** facilita il lavoro del programmatore in quanto offre gli strumenti per disegnare l'interfaccia grafica e vedere in modo immediato l'aspetto dell'applicazione come apparirà all'utente finale. Questi ambienti di programmazione favoriscono uno stile di **programmazione visuale**, in cui il programmatore manipola direttamente gli elementi dell'interfaccia grafica senza la necessità di scrivere il codice Java, che viene autogenerato.

Nelle pagine seguenti sono presentate le modalità operative per realizzare programmi guidati dagli eventi e interfacce grafiche per l'utente attraverso l'ambiente di sviluppo **NetBeans**, già presentato nell'inserto dopo il Capitolo 3.

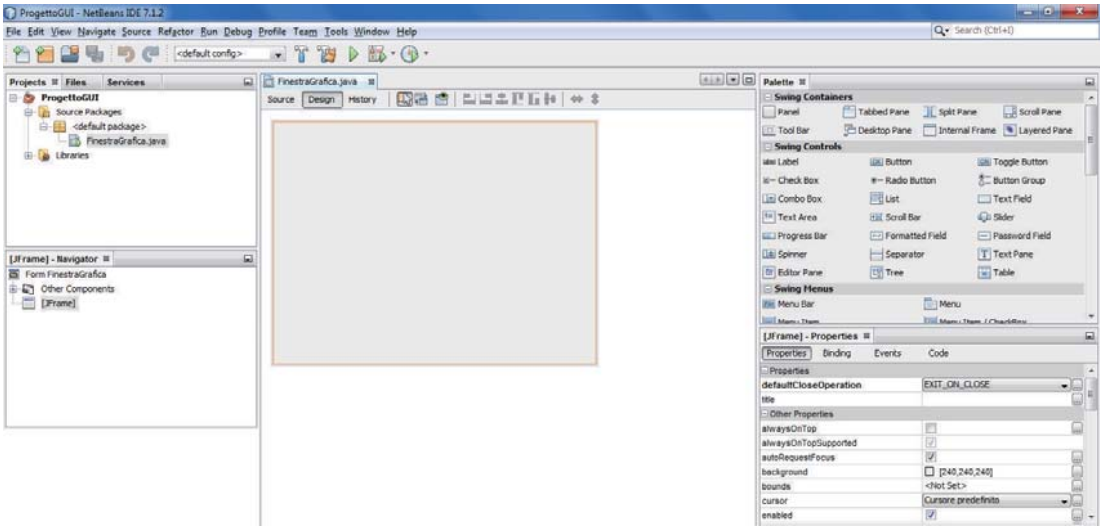
Per creare un nuovo progetto basato su un'interfaccia grafica, si deve fare clic sul menu **File**, **New Project**, poi nella categoria **Java**, scegliere il tipo di progetto **Java Application** e infine fare clic su **Next**.
Nella successiva pagina, scriviamo il nome del progetto nella casella **Project Name** e verifichiamo che non sia stata selezionata l'opzione **Create Main Class**, in quanto la classe principale verrà creata successivamente.



Per completare la creazione del progetto facciamo clic sul pulsante **Finish**. Il nuovo progetto viene visualizzato nella lista dei *Projects* in alto a sinistra.
Ciascuna finestra utilizzata nel programma corrisponde a un oggetto di classe **JFrame**, detto anche *contenitore*.
Un **contenitore** (*container*) è una finestra, o più in generale un elemento grafico sul quale il programmatore inserisce gli elementi grafici che l'utente finale vede. Gli oggetti dell'interfaccia grafica che vengono inseriti in un *container* vengono chiamati **componenti** (*component*) o **controlli** (*control*).
Per aggiungere una finestra al progetto si deve fare clic con il tasto destro sul nome del progetto, poi su **New** e infine su **JFrame Form**.



Viene aperta una finestra in cui si deve indicare il nome della classe principale dell'applicazione.
Per completare la creazione della finestra facciamo clic sul pulsante **Finish**.



La finestra di lavoro principale mostra, nella parte centrale, l'anteprima dell'unico *container* del progetto appena creato.

Nella parte superiore si trova la **Barra dei menu**. Al di sotto della Barra dei menu si trova la **Barra degli strumenti** contenente un insieme di icone che rendono più veloce l'accesso alle scelte di uso più frequente.
Si può modificare l'insieme delle barre visualizzate facendo clic con il tasto destro sulla Barra degli strumenti e selezionando le barre preferite.

L'area sotto la Barra degli strumenti è solitamente divisa in tre zone (sinistra, centrale, destra) che si caratterizzano per la presenza di **finestre operative**. Ogni finestra può essere spostata tra le varie zone, può essere chiusa e nascosta alla visualizzazione del programmatore. L'elenco delle finestre operative che possono essere visualizzate è contenuto nella voce di menu **Window**.

Come impostazione predefinita, nella zona a sinistra è visualizzata la finestra **Projects** che mostra l'elenco dei file del progetto e le librerie utilizzate. Solitamente è anche presente la finestra **Navigator** che, in base all'elemento selezionato, permette di muoversi velocemente tra le sue componenti. Per esempio, se viene selezionato un file *.java*, nella finestra *Navigator* vengono elencati i metodi presenti nel file.

Nella zona centrale viene mostrato sia il codice che l'anteprima del *container* grafico che si sta progettando. Si possono aprire più file contemporaneamente e navigare tra di essi utilizzando le linguette.
All'interno di ogni linguetta, il pulsante **Source** mostra il codice sorgente, mentre il pulsante **Design** mostra l'area in cui si può costruire in modo visuale l'interfaccia utente.
L'anteprima dell'interfaccia utente viene attivata facendo clic sul pulsante

che si trova alla destra del pulsante *Design*.

Nella zona a destra sono presenti le finestre **Palette** e **Properties**. La finestra *Palette* contiene tutte le componenti standard dell'interfaccia utente (caselle di testo, pulsanti, etichette, ecc.) che possono essere trascinate sul *container* per comporre la maschera visualizzata dall'utente finale.
La finestra *Properties* mostra le caratteristiche degli oggetti grafici che possono essere modificate: per esempio la posizione, la dimensione e il tipo di font.

6 Creazione di applicazioni in NetBeans

In generale, per realizzare un'applicazione dotata di interfaccia grafica, si seguono le seguenti fasi:

1. progettare l'**applicazione**
2. progettare le **finestre** che compongono l'interfaccia per l'utente
3. disegnare l'interfaccia, inserendo tutte le **componenti** (pulsanti, menu, ecc.)
4. definire le **proprietà** degli oggetti (posizione, dimensione, ecc.)
5. associare alle componenti il codice di **gestione degli eventi** (per esempio: quando si fa clic sul pulsante X viene eseguito il metodo Y)
6. eseguire il **test** dell'applicazione.

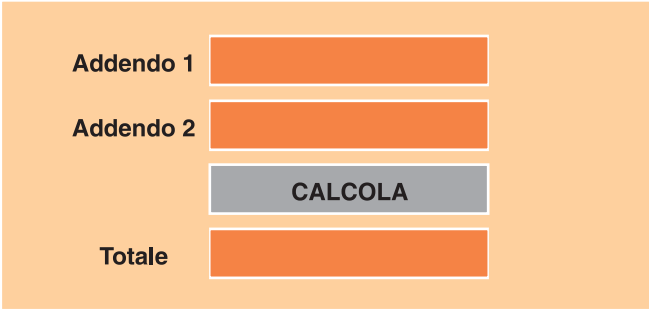
PROGETTO 2

Creare un'applicazione con interfaccia grafica per calcolare la somma di due numeri.

Per creare l'applicazione seguiremo i sei punti elencati in precedenza.

1. *Progettare l'applicazione*
L'applicazione deve mostrare due caselle in cui l'utente inserisce i numeri da sommare. Deve essere presente un pulsante per attivare il calcolo e un'ulteriore casella per visualizzare il totale.

2. *Progettare le finestre*
L'applicazione è composta da una finestra schematizzata nel seguente modo:

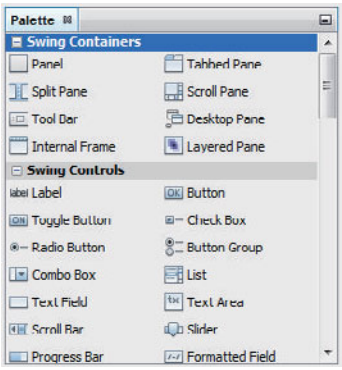


Facendo clic sul pulsante *Calcola*, si ottiene la somma dei due addendi.

3. *Disegnare l'interfaccia*
La creazione dell'interfaccia utente prevede i seguenti passi:
a. Aprire il programma *NetBeans* e creare un nuovo progetto attraverso la scelta *New* nel menu *File*.
b. Assegnare il nome *CalcolatriceProject* al progetto.
c. Aggiungere una *JFrame* facendo clic con il tasto destro sul nome del progetto e poi su *New*, *JFrame Form*.
d. Assegnare il nome *SommaFrame* alla classe del *JFrame*.
A questo punto viene visualizzata la finestra per disegnare l'interfaccia.

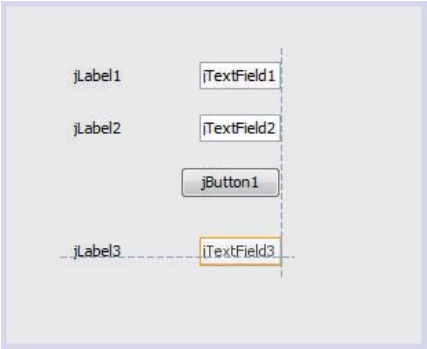
Nella finestra di progettazione aggiungiamo le componenti, selezionandole dalla **Palette**. Il riquadro *Palette* compare solitamente nella parte destra della finestra di *NetBeans*. Se non fosse visibile, lo si può selezionare dal menu *Window*. Le seguenti componenti vanno trascinate dal riquadro *Palette* alla finestra di *Design*:

- due **JLabel** (etichetta) e due **JTextField** (casella di testo) per i dati di input,
- un **JButton** (pulsante) per il comando di azione,
- una **JLabel** e un **JTextField** per il dato di output.



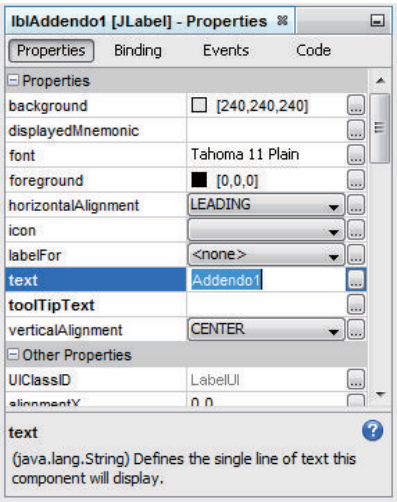
Le componenti *JLabel*, *JTextField* e *JButton* vengono visualizzate sulla finestra di *Design* con nomi predefiniti seguiti da un numero progressivo. Facendo clic su una componente, vengono mostrati, sui bordi dell'oggetto, i quadratini bianchi per il ridimensionamento dell'elemento grafico con il mouse.

Trascinando l'oggetto si può modificarne la posizione e, durante questa operazione, *NetBeans* facilita il loro posizionamento suggerendo l'allineamento migliore per la componente tramite linee tratteggiate.



4. *Definire le proprietà delle componenti*
Molte proprietà, per esempio la posizione all'interno del *JFrame* o le dimensioni della componente, possono essere direttamente modificate agendo sull'oggetto con il mouse. Le altre proprietà vengono visualizzate nella finestra **Properties**, automaticamente visualizzata non appena si seleziona una componente grafica. Se non è visibile nella finestra di *NetBeans,* il riquadro *Properties* si può selezionare dal menu *Window*.

Il testo visualizzato nell'etichetta *jLabel1* può essere modificato con i seguenti passi:
- selezionare l'etichetta dalla finestra di *Design*,
- selezionare la proprietà **Text** nel riquadro *Properties*,
- impostare il valore *Addendo1*.



Ogni controllo posizionato nella finestra di *Design* rappresenta un oggetto, inteso come istanza della classe della relativa componente grafica. Per poterlo utilizzare all'interno del codice Java, occorre attribuire ad esso un nome impostando la proprietà **Variable Name**, che si trova nel riquadro *Properties* all'interno della linguetta *Code*.
Solitamente vengono assegnati nomi con un prefisso che permette di individuare il tipo di oggetto (per esempio **lbl** per le *JLabel*, **btn** per i *JButton* e **txt** per i *JTextField*) e una o più parole di senso compiuto che indicano la funzione (per esempio *lblAddendo1* e *btnCalcola*).

La seguente tabella riassume le proprietà impostate per le componenti dell'interfaccia grafica.

Component	Variable Name	Text
JLabel	lblAddendo1	Addendo1
JLabel	lblAddendo2	Addendo2
JLabel	lblTotale	Totale
JButton	btnCalcola	Calcola
JTextField	txtAddendo1	
JTextField	txtAddendo2	
JTextField	txtTotale	

Oltre alle proprietà *Text* e *Variable Name*, le altre principali proprietà degli oggetti precedenti sono:

background	Imposta il colore di sfondo dell'oggetto.
editable	Determina se la casella di testo può essere modificabile.
font	Specifica le caratteristiche del font utilizzato per i caratteri.
foreground	Imposta il colore di primo piano utilizzato per il testo dell'oggetto.
tooltip	Indica il testo da visualizzare quando l'utente si sposta con il mouse sopra l'oggetto.
border	Imposta lo stile del bordo dell'oggetto.
enabled	Determina se il controllo è attivato, cioè se è in grado di rispondere agli eventi generati dall'utente. Per esempio, se un pulsante non è attivo resta visibile, ma assume un aspetto più sfumato e l'utente non può fare clic.
Horizontal Size	Determina la larghezza dell'oggetto in pixel.
Vertical Size	Determina l'altezza dell'oggetto in pixel.

5. *Gestire gli eventi*
L'evento che si vuole gestire è il clic sul pulsante *btnCalcola*. A seguito del clic, la somma dei due addendi deve essere assegnata alla proprietà *Text* della componente *txtTotale*.
Per fare questo occorre fare doppio clic sul pulsante *btnCalcola* nella finestra di *Design*. In alternativa si può fare clic con il tasto destro, sempre sul pulsante *btnCalcola*, e poi, nel menu di scelta rapida, selezionare *Events*, *Action* e *actionPerformed*.
Si apre la finestra del codice e viene creato automaticamente un metodo *btnCalcolaActionPerformed* che viene eseguito quando l'utente fa clic sul pulsante, cioè quando si attiva l'evento **actionPerformed**.

L'intestazione e i parametri del metodo vengono scritti automaticamente.
Il programmatore deve scrivere, tra parentesi graffe, le istruzioni per calcolare la somma e inserirla nella casella del totale:

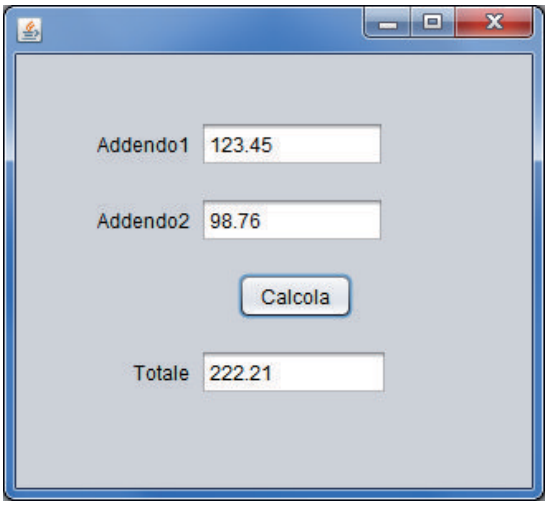
```
private void btnCalcolaActionPerformed(java.awt.event.ActionEvent evt) {
  double num1, num2, totale;
```

```
  num1 = Double.parseDouble(txtAddendo1.getText());
  num2 = Double.parseDouble(txtAddendo2.getText());

  totale = num1+num2;

  txtTotale.setText("" + totale);
}
```

Si noti che la variabile numerica *totale,* per essere visualizzata nella casella di testo, viene trasformata in stringa tramite concatenazione con la stringa vuota ("").
A questo punto occorre salvare il progetto: nel menu **File**, scegliere **Save All** oppure fare clic sull'icona **Save All**

nella Barra degli strumenti.

6. *Effettuare il test dell'applicazione*
Premere l'icona per avviare l'esecuzione dell'applicazione. Se il programma non contiene errori si apre il *JFrame* come una normale finestra di Windows.
Per il debug e l'esecuzione del programma in ambiente *NetBeans*, vale quanto già visto nell'inserto dopo il Capitolo 3.



Nei paragrafi seguenti vengono presentate le componenti grafiche principali del package *javax.swing*. I progetti e i frammenti di codice riportati illustrano come si possono costruire le interfacce utente tramite il codice Java, a partire dalle classi della libreria. Avendo chiari questi aspetti, anche la programmazione visuale in *NetBeans* e il codice generato automaticamente risulteranno più comprensibili.

AUTOVERIFICA
Problemi da 5 a 10 pag. 347

2. La palette delle componenti Swing e AWT
3. Il codice generato automaticamente da NetBeans

7 Etichette e pulsanti

L'**etichetta** è rappresentata dalla classe **JLabel** e può contenere solo una riga di testo. Il contenuto di un'etichetta viene normalmente stabilito durante la progettazione dell'interfaccia e può essere modificato solo dal programma, ma non dall'utente.

Un'etichetta può essere definita attraverso tre diversi costruttori:

- **JLabel()**

  costruttore senza parametri che crea un'etichetta vuota.
- **JLabel(*String*)**

  crea un'etichetta contenente come testo la stringa passata tramite il parametro; la stringa viene allineata a sinistra all'interno dell'etichetta.
- **JLabel(*String, int*)**

  il secondo parametro di tipo intero indica la modalità di allineamento della stringa, usando come valori le **costanti** statiche definite all'interno della classe *JLabel* e precisamente: *JLabel.LEFT* per allineare la stringa a sinistra, *JLabel.RIGHT* per l'allineamento a destra e *JLabel.CENTER* per l'allineamento al centro.

Per esempio:

```
JLabel lblVuota    = new JLabel();
JLabel lblNome     = new JLabel("Mario");
JLabel lblCompleta = new JLabel("Risultato del calcolo", JLabel.CENTER);
```

Per aggiungere queste etichette al pannello *p* si deve usare il metodo **add**:

```
p.add(lblVuota);
p.add(lblNome);
p.add(lblCompleta);
```

Se non si vuole creare esplicitamente l'oggetto, perché non deve essere successivamente manipolato, si può usare anche un modo più diretto per aggiungere una componente. Per esempio:

```
p.add(new JLabel(". . ."));
```

Dopo aver creato un'etichetta, si possono invocare i suoi metodi.
Il metodo **setText(String)** consente di modificare il testo dell'etichetta.
Il metodo **setForeground(Color)** modifica il colore del testo, mentre il metodo **setBackground (Color)** modifica il colore di sfondo dell'etichetta.

I seguenti esempi usano i metodi per modificare le caratteristiche di un'etichetta:

```
lblNome.setText("-- J A V A --");
lblNome.setForeground(Color.red);
lblNome.setBackground(Color.white);
```

Gli ultimi due metodi ricevono come parametro un oggetto della classe **Color**. In questa classe sono presenti alcune costanti statiche che possono essere usate per indicare un particolare colore.

Esempi di **costanti** per il colore sono:

- Color.black
- Color.blue
- Color.cyan
- Color.darkGray
- Color.gray
- Color.green
- Color.lightGray
- Color.magenta
- Color.orange
- Color.pink
- Color.red
- Color.white
- Color.yellow

La classe *JLabel*, come tutte le classi che rappresentano le componenti, oltre ai propri metodi eredita tutti quelli della classe *JComponent* e della classe *Object*. Tra i metodi visti, *setText* è un metodo proprio della classe *JLabel*, mentre *setBackground* e *setForeground* sono ereditati dalla classe *JComponent*.

I **pulsanti** vengono creati usando la classe **JButton**. Solitamente contengono una stringa e sono usati per invocare un'azione quando vengono premuti dall'utente. Il costruttore dei pulsanti può non avere parametri oppure contiene la stringa che viene visualizzata sopra il pulsante.
Per esempio:

```
JButton btnOk = new JButton("OK");
```

Il modo con cui gestire l'evento generato della pressione del pulsante sarà spiegato in un paragrafo successivo dedicato alla gestione degli eventi.
Ogni pulsante creato è attivo, cioè può essere cliccato. È possibile disattivare un pulsante richiamando il metodo **setEnabled(*boolean*)**: esso riceve come parametro il valore booleano *false* per disattivare il pulsante oppure il valore *true* per attivarlo.
Il seguente esempio rende il pulsante *OK* non cliccabile:

```
btnOk.setEnabled(false);
```

8 Caselle e aree di testo

Le **caselle di testo** sono realizzate dalla classe **JTextField**. Questa classe è una sottoclasse della classe *JTextComponent*, la quale ha come sottoclasse anche la classe *JTextArea*. Vedremo come entrambe le sottoclassi usino i metodi che vengono ereditati dalla classe *JTextComponent*.
La classe **JTextField** crea una casella di testo composta da una sola riga che può essere usata per l'input o l'output di stringhe. I costruttori di questa classe consentono di impostare la stringa da visualizzare e il numero di colonne della casella, cioè la sua dimensione.

Esistono quattro costruttori che permettono di impostare i due parametri in diversi modi:

- **JTextField()**
  crea una casella senza specificare il contenuto e la dimensione.
- **JTextField(*String*)**
  crea una casella inizializzata con un certo valore e con dimensione uguale alla lunghezza della stringa.
- **JTextField(*int*)**
  crea una casella vuota di dimensione specificata dal parametro; una casella creata in questo modo è usata per permettere all'utente di inserire i valori.
- **JTextField(*String, int*)**
  consente di impostare sia il contenuto della casella che la sua dimensione espressa dal parametro intero.

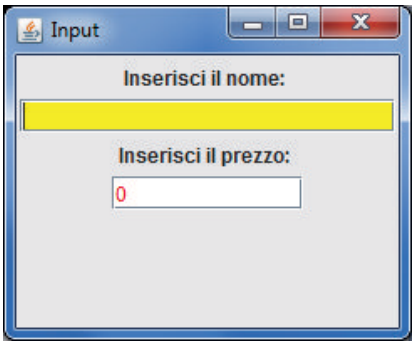
PROGETTO 3

Creare una maschera di input per inserire il nome di un oggetto e il suo prezzo.

Nel programma sono state inserite le etichette per indicare all'utente i tipi di dato che devono essere inseriti nelle caselle di testo. Sia le etichette che le caselle di testo vengono aggiunte a un pannello, che a sua volta viene aggiunto alla finestra del programma.

PROGRAMMA JAVA (*Input.java*)

```
import javax.swing.*;
import java.awt.*;

class Input
{
  public static void main(String argv[])
  {
    JFrame f = new JFrame("Input");
    JPanel p = new JPanel();

    JTextField nome = new JTextField(20);
    JTextField prezzo = new JTextField("0", 10);

    p.add(new JLabel("Inserisci il nome: ", JLabel.RIGHT));
    nome.setBackground(Color.yellow);
    p.add(nome);
    p.add(new JLabel("Inserisci il prezzo: ", JLabel.RIGHT));
    prezzo.setForeground(Color.red);
    p.add(prezzo);

    f.getContentPane().add(p);

    f.setSize(200,200);
    f.setLocation(100,100);

    f.setVisible(true);
  }
}
```



La classe *JTextField* eredita tre metodi molto utili dalla classe *JTextComponent*. Questi metodi potranno essere usati anche con la classe *JTextArea*.
Il metodo **setText(*String*)** consente, dopo che la casella di testo è stata creata, di modificarne il contenuto.
Per esempio si può modificare l'oggetto *prezzo* del progetto precedente nel seguente modo:

```
prezzo.setText("0.00");
```

Il metodo **getText()** consente di leggere il contenuto di una casella di testo. Non ha parametri e restituisce un oggetto di classe *String*. Questo metodo è usato per leggere il valore inserito dall'utente. La lettura dell'input dalla casella di testo sarà spiegata successivamente all'interno della gestione degli eventi.
Il metodo **setEditable(*boolean*)** rende una casella di testo editabile o non editabile.
Una casella è editabile se l'utente, posizionandosi con il mouse, può digitare dei caratteri. Usando il metodo *setEditable*, si può decidere se permettere all'utente la modifica del contenuto della casella (valore *true*) oppure se impedirne le modifiche (valore *false*).
Una casella non editabile è simile a un'etichetta e può essere usata dal programma per mostrare i messaggi inviati all'utente o i risultati di un'elaborazione.
Per rendere non modificabile la casella *prezzo*, bisogna invocare il metodo *setEditable* nel seguente modo:

```
prezzo.setEditable(false);
```

La classe **JTextArea** consente di creare un'area di testo formata da più righe. I costruttori sono simili a quelli della casella di testo con l'aggiunta di un parametro che indica il numero di righe. I costruttori della classe *JTextArea* consentono di impostare la stringa da visualizzare al suo interno, la dimensione dell'area di testo espressa in numero di righe e numero di colonne e la visualizzazione delle barre di scorrimento. Esistono quattro costruttori che permettono di impostare i parametri in vari modi:

- **JTextArea()**
  crea un'area di testo vuota.
- **JTextArea(*int, int*)**
  crea un'area di testo vuota, avente come numero di righe e di colonne i due parametri interi.
- **JTextArea(*String*)**
  crea un'area di testo con uno specifico valore di testo.
- **JTextArea(*String, int, int, int*)**
  crea un'area di testo con uno specifico valore di testo, avente come numero di righe e di colonne i due parametri interi; il quarto parametro è una **costante** intera che specifica la visualizzazione delle barre di scorrimento: *SCROLLBARS_BOTH* (valore di *default*), *SCROLLBARS_VERTICAL_ONLY*, *SCROLLBARS_HORIZONTAL_ONLY*, *SCROLLBARS_NONE*.

Per esempio, la seguente riga di codice definisce un'area di testo composta da 10 righe e larga 30 caratteri:

```
JTextArea messaggi = new JTextArea(10,30);
```

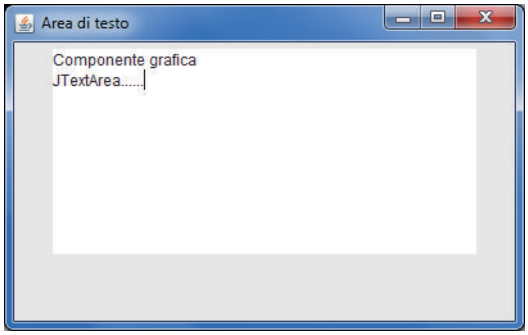
Agli oggetti di questa classe si possono applicare gli stessi metodi usati per *JTextField*.
Con il metodo **setText(*String*)** si modifica il contenuto dell'area di testo: viene cancellato il testo precedente e viene aggiunto il nuovo testo.
Con il metodo **getText()** si legge tutto il contenuto dell'area di testo.
Si può rendere modificabile o meno l'area usando il metodo **setEditable(*boolean*)**.
Le aree di testo hanno un metodo aggiuntivo **append(*String*)** per inserire nuovo testo alla fine dell'area senza cancellare il contenuto già esistente.

PROGETTO 4

Aggiungere un'area di testo di 10 righe e 30 colonne a una finestra.

PROGRAMMA JAVA (*Area.java*)

```
import javax.swing.*;
import java.awt.*;

class Area
{
  public static void main(String argv[])
  {
    JFrame f = new JFrame("Area di testo");
    JPanel p = new JPanel();

    JTextArea messaggi = new JTextArea(10, 30);
    p.add(messaggi);

    f.getContentPane().add(p);
    f.setSize(500,300);
    f.setLocation(100,0);

    f.setVisible(true);
  }
}
```



9 Caselle combinate e caselle di controllo

Le **caselle combinate** (*combo box*) raggruppano un elenco di voci e consentono, tramite un menu a tendina, la scelta di una singola voce. Questo elemento grafico viene implementato in Java usando la classe **JComboBox**. La costruzione di una combo box avviene in due fasi:
- creazione dell'oggetto di classe *JComboBox,*
- aggiunta delle voci all'oggetto.

La creazione viene eseguita attraverso un costruttore che non possiede parametri. Successivamente le voci vengono aggiunte, una ad una, usando il metodo **addItem(*String*)**. Il parametro di questo metodo specifica il testo che deve essere visualizzato all'interno della combo box.

Il seguente frammento di codice crea una casella combinata che contiene l'elenco di quattro colori:

```
JComboBox colori = new JComboBox();
colori.addItem("rosso");
colori.addItem("verde");
colori.addItem("bianco");
colori.addItem("arancione");
```

Si può aggiungere questa componente a una finestra con il metodo **add**.
L'utente può modificare il valore visualizzato in una combo box agendo con il mouse sul corrispondente menu a tendina.

La **casella di controllo** (*check box*) è un elemento grafico che gestisce solo due stati, corrispondenti ai valori booleani *true* e *false*. Se la casella ha il segno di spunta, ad essa si associa il valore booleano *true*, altrimenti si associa il valore *false*.

Questo elemento grafico viene implementato in Java con la classe **JCheckBox** che mette a disposizione i pulsanti a due stati; se l'utente fa un clic sulla casella, ne causa il cambio di stato. Ogni casella di controllo è fornita di un'etichetta che viene usata per spiegare all'utente il significato della casella stessa.

La seguente riga di codice crea una casella di controllo come oggetto di classe *JCheckBox*:

```
JCheckBox c = new JCheckBox("prima scelta");
```

Per *default*, le caselle di controllo sono disattivate e il loro valore è impostato a *false*. Una casella può essere impostata a *true* usando il costruttore che contiene, come secondo parametro, il valore booleano da attribuire ad essa: **JCheckBox(*String, boolean*).**

PROGETTO 5

Presentare un menu di piatti per la scelta dell'utente.

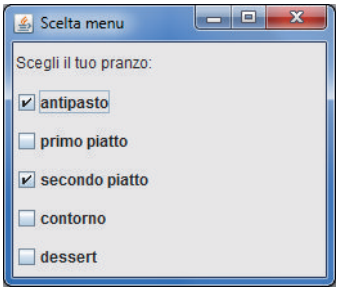
Nel programma viene utilizzata la classe **GridLayout** per facilitare la disposizione (*layout*) delle componenti in modo ordinato all'interno di una griglia (*Grid*).
In particolare, la seguente istruzione imposta per il pannello *p* un layout con una griglia formata da 6 righe e 1 colonna:

```
p.setLayout(new GridLayout(6,1));
```

La spiegazione in dettaglio di questo e di altri *layout* sarà illustrata alla fine di questo paragrafo.

PROGRAMMA JAVA (*Scelte.java*)

```
import java.awt.*;
import javax.swing.*;

class Scelte
{
  public static void main(String argv[])
  {
    JFrame f = new JFrame();
    JPanel p = new JPanel();
    JCheckBox c1 = new JCheckBox("antipasto", true);
    JCheckBox c2 = new JCheckBox("primo piatto");
    JCheckBox c3 = new JCheckBox("secondo piatto", true);
    JCheckBox c4 = new JCheckBox("contorno");
    JCheckBox c5 = new JCheckBox("dessert");

    // dispone gli elementi in una griglia
    p.setLayout(new GridLayout(6,1));

    p.add(new Label("Scegli il tuo pranzo:"));
    p.add(c1);
    p.add(c2);
    p.add(c3);
    p.add(c4);
    p.add(c5);

    f.getContentPane().add(p);

    // esce dal programma quando la finestra viene chiusa
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    f.setTitle("Scelta menu");
    f.setSize(250, 200);
    f.setVisible(true);
  }
}
```



Si osservi che il codice contiene l'istruzione che termina il programma quando l'utente fa clic sul pulsante di chiusura della finestra:

```
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

LAYOUT DEGLI ELEMENTI GRAFICI

In Java, il modo con cui le componenti vengono disposte e posizionate (**layout**) dipende dal contenitore. A ogni contenitore può essere associato un gestore che si preoccupa della disposizione degli elementi. Questo gestore è chiamato **Layout Manager**. Tutti i contenitori hanno un *Layout Manager* di *default*, che può essere comunque cambiato usando un metodo e definendo il nuovo gestore. Il gestore interviene la prima volta che viene visualizzato un contenitore, per disporre le componenti, e successivamente tutte le volte che ne vengono modificate le dimensioni, per riposizionare le componenti. Ogni volta che vengono modificate le dimensioni di una finestra, le componenti vengono riadattate, ingrandendole o riducendole, ma mantenendo lo schema definito dal gestore che è stato scelto.

Java permette di utilizzare diversi gestori. Ognuno dispone gli elementi in un modo particolare e richiede l'uso di un metodo **add** adeguato per aggiungerne di nuovi.

Per assegnare un *Layout Manager* a un contenitore si deve utilizzare il metodo **setLayout**, che riceve come parametro un oggetto che rappresenta il gestore.

Per assegnare a un Frame *f* il gestore **FlowLayout** si usa la seguente istruzione:

```
f.setLayout(new FlowLayout());
```

Tra parentesi viene creato un nuovo oggetto che rappresenta il gestore e viene passato al metodo *setLayout*. L'istruzione precedente è un'abbreviazione che consente di creare l'oggetto e di passarlo contemporaneamente come parametro. Si evita così di creare esplicitamente un oggetto *FlowLayout*. La precedente istruzione corrisponde alle seguenti:

```
FlowLayout gestore = new FlowLayout();
f.setLayout(gestore);
```

Osservando le istruzioni precedenti si vede che il gestore è un oggetto della classe *FlowLayout*.

Di seguito vengono presentati alcuni dei gestori che si possono usare in Java, specificando sia le modalità con cui dispongono gli elementi grafici, sia il metodo *add* da utilizzare per aggiungere le componenti.

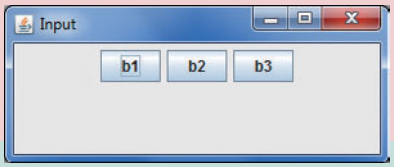
- **FlowLayout**

La classe *FlowLayout* viene usata per disporre le componenti su una stessa riga, una dopo l'altra, nello stesso ordine con cui vengono aggiunte. Quando la riga viene completata, si passa alla riga successiva. Le componenti aggiunte vengono centrate nella riga e sono separate con uno spazio di 5 pixel.
Questo gestore è associato per *default* a tutti i pannelli.
Il metodo **add** che viene usato per aggiungere le componenti a un contenitore di tipo *FlowLayout*, riceve come unico parametro la componente da inserire.

Per esempio, le seguenti istruzioni creano un pannello formato da tre bottoni:

```
JPanel p = new JPanel();
JButton b1 = new JButton("b1");
JButton b2 = new JButton("b2");
JButton b3 = new JButton("b3");
```

```
p.setLayout(new FlowLayout());
p.add(b1);
p.add(b2);
p.add(b3);
```

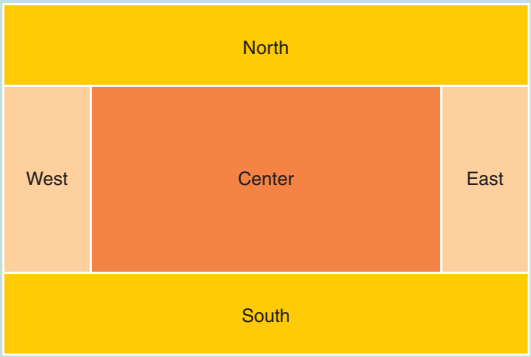


Nella prima parte vengono creati il contenitore *p* e i pulsanti. Viene poi impostato il tipo di *Layout Manager* del pannello e vengono aggiunti, uno ad uno, i pulsanti.
Il costruttore può anche impostare per il *FlowLayout* l'allineamento delle componenti attraverso le **costanti** statiche: *CENTER*, *RIGHT* e *LEFT*.
Per allineare a sinistra le componenti del precedente pannello si deve impostare il layout nel seguente modo:

```
p.setLayout(new FlowLayout(FlowLayout.LEFT));
```

- **BorderLayout**

Usando la classe *BorderLayout*, il contenitore viene diviso in cinque regioni, in ognuna delle quali può essere inserita una componente. Le regioni sono indicate con i seguenti nomi: *North*, *South*, *Center*, *East*, *West*, secondo la disposizione mostrata nella figura seguente.



La zona centrale tende a occupare più spazio possibile riducendo le altre al minimo indispensabile per visualizzare il loro contenuto. Si possono aggiungere le componenti solo ad alcune zone, lasciandone altre vuote. In questo caso il loro spazio sarà utilizzato dalle componenti presenti. Il *BorderLayout* è associato per *default* a tutte le finestre.
Se si vuole applicare questo gestore ai pannelli, occorre usare il metodo *setLayout* nel seguente modo:

```
JPanel p = new JPanel();
p.setLayout(new BorderLayout());
```

Per aggiungere gli elementi al contenitore si usa un metodo **add** diverso da quello utilizzato con il *FlowLayout*.
Il metodo *add* per il *BorderLayout* riceve due parametri, di cui il primo indica la componente che deve essere aggiunta, mentre il secondo indica la zona in cui va aggiunta. La zona è indicata usando una stringa che ne specifica il nome.
Per esempio, l'aggiunta di un bottone *b* nella parte inferiore di una finestra *f* che usa il *BorderLayout* viene eseguita richiamando il metodo *add* nel seguente modo:

```
f.add(b, "South");
```

Questo tipo di disposizione viene usato quando c'è una componente che deve occupare molto spazio, per esempio un'area di testo o un'area di disegno, e che quindi può essere opportunamente aggiunta al contenitore nella zona centrale. Le zone laterali vengono usate per inserire i pannelli che solitamente contengono i comandi dell'applicazione.

- **GridLayout**

La classe *GridLayout* permette di posizionare le componenti in una griglia, come visto nel progetto precedente. Il contenitore viene diviso in tante celle, tutte della stessa dimensione, come una tabella, organizzata in righe e colonne: ogni componente può essere inserita in una cella.

Esistono due costruttori per questo gestore che differiscono per il numero di parametri che ricevono.

GridLayout(*int, int*)
consente di specificare il numero di righe e il numero di colonne che formano la griglia. Per esempio:
GridLayout(5,3) definisce una griglia con 5 righe e 3 colonne.

GridLayout(*int, int, int, int*)
oltre alle dimensioni della griglia, permette di indicare quanto spazio lasciare tra le componenti. Ha quattro parametri: i primi due hanno lo stesso significato del costruttore precedente; il terzo parametro indica quanti pixel separano le componenti orizzontalmente, mentre il quarto parametro indica la spaziatura verticale.

Le componenti sono aggiunte usando un metodo **add** che possiede come unico parametro la componente. Le celle vengono riempite usando le componenti nell'ordine con cui vengono inserite. Prima si riempie la prima riga, da sinistra a destra e poi si passa alle celle sulla seconda riga e così via.

PROGETTO 6

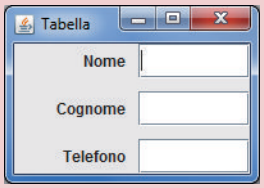
Predisporre un'interfaccia grafica per l'inserimento di un nome, di un cognome e di un numero di telefono.

Questa interfaccia è realizzata usando una tabella avente nella prima colonna le etichette e nella seconda i campi di testo. Essendo tre i valori di input, la griglia è composta da 3 righe e 2 colonne.

PROGRAMMA JAVA (*Tabella.java*)

```
import javax.swing.*;
import java.awt.*;

class Tabella
{
  public static void main(String argv[])
  {
    JFrame f = new JFrame("Tabella");
    JPanel p = new JPanel();
```

```
    JTextField txtNome     = new JTextField(30);
    JTextField txtCognome  = new JTextField(30);
    JTextField txtTelefono = new JTextField(30);

    p.setLayout(new GridLayout(3,2,10,10));
    p.add(new JLabel("Nome", JLabel.RIGHT));
    p.add(txtNome);
    p.add(new JLabel("Cognome", JLabel.RIGHT));
    p.add(txtCognome);
    p.add(new JLabel("Telefono", JLabel.RIGHT));
    p.add(txtTelefono);

    f.getContentPane().add(p);
    f.setSize(200,120);
    f.setLocation(100,100);
    f.setVisible(true);
  }
}
```



AUTOVERIFICA
Problemi da 11 a 14 pag. 347-348

4. La componente per gestire un'area di disegno
5. I contenitori *ScrollPane* e *TabbedPane*

10 Gestione degli eventi

I due aspetti principali dell'interazione con l'utente sono:
- riconoscere quando l'utente compie le azioni
- predisporre le operazioni da eseguire in corrispondenza delle azioni compiute dall'utente.

Il primo è un compito affidato al sistema che gestisce gli eventi. Il secondo è affidato al programmatore che, in base all'applicazione, scrive il codice da eseguire per ogni possibile azione dell'utente. Queste due fasi sono naturalmente collegate: durante l'esecuzione, il gestore degli eventi, in base all'evento che rileva, manda in esecuzione il codice appropriato.

L'obiettivo di chi gestisce un'interfaccia utente è la rapidità di risposta alle azioni dell'utente: nel linguaggio Java gli eventi vengono gestiti solo da chi si registra come **gestore**. Tutti gli eventi che non hanno un gestore vengono ignorati, mentre se un evento si accorge di avere un gestore, ne affida ad esso la gestione.

Questo modello viene chiamato **a delegazione**, perché il programmatore stabilisce, con una delega, chi può essere abilitato a gestire un evento.

Gli eventi sono generati a partire da un particolare oggetto, chiamato **origine**. Per esempio, un pulsante genera un evento quando si fa clic su di esso, una finestra genera un evento quando viene chiusa. Tutte le componenti sono possibili oggetti di origine.

Uno o più **ascoltatori** (*listener*) possono registrarsi nell'oggetto origine per essere avvisati della generazione di un particolare tipo di evento. Gli ascoltatori sono in pratica i gestori degli eventi. L'origine, in presenza di un evento, avverte solo gli ascoltatori che si sono registrati.

Un ascoltatore è un oggetto la cui classe è costruita in un modo particolare. Deve implementare un'interfaccia di tipo **listener**. Un'interfaccia è una classe astratta in cui tutti i metodi sono astratti.
Una classe che implementa un'interfaccia, eredita dall'interfaccia tutti i metodi astratti e, per ognuno, deve fornire un'implementazione.

Per realizzare la gestione degli eventi si deve procedere nel seguente modo:

1) Creare uno o più ascoltatori in base agli eventi che si vogliono gestire.
2) Registrare l'ascoltatore in un oggetto origine che si vuole controllare.
3) Gestire l'evento eseguendo il metodo associato.

1. Creazione degli ascoltatori

Esistono diversi tipi di ascoltatori, ognuno rappresentato da un'interfaccia: l'interfaccia per creare l'ascoltatore delle finestre, dei pulsanti, del mouse e della tastiera. Una classe diventa ascoltatore se implementa un'interfaccia di tipo *listener*.

Per esempio:

```
import java.awt.event.*;

class GestoreFinestra implements WindowListener
{
}
```

La parola chiave **implements** specifica che la classe implementa una particolare interfaccia. Il funzionamento è simile all'ereditarietà. In questo caso l'interfaccia **WindowListener** contiene i metodi astratti che vengono ereditati dalla classe *GestoreFinestra*. Questa, non essendo astratta, deve fornire un'implementazione di tutti i metodi ereditati.

Le interfacce che gestiscono gli eventi sono contenute nel package **java.awt.event** che quindi deve essere importato.

Per completare la creazione dell'ascoltatore bisogna conoscere quali sono i metodi astratti da implementare. In questo caso, l'ascoltatore per la finestra deve possedere i seguenti metodi:

```
public void windowIconified(WindowEvent e) {}
public void windowDeiconified(WindowEvent e) {}
public void windowActivated(WindowEvent e) {}
public void windowDeactivated(WindowEvent e) {}
public void windowOpened(WindowEvent e) {}
public void windowClosed(WindowEvent e) {}
public void windowClosing(WindowEvent e) {}
```

È obbligatorio che tutti i metodi di un ascoltatore siano presenti anche se non contengono alcuna istruzione. All'interno di questi metodi vanno inserite le istruzioni che devono essere eseguite in risposta a un particolare evento. Ogni metodo degli ascoltatori riceve come parametro l'evento che è stato generato.

Per esempio, se il programma deve terminare quando la finestra viene chiusa, si deve sovrascrivere il metodo *windowClosing* nel seguente modo.

```
public void windowClosing(WindowEvent e)
{
  System.exit(0);
}
```

2. Registrazione presso l'origine
Dopo aver predisposto gli ascoltatori, si devono scegliere gli oggetti origine. La scelta ricade sugli oggetti che possono generare un evento e ai quali si vuole associare una risposta. Un oggetto origine è una componente, per esempio una finestra o un pulsante.
Ogni componente possiede i metodi per registrare un particolare ascoltatore.
Il metodo **addWindowListener** è usato dalle finestre per registrare gli ascoltatori di tipo *WindowListener*, che gestiscono gli eventi legati alle finestre.
Il metodo **addActionListener** è usato dai pulsanti per registrare gli ascoltatori che gestiscono gli eventi sui pulsanti.
Supponiamo di aver dichiarato la finestra *f* nel seguente modo:

```
Frame f = new Frame();
```

Se utilizziamo un ascoltatore di classe *GestoreFinestra*, la sua registrazione avviene invocando il metodo *addWindowListener* sull'oggetto finestra:

```
f.addWindowListener(new GestoreFinestra());
```

3. Modalità di esecuzione
Dopo aver spiegato il modo con cui creare un gestore, vediamo in pratica quello che si verifica quando viene generato un evento.

L'utente, interagendo con il programma, può compiere le azioni che generano gli eventi. Ogni evento nasce da una particolare componente, chiamata *origine*. A seguito della nascita di un evento, l'origine cerca un *ascoltatore*, tra quelli che si sono registrati, in grado di gestire quel tipo di evento. Se non ci sono ascoltatori, l'evento viene abbandonato. Se invece l'origine trova un ascoltatore, invoca il metodo associato all'evento.
Dopo aver completato la gestione di un evento, il sistema passa all'evento successivo tra quelli in coda, oppure resta in attesa.

Il seguente progetto mostra come gestire l'evento di **chiusura di una finestra**.

PROGETTO 7

Creare una finestra e registrare il gestore per gli eventi legati alla finestra.

Nell'esecuzione dei programmi presentati in precedenza, le applicazioni grafiche dovevano essere chiuse forzosamente usando la combinazione di tasti *Ctrl + C*. Con la gestione degli eventi si può trattare questa situazione in modo più corretto.
L'evento da gestire è la chiusura di una finestra: quando l'utente preme il tasto di chiusura, genera un evento che deve essere gestito per terminare l'applicazione.

Il *GestoreFinestra* è impostato per stampare un messaggio e poi terminare il programma.

IMPLEMENTAZIONE DELLA CLASSE (*GestoreFinestra.java*)

```
import java.awt.event.*;

class GestoreFinestra implements WindowListener
{
  public void windowIconified(WindowEvent e) {}
  public void windowDeiconified(WindowEvent e) {}
  public void windowActivated(WindowEvent e) {}
  public void windowDeactivated(WindowEvent e) {}
  public void windowOpened(WindowEvent e) {}
  public void windowClosed(WindowEvent e) {}
  public void windowClosing(WindowEvent e)
  {
    System.out.println("Programma terminato.");
    System.exit(0);
  }
}
```

PROGRAMMA JAVA (*Vuota.java*)

```
import javax.swing.*;
import java.awt.*;

class Vuota
{
  public static void main(String argv[])
  {
    JFrame f = new JFrame("Finestra vuota");

    f.setSize(200,200);
    f.addWindowListener(new GestoreFinestra());

    f.setVisible(true);
  }
}
```

L'interfaccia *WindowListener* contiene i metodi che vengono attivati a seguito di particolari eventi. Nel progetto precedente ci sono metodi che non contengono alcuna istruzione. Nella seguente tabella sono indicati, per ogni evento, i metodi che vengono eseguiti.

Evento	Metodo eseguito
Quando la finestra viene ridotta ad icona	WindowIconified
Quando la finestra viene ingrandita	WindowDeiconified
Quando la finestra riceve lo stato attivo	WindowActivated
Quando la finestra perde lo stato attivo	WindowDeactivated
Quando la finestra viene aperta	WindowOpened
Quando la finestra è stata chiusa	WindowClosed
Quando la finestra si sta chiudendo	WindowClosing

Il seguente progetto mostra come gestire l'evento di **clic su un pulsante**.

PROGETTO 8

Creare un'interfaccia con due pulsanti e un'area di testo, gestendo gli eventi della finestra e dei pulsanti.

Disegno dell'interfaccia grafica
La finestra è organizzata con un *BorderLayout*. Nella zona in alto e in basso vengono posizionati due pulsanti che rappresentano rispettivamente il pulsante superiore e quello inferiore. Nella zona centrale viene collocata l'area di testo, impostata come non editabile, dentro la quale sono visualizzati i messaggi.

Gestione degli eventi
L'unico evento disponibile per i pulsanti è la pressione su di essi con il tasto del mouse. Un ascoltatore, per questo evento, deve implementare l'interfaccia **ActionListener**. Questa interfaccia contiene un solo metodo che deve essere ridefinito e che si chiama **actionPerformed**.
Quando un pulsante viene premuto, genera un evento. Se è stato registrato un ascoltatore, viene invocato il metodo *actionPerformed* contenuto nell'ascoltatore.
Lo stesso ascoltatore può essere registrato da più pulsanti. Questo significa che viene eseguito lo stesso metodo ogni volta che un pulsante viene premuto. Per discriminare il pulsante e quindi le diverse operazioni da eseguire, all'interno del metodo *actionPerformed* viene eseguito un controllo per stabilire chi ha generato l'evento.
Il metodo *actionPerformed*, riceve come parametro un oggetto di classe **ActionEvent**. Per acquisire l'indicazione dell'oggetto che ha generato l'evento si usa il metodo **getActionCommand()**. Questo metodo restituisce una stringa che rappresenta il testo mostrato dal pulsante. Confrontando questa scritta con i pulsanti che si vogliono gestire, si può stabilire chi ha generato l'evento e quindi eseguire le opportune operazioni.

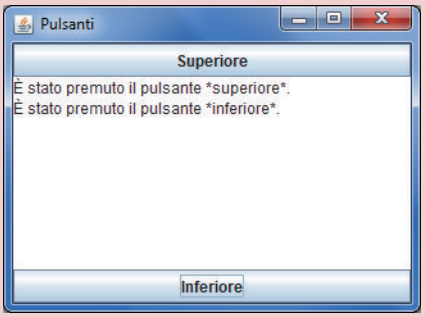
IMPLEMENTAZIONE DELLA CLASSE (*GestorePulsante.java*)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class GestorePulsante implements ActionListener
{
  private JTextArea a;

  public GestorePulsante(JTextArea a)
  {
    this.a = a;
  }

  public void actionPerformed(ActionEvent e)
  {
    String pulsante = e.getActionCommand();

    if (pulsante.equals("Superiore"))
    {
      a.append("È stato premuto il pulsante *superiore*.\n");
    }
```

```
    if (pulsante.equals("Inferiore"))
    {
      a.append("È stato premuto il pulsante *inferiore*.\n");
    }
  }
}
```

PROGRAMMA JAVA (*Pulsanti.java*)

```
import javax.swing.*;
import java.awt.*;

class Pulsanti
{
  public static void main(String argv[])
  {
    JFrame f = new JFrame("Pulsanti");
    JPanel p = new JPanel();
    JButton sup = new JButton("Superiore");
    JButton inf = new JButton("Inferiore");
    JTextArea a = new JTextArea(50,10);

    p.setLayout(new BorderLayout());
    p.add(sup, "North");
    p.add(a, "Center");
    p.add(inf, "South");

    f.addWindowListener(new GestoreFinestra());
    sup.addActionListener(new GestorePulsante (a));
    inf.addActionListener(new GestorePulsante(a));

    a.setEditable(false);

    f.getContentPane().add(p);
    f.setSize(450,300);
    f.setVisible(true);
  }
}
```



La classe *GestoreBottone* ha un costruttore che riceve come parametro una *JTextArea*. Questa è un riferimento all'area di testo che deve essere gestita dalla pressione dei pulsanti. Essa è passata come parametro al gestore dei pulsanti, perché altrimenti non saprebbe come scrivere all'interno dell'area di testo.
Il messaggio all'interno dell'area di testo viene stampato usando il metodo **append*(String)***.
Per la gestione della finestra, viene usata la medesima classe *GestoreFinestra* definita nel progetto precedente.

Il seguente progetto mostra come gestire la **lettura da una casella di testo** a seguito di un evento.

PROGETTO 9

Realizzare, con un'interfaccia grafica, un convertitore tra gradi centigradi e fahrenheit.

La formula di conversione è: fahrenheit = 32+(centigradi/100)*180.

Disegno dell'interfaccia grafica
La finestra è organizzata con un *GridLayout*: la griglia è composta da tre celle disposte verticalmente. Nella prima si inserisce un pannello per i gradi centigradi contenente un'etichetta e una casella di testo. Nella seconda viene posizionato un pulsante per eseguire la conversione. Nella terza si inserisce un altro pannello per la misura in fahrenheit con un'etichetta e una casella di testo.
La tabella seguente riassume i nomi degli oggetti utilizzati nell'interfaccia grafica con la relativa classe.

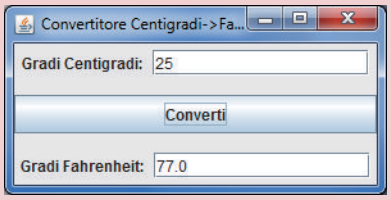
Classe	Nome oggetto
JFrame	f
JPanel	p1
JPanel	p2
JTextField	txtCentigradi
JTextField	txtFahrenheit
JButton	btnConverti

Gestione degli eventi
La lettura del contenuto di una casella di testo, come già detto, è realizzata dal metodo **getText()**. La lettura, però, deve essere fatta dopo che l'utente ha inserito il valore. Si può usare un pulsante con il quale l'utente possa confermare l'avvenuto inserimento dei dati. In questo modo la lettura della casella di testo avviene all'interno del gestore del pulsante.

IMPLEMENTAZIONE DELLA CLASSE (*ConvertFrame.java*)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class ConvertFrame extends JFrame implements ActionListener
{
  private JPanel p1 = new JPanel();
  private JPanel p2 = new JPanel();
  private JTextField txtCentigradi = new JTextField(15);
  private JTextField txtFahrenheit = new JTextField(15);
  private JButton btnConverti = new JButton("Converti");

  public ConvertFrame()
  {
    super("Convertitore Centigradi->Fahrenheit");

    addWindowListener(new GestoreFinestra());
```

```
    // inserisce le componenti nei pannelli
    p1.add(new JLabel("Gradi Centigradi: "));
    p1.add(txtCentigradi);
    p2.add(new JLabel("Gradi Fahrenheit: "));
    p2.add(txtFahrenheit);

    // inserisce le componenti nella finestra disponendole con una griglia
    setLayout(new GridLayout(3,1,5,10));
    add(p1);
    add(btnConverti);
    add(p2);
    btnConverti.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e)
  {
    String pulsante = e.getActionCommand();
    double cent, fahr;

    if (pulsante.equals("Converti"))
    {
      try
      {
        String numeroLetto = txtCentigradi.getText();
        cent = Double.valueOf(numeroLetto).doubleValue();
        fahr = 32+(cent/100)*180;
        txtFahrenheit.setText(""+fahr);
      }
      catch(Exception exc)
      {
        txtFahrenheit.setText("");
      }
    }
  }
}
```



PROGRAMMA JAVA (*Convertitore.java*)

```
class Convertitore
{
  public static void main(String argv[])
  {
    ConvertFrame f = new ConvertFrame();

    f.pack();
    f.setVisible(true);
  }
}
```

La classe *Convertitore* è una classe semplice contenente solo il *main*. Il suo compito è creare una finestra di classe *ConvertFrame* e di visualizzarla. Il metodo **pack()** impone che le dimensioni della finestra siano le più piccole possibili, sufficienti a contenere tutte le componenti.
La classe *ConvertFrame* è una sottoclasse di un *JFrame* e quindi è una finestra. Contiene come attributi privati tutti gli elementi grafici che vengono usati all'interno della finestra. Il suo costruttore posiziona le componenti e registra il gestore della finestra e quello del pulsante. Essendo una finestra, possono essere usati i metodi ereditati dalla classe *JFrame* senza usare la notazione con il punto. Ne sono un esempio i metodi *addWindowListener*, *setLayout* e *add*.
Il gestore del pulsante è implementato dalla stessa classe *ConvertFrame*. Questo viene fatto usando la parola chiave *implements* nella definizione della classe.
Non è quindi necessario che un ascoltatore sia una classe a sé stante, ma può essere inserito in un'altra classe, basta che vengano ridefiniti tutti i metodi di quella interfaccia. In questo caso, l'interfaccia è la *ActionListener* e l'unico metodo che viene ridefinito è *actionPerformed*.
Per registrare questo gestore nella componente *btnConverti*, corrispondente al pulsante, è stata usata la seguente istruzione:

```
btnConverti.addActionListener(this);
```

Questo significa che l'ascoltatore è **this**, cioè l'oggetto stesso che contiene questa istruzione. Raggruppare un ascoltatore all'interno della classe in cui viene usato è utile, perché l'ascoltatore può accedere liberamente a tutte le componenti dell'interfaccia.
Nel programma precedente, l'ascoltatore accede in lettura alla casella di testo *txtCentigradi* e in scrittura alla casella di testo *txtFahrenheit*. Anche in questo programma viene usata la classe *GestoreFinestra* definita nei progetti precedenti.

Il seguente progetto mostra come gestire la **lettura da una combo box** a seguito di un evento.

PROGETTO 10

Offrire all'utente la possibilità, tramite una casella combinata, di modificare il colore di sfondo di un'area.

Disegno dell'interfaccia grafica
La finestra è organizzata con un *BorderLayout*. Nella zona in alto si dispone un pannello contenente un'etichetta, una combo box per la scelta e un pulsante per effettuare il cambio di colore. La zona centrale, la più ampia della finestra, è occupata da un pannello impostato inizialmente con lo sfondo bianco.
La tabella seguente riassume i nomi degli oggetti utilizzati nell'interfaccia grafica con la relativa classe.

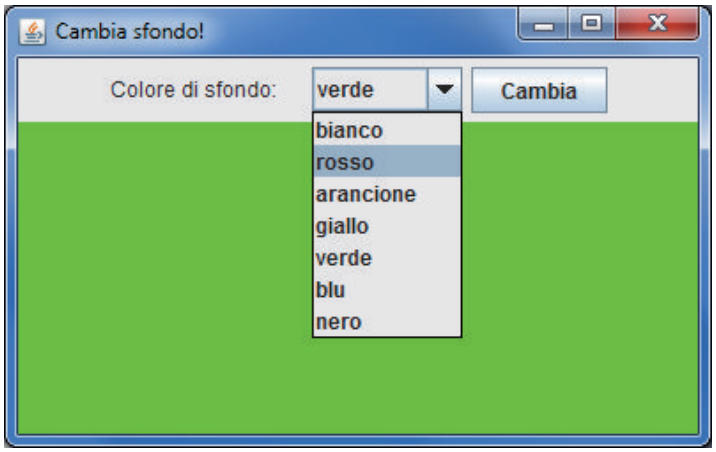
Classe	Nome oggetto
JFrame	f
JPanel	p
JPanel	panArea
JComboBox	cbColori
JButton	btnCambia

Gestione degli eventi
Per la lettura da una casella combinata si adotta lo stesso procedimento usato con le caselle di testo. Un utente può cambiare il valore della casella combinata e inviare al programma questo valore facendo clic su un pulsante. Perciò la lettura del valore della combo box è inserito nel gestore del pulsante. Viene utilizzato il metodo **getSelectedIndex()** che restituisce un numero corrispondente all'indice della voce selezionata. Le voci sono numerate in base all'ordine con cui sono state aggiunte alla combo box in fase di creazione. La prima voce ha indice 0, la seconda indice 1 e così via.
La classe *GestoreFinestra* è la stessa utilizzata nei progetti precedenti.

IMPLEMENTAZIONE DELLA CLASSE (*CambiaFrame.java*)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class CambiaFrame extends Frame implements ActionListener
{
  private Color sfondo       = Color.white;
  private JPanel p           = new JPanel();
  private JPanel panArea     = new JPanel ();
  private JComboBox cbColori = new JComboBox();
  private JButton btnCambia  = new JButton("Cambia");

  public CambiaFrame()
  {
    super("Cambia sfondo!");

    setSize(400,250);
    addWindowListener(new GestoreFinestra());

    inizializzaCombo();

    panArea.setBackground(sfondo);

    // inserisce le componenti nel pannello in alto
    p.add(new Label("Colore di sfondo: "));
    p.add(cbColori);
    p.add(btnCambia);

    // inserisce le componenti nella finestra
    add(p, "North");
    add(panArea, "Center");
    btnCambia.addActionListener(this);
  }

  private void inizializzaCombo()
  {
    // aggiunge le voci alla combo-box
    cbColori.addItem("bianco");
    cbColori.addItem("rosso");
    cbColori.addItem("arancione");
    cbColori.addItem("giallo");
    cbColori.addItem("verde");
    cbColori.addItem("blu");
    cbColori.addItem("nero");
  }
```

```
  public void actionPerformed(ActionEvent e)
  {
    String pulsante = e.getActionCommand();

    if (pulsante.equals("Cambia"))
    {
      switch (cbColori.getSelectedIndex())
      {
        case 0: sfondo = Color.white;  break;
        case 1: sfondo = Color.red;    break;
        case 2: sfondo = Color.orange; break;
        case 3: sfondo = Color.yellow; break;
        case 4: sfondo = Color.green;  break;
        case 5: sfondo = Color.blue;   break;
        case 6: sfondo = Color.black;  break;
      }

      panArea.setBackground(sfondo);
    }
  }
}
```



PROGRAMMA JAVA (*Cambia.java*)

```
class Cambia
{
  public static void main(String argv[])
  {
    CambiaFrame f = new CambiaFrame();

    f.setVisible(true);
  }
}
```

11 Finestre di dialogo

Le interfacce grafiche utilizzano le **finestre di dialogo** per inviare all'utente messaggi di avvertimento o di errore oppure la richiesta di inserimento dati o di conferma all'esecuzione delle operazioni. Ci sono poi le finestre di dialogo tipiche per l'apertura e il salvataggio dei file, per la scelta dei *font* di caratteri oppure per la selezione dei colori.
La componente Swing, che rappresenta le finestre di dialogo standard, si chiama **JOptionPane**: essa produce una finestra di *popup* sul video per comunicare un messaggio all'utente o richiedere un dato. I metodi utilizzati da questa classe sono:

Metodo	Descrizione
showConfirmDialog	Chiede una conferma all'utente con possibilità di risposta *Si*, *No*, *Annulla*
showInputDialog	Fornisce un prompt per l'inserimento di dati
showMessageDialog	Comunica un messaggio all'utente
showOptionDialog	Visualizza una finestra di carattere generale che unifica le funzionalità delle tre precedenti

Per esempio l'istruzione seguente produce sul video la finestra di dialogo in figura:

```
JOptionPane.showMessageDialog(null,
   "Dati registrati",
   "Inserimento",
   JOptionPane.INFORMATION_MESSAGE);
```

Il primo parametro indica il frame principale che può essere *this* o *null*:
- con **this** il frame principale rimane bloccato fino alla risposta dell'utente alla finestra di dialogo
- con **null** la finestra di dialogo viene visualizzata al centro dello schermo ed è indipendente dalla finestra principale.

Il secondo parametro è il messaggio che compare al centro della finestra di dialogo, mentre il terzo parametro è il titolo della finestra.
L'ultimo parametro specifica il tipo di icona da visualizzare nella parte sinistra della finestra.
Le icone standard sono rappresentate con i seguenti valori per il parametro:

 **JOptionPane.ERROR_MESSAGE**

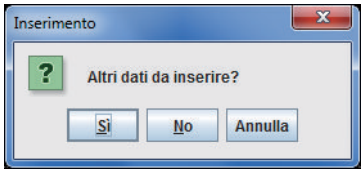
 **JOptionPane.INFORMATION_MESSAGE**

 **JOptionPane.WARNING_MESSAGE**

 **JOptionPane.QUESTION_MESSAGE**

JOptionPane.PLAIN_MESSAGE (non visualizza alcuna icona)

Si osservi che i nomi delle icone sono scritti tutto in maiuscolo.
Come si vede dalla figura della finestra precedente, viene aggiunto automaticamente il pulsante *OK*.
I pulsanti possono essere impostati in modo diverso da questo di *default* con un ulteriore parametro, da inserire dopo il titolo della finestra, che può assumere i seguenti valori:
JOptionPane.DEFAULT_OPTION (pulsante *OK*)
JOptionPane.YES_NO_OPTION (pulsanti *Sì*, *No*)
JOptionPane.YES_NO_CANCEL_OPTION (pulsanti *Sì*, *No*, *Annulla*)
JOptionPane.OK_CANCEL_OPTION (pulsanti *OK*, *Annulla*).



Per esempio per visualizzare la finestra di dialogo della figura, occorre scrivere la seguente istruzione:

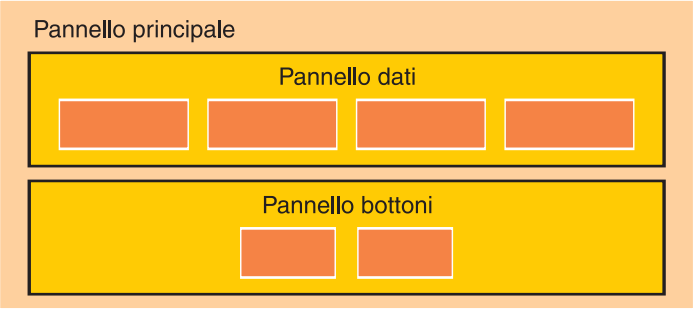
```
JOptionPane.showConfirmDialog(null,
   "Altri dati da inserire?",
   "Inserimento",
   JOptionPane.YES_NO_CANCEL_OPTION,
   JOptionPane.QUESTION_MESSAGE);
```

PROGETTO 11

Scrivere il programma per richiedere all'utente il cognome e il nome.

I dati da inserire devono essere considerati obbligatori, per cui se l'utente lascia una delle caselle di input vuote, il programma visualizza una finestra con un messaggio di avvertimento, altrimenti visualizza una finestra di informazione che riassume i dati inseriti.

Disegno dell'interfaccia grafica
La finestra dell'interfaccia utente contiene un pannello principale che a sua volta organizza le componenti in due pannelli secondari: il primo contiene due etichette e due caselle di testo per il cognome e il nome, il secondo contiene i pulsanti *Annulla* e *OK*.



La tabella seguente riassume i nomi degli oggetti utilizzati nell'interfaccia grafica con la relativa classe.

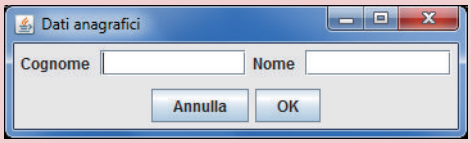
Classe	Nome oggetto
JFrame	f
JPanel	pan
JPanel	panDati
JLabel	lblCognome
JTextField	txtCognome
JLabel	lblNome
JTextField	txtNome
JPanel	panPulsanti
JButton	btnAnnulla
JButton	btnOk

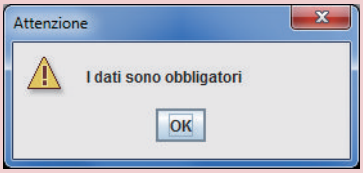
Gestione degli eventi
All'inizio dell'esecuzione le caselle di testo sono vuote. L'utente inserisce i dati di una persona. Quando fa clic sul pulsante *OK*, il programma controlla che le due caselle di testo non siano vuote. In base al risultato del controllo mostra una diversa finestra di dialogo. Se l'utente fa clic sul pulsante *Annulla*, le caselle di testo vengono svuotate.

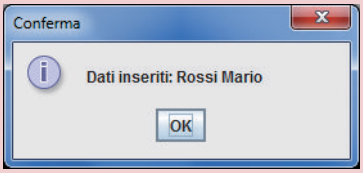
IMPLEMENTAZIONE DELLA CLASSE (*Anagrafe.java*)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Anagrafe extends JFrame implements ActionListener
{
  private JPanel pan = new JPanel();

  private JPanel panDati = new JPanel();
  private JLabel lblCognome = new JLabel ("Cognome ", JLabel.RIGHT);
  private JTextField txtCognome = new JTextField(10);
  private JLabel lblNome = new JLabel ("Nome ", JLabel.RIGHT);
  private JTextField txtNome = new JTextField(10);

  private JPanel panPulsanti = new JPanel();
  private JButton btnAnnulla = new JButton("Annulla");
  private JButton btnOk = new JButton("OK");

  public Anagrafe()
  {
    pan.setLayout(new BorderLayout());

    // imposta le stringhe di comando e gli ascoltatori per gli eventi
    // sui pulsanti di comando
    btnAnnulla.setActionCommand("annulla");
    btnOk.setActionCommand("ok");
    btnAnnulla.addActionListener(this);
    btnOk.addActionListener(this);
```

```
    // inserisce le componenti nei pannelli
    panDati.add(lblCognome);
    panDati.add(txtCognome);
    panDati.add(lblNome);
    panDati.add(txtNome);
    panPulsanti.add(btnAnnulla);
    panPulsanti.add(btnOk);
    pan.add(panDati, "Center");
    pan.add(panPulsanti,"South");

    // aggiunge il pannello principale al frame
    this.getContentPane().add(pan, BorderLayout.CENTER);
    // controlla l'uscita dal programma quando la finestra viene chiusa
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }

  public void actionPerformed(ActionEvent e)
  {
    if ("annulla".equals(e.getActionCommand()))
    {
      txtCognome.setText("");
      txtNome.setText("");
    }
    if ("ok".equals(e.getActionCommand()))
    {
      String cognomeLetto = txtCognome.getText();
      String nomeLetto = txtNome.getText();
      if ((cognomeLetto.equals("")) || (nomeLetto.equals("")))
      {
        JOptionPane.showMessageDialog(this,
            "I dati sono obbligatori",
            "Attenzione",
            JOptionPane.WARNING_MESSAGE);
      }
      else
      {
        String messaggio = "Dati inseriti: " + cognomeLetto + " " + nomeLetto;
        JOptionPane.showMessageDialog(this,
            messaggio,
            "Conferma",
            JOptionPane.INFORMATION_MESSAGE);
      }
    }
  }
}
```







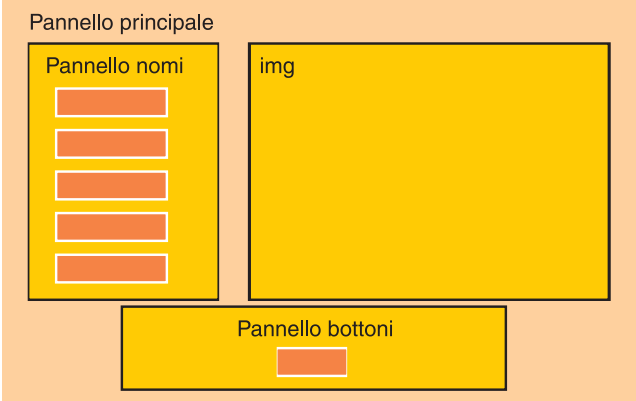
PROGRAMMA JAVA (*DatiAnagrafici.java*)

```
import javax.swing.*;

class DatiAnagrafici
{
  public static void main(String argv[])
  {
    Anagrafe f = new Anagrafe();

    f.setTitle("Dati anagrafici");
    f.pack();
    f.setVisible(true);
  }
}
```

PROGETTO 12

Scrivere il programma per consentire all'utente la scelta in un elenco di città italiane. In corrispondenza della città selezionata, viene visualizzata una fotografia della città. Si deve anche controllare l'uscita dal programma chiedendone conferma all'utente.

Disegno dell'interfaccia grafica
La finestra dell'interfaccia utente contiene un pannello principale contenente, a sua volta, un pannello per i nomi delle città nella parte sinistra, l'immagine della città al centro e il pannello per il pulsante *Uscita* in basso. L'elenco delle città è organizzato con un gruppo di pulsanti di opzione (*JRadioButton*), in modo tale che si possa selezionare una sola città per volta. Le immagini delle città sono registrate in una sottodirectory *images* della directory dove si trova l'applicazione: hanno come nome di file il nome della città ed estensione *.jpg*.



Il layout assegnato al pannello principale è un *BorderLayout*, mentre il pannello dei nomi ha un *GridLayout* con 5 righe e 1 colonna.
La componente **JRadioButton** visualizza un piccolo cerchio, con accanto una descrizione: esso viene annerito quando l'utente fa clic su di esso. Per consentire una sola scelta possibile, i diversi *JRadioButton* devono essere assegnati ad un **ButtonGroup** (gruppo di pulsanti) con il metodo **add**.

Per una migliore gestione dei dati, ma anche per una maggiore chiarezza del codice, i nomi delle città sono organizzati in un array: il nome della città, infatti, coincide con il nome del pulsante, con il nome dell'immagine e con il nome assegnato al comando di azione per la gestione degli eventi sul *JRadioButton*.
L'immagine della città è contenuta in una componente **JLabel**: l'immagine viene associata alla *JLabel* attraverso il metodo **setIcon**.

La tabella seguente riassume i nomi degli oggetti utilizzati nell'interfaccia grafica con la relativa classe.

Classe	Nome oggetto
JFrame	f
JPanel	pan
JPanel	panNomi
JRadioButton	rbCitta
ButtonGroup	bg
JLabel	imgCitta
JPanel	panPulsanti
JButton	btnUscita

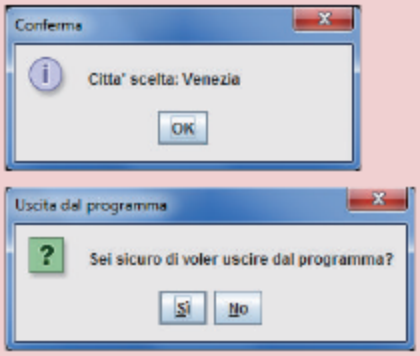
Gestione degli eventi
L'utente interagisce con il programma facendo clic sui pulsanti di opzione. A seguito di un clic, viene mostrata una fotografia della città e al centro del video compare una finestra di dialogo che visualizza il nome della città scelta. L'utente può uscire dal programma facendo clic sul pulsante di *Uscita* che attiva una finestra di dialogo per chiedere la conferma.
L'utente ha dunque due possibilità per uscire dal programma: con il pulsante di chiusura in alto a destra della finestra oppure con il pulsante *Uscita*.

IMPLEMENTAZIONE DELLA CLASSE (*Immagini.java*)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Immagini extends JFrame implements ActionListener
{
  static String arrCitta[] = new String[5];
  private JPanel pan = new JPanel();
  private JPanel panNomi = new JPanel();
  private JRadioButton rbCitta[] = new JRadioButton[5];
  private ButtonGroup bg = new ButtonGroup();
  private JLabel imgCitta = new JLabel();
  private JPanel panPulsanti = new JPanel();
  private JButton btnUscita = new JButton("Uscita");
```

```
public Immagini()
{
  arrCitta[0] = "Milano";
  arrCitta[1] = "Roma";
  arrCitta[2] = "Napoli";
  arrCitta[3] = "Venezia";
  arrCitta[4] = "Firenze";

  pan.setLayout(new BorderLayout());
  panNomi.setLayout(new GridLayout(5,1));

  // crea i pulsanti di opzione, li aggiunge al gruppo
  // e imposta la stringa di comando per l'evento sul pulsante
  for (int i=0; i<5; i++)
  {
    rbCitta[i] = new JRadioButton(arrCitta[i]);
    bg.add(rbCitta[i]);
    rbCitta[i].setActionCommand(arrCitta[i]);
  }

  // imposta la prima opzione come preselezionata
  rbCitta[0].setSelected(true);
  imgCitta.setIcon(new ImageIcon("images/" + arrCitta[0] + ".jpg"));

  // imposta la stringa di comando per l'evento sul pulsante di uscita
  btnUscita.setActionCommand("uscita");

  // registra gli ascoltatori per i pulsanti
  for (int i=0; i<5; i++) rbCitta[i].addActionListener(this);
  btnUscita.addActionListener(this);

  // inserisce le componenti nei pannelli
  for (int i=0; i<5; i++) panNomi.add(rbCitta[i]);
  panPulsanti.add(btnUscita);
  pan.add(panNomi, "West");
  pan.add(imgCitta, "Center");
  pan.add(panPulsanti,"South");

  // aggiunge il pannello al frame
  this.getContentPane().add(pan, BorderLayout.CENTER);
  // controlla l'uscita dal programma quando la finestra viene chiusa
  this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

// gestione degli eventi
public void actionPerformed(ActionEvent e)
{
  if ("uscita".equals(e.getActionCommand()))
  {
```

```
      int n = JOptionPane.showConfirmDialog(null,
              "Sei sicuro di voler uscire dal programma?",
              "Uscita dal programma",
              JOptionPane.YES_NO_OPTION);
      if (n == JOptionPane.YES_OPTION)
      {
       System.exit(0);
      }
    }
    else
    {
      imgCitta.setIcon(new ImageIcon("images/" + e.getActionCommand() + ".jpg"));
      String messaggio = "Citta' scelta: " + e.getActionCommand();
      JOptionPane.showMessageDialog(null,
        messaggio,
        "Conferma",
        JOptionPane.INFORMATION_MESSAGE);
    }
  }
}
```



PROGRAMMA JAVA (*CittaItaliane.java*)

```
class CittaItaliane
{
  public static void main(String argv[])
  {
    Immagini f = new Immagini();

    f.setTitle("Citta' Italiane");
    f.setSize(250, 200);
    f.setVisible(true);
  }
}
```

AUTOVERIFICA
Problemi da 15 a 21 pag. 348

6. Evento sulla modifica delle caselle di testo

12 I menu

I menu con Swing sono realizzati dalle componenti:
- **JMenuBar** per la barra dei menu,
- **JMenu** per i menu a tendina inseriti nella barra,
- **JMenuItem** per le voci contenute nei menu.

La barra dei menu, per convenzione, si trova all'esterno del *content pane* che contiene le componenti grafiche della finestra ed è posizionata in alto nella finestra stessa.
Ogni barra può contenere uno o più menu a tendina e ogni menu può contenere una o più voci, oltre alle linee di separazione tra gruppi di voci. Il menu viene aggiunto alla barra con il metodo **add**. Anche la voce viene aggiunta al menu con il metodo **add**.
La barra dei menu viene assegnata al frame con il metodo **setJMenuBar**.

Le seguenti linee di codice illustrano le modalità per costruire il menu *File* e per inserire in esso la voce di menu *Nuovo*:

```
JMenuBar barra;
JMenu menu;
JMenuItem voce;

menu = new JMenu("File");
barra.add(menu);
voce = new JMenuItem("Nuovo")
menu.add(voce);
```

Per aggiungere una linea di separazione tra gruppi di voci, nello stesso menu, si usa il metodo **addSeparator**.

```
menu.addSeparator();
```

Nelle interfacce dei software spesso ai menu e alle voci di menu vengono associati caratteri alfabetici, mnemonici, che corrispondono al tasto da premere insieme al tasto **Alt** per selezionare il menu o la voce con una scorciatoia da tastiera (*shortcut*). La lettera può essere una tra quelle che compongono la descrizione del menu o della voce, ma deve essere univoca all'interno dello stesso menu. La lettera scelta risulta sottolineata nella visualizzazione del menu.
Per esempio le istruzioni seguenti impostano la lettera F come mnemonica per il menu e la lettera N per la voce di menu:

```
menu.setMnemonic(KeyEvent.VK_F);
voce.setMnemonic(KeyEvent.VK_N);
```

Le lettere sono rappresentate con le costanti **VK**_*lettera*. Non viene fatta alcuna distinzione tra maiuscole e minuscole.

Per le componenti di tipo *JMenuItem* si può usare una forma più compatta, inserendo la specificazione della lettera come secondo parametro, direttamente nell'istruzione di creazione della voce:

```
voce = new JMenuItem("Nuovo", KeyEvent.VK_N);
```

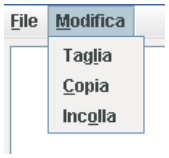
PROGETTO 13

Costruire un'applicazione per l'editing di testi che consenta all'utente di scegliere le funzioni del programma all'interno della barra dei menu.

Il programma simula un editor di testi con le funzionalità per creare un nuovo testo, per salvare su disco il testo e per aprire un testo già esistente. Sono implementate anche le funzioni di *Taglia*, *Copia* e *Incolla*, tipiche nell'editing dei testi.

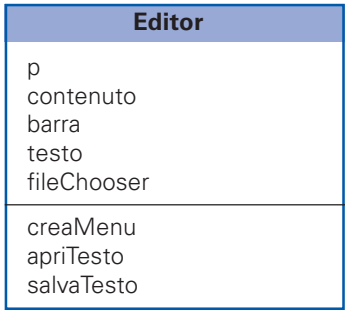
Disegno dell'interfaccia grafica
La finestra dell'applicazione contiene la barra dei menu e un pannello di tipo **JScrollPane** che fornisce un riquadro il cui contenuto può essere letto attraverso le barre di scorrimento orizzontali e verticali. Le barre di scorrimento sono visualizzate automaticamente quando le dimensioni del contenuto superano le dimensioni del riquadro. Il testo è rappresentato con una componente **JTextArea**, che è un'area di testo multilinea per visualizzare testo senza evidenziazioni.
La struttura della barra dei menu è illustrata dalla seguente tabella:

Menu	Voce	Tasto mnemonico
File		F
	Nuovo	N
	Apri	R
	Salva	S
	separatore	
	Esci	E
Modifica		M
	Taglia	L
	Copia	C
	Incolla	O



Per le operazioni di *Apri* e *Salva*, si utilizza una componente **JFileChooser**: esso è lo strumento che consente di navigare nelle directory e sottodirectory del disco per selezionare un file. I metodi usati nel programma sono:
- **showOpenDialog** per l'apertura di un file esistente
- **showSaveDialog** per il salvataggio su disco.

Questi metodi provocano la visualizzazione delle finestre di dialogo *Apri* e *Salva*, tipiche dei software con interfaccia grafica.
L'applicazione si basa sulla classe *Editor* che viene creata come sottoclasse di *JFrame* e possiede gli attributi e i metodi indicati nel seguente diagramma:



Gestione degli eventi
L'utente interagisce con questo programma azionando le voci della barra dei menu. Il controllo sulla voce di menu selezionata è realizzato con il metodo **getSource** per determinare la sorgente dell'evento generato:

```
JMenuItem source = (JMenuItem)(e.getSource());
String s = source.getText();
```

Il metodo **getText** restituisce la descrizione della sorgente dell'evento, che corrisponde alla descrizione della voce di menu. In base alla voce di menu scelta, viene attivata l'esecuzione del metodo delegato alla sua gestione.

IMPLEMENTAZIONE DELLA CLASSE (*Editor.java*)

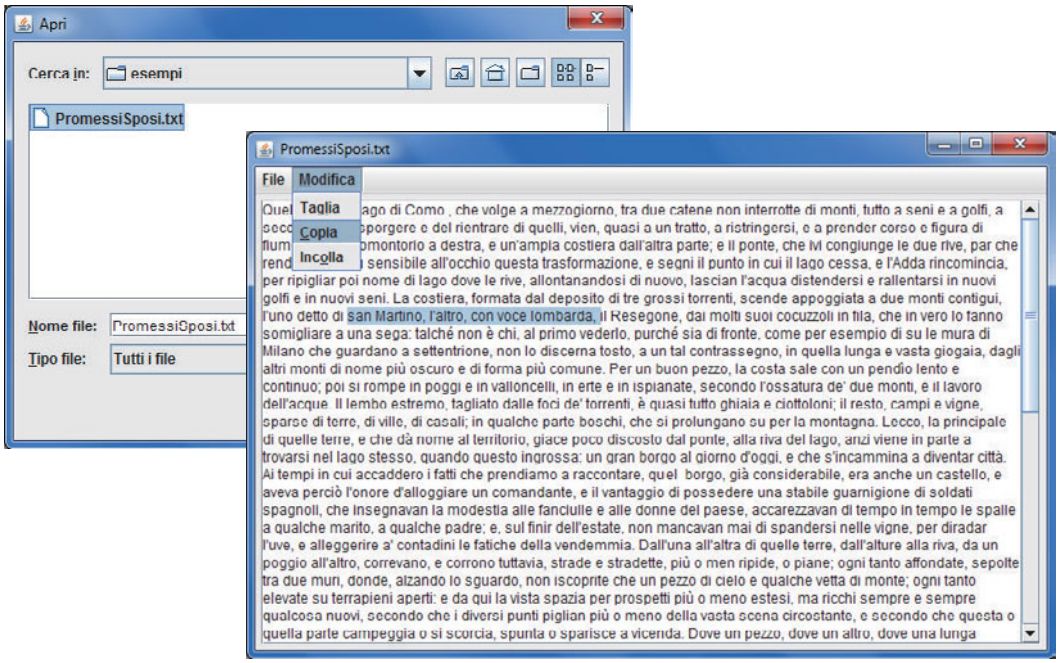
```
import javax.swing.*;
import javax.swing.text.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class Editor extends JFrame implements ActionListener
{
  private JPanel p = new JPanel();
  private JScrollPane contenuto = new JScrollPane();
  private JMenuBar barra = new JMenuBar();
  private JTextArea testo = new JTextArea();
  private JFileChooser fileChooser = new JFileChooser();

  public Editor()
  {
    // crea la barra dei menu
    creaMenu();
    this.setJMenuBar(barra);

    // imposta le caratteristiche dell'area di testo
    testo = new JTextArea(25, 60);
    testo.setEditable(true);
    testo.setFont(new Font("Arial", Font.PLAIN, 12));
    testo.setLineWrap(true);
    testo.setWrapStyleWord(true);

    // aggiunge le componenti del frame
    contenuto = new JScrollPane(testo);
    p.add(contenuto);
    this.getContentPane().add(p, BorderLayout.CENTER);

    // controlla l'uscita dal programma quando la finestra viene chiusa
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
```

```
// creazione delle voci di menu
public JMenuBar creaMenu()
{
  JMenu menu;
  JMenuItem voce;

  // menu File
  menu = new JMenu("File");
  menu.setMnemonic(KeyEvent.VK_F);
  barra.add(menu);

  // voci
  voce = new JMenuItem("Nuovo", KeyEvent.VK_N);
  voce.addActionListener(this);
  menu.add(voce);

  voce = new JMenuItem("Apri", KeyEvent.VK_R);
  voce.addActionListener(this);
  voce.setActionCommand("apri");
  menu.add(voce);

  voce = new JMenuItem("Salva", KeyEvent.VK_S);
  voce.addActionListener(this);
  menu.add(voce);

  menu.addSeparator();

  voce = new JMenuItem("Esci", KeyEvent.VK_E);
  voce.addActionListener(this);
  voce.setActionCommand("esci");
  menu.add(voce);

  // menu Modifica
  menu = new JMenu("Modifica");
  menu.setMnemonic(KeyEvent.VK_M);
  barra.add(menu);

  // voci
  voce = new JMenuItem("Taglia", KeyEvent.VK_L);
  voce.addActionListener(this);
  menu.add(voce);
  voce = new JMenuItem("Copia", KeyEvent.VK_C);
  voce.addActionListener(this);
  menu.add(voce);
  voce = new JMenuItem("Incolla", KeyEvent.VK_O);
  voce.addActionListener(this);
  menu.add(voce);

  return barra;
}
```

```
  // funzioni del menu
  public void apriTesto()
  {
    int status = fileChooser.showOpenDialog(this);
    if (status==JFileChooser.APPROVE_OPTION)
    {
      try
      {
        File f = fileChooser.getSelectedFile();
        Reader fIN = new FileReader(f);
        testo.read(fIN,null);
        setTitle(f.getName());
      }
      catch(Exception e) {}
    }
  }

  public void salvaTesto()
  {
    int status = fileChooser.showSaveDialog(this);
    if (status==JFileChooser.APPROVE_OPTION)
    {
      try
      {
        File f = fileChooser.getSelectedFile();
        Writer fOUT = new FileWriter(f);
        testo.write(fOUT);
        setTitle(f.getName());
      }
      catch(Exception e) {}
    }
  }

  // gestione degli eventi
  public void actionPerformed(ActionEvent e)
  {
    JMenuItem source = (JMenuItem)(e.getSource());
    String s = source.getText();
    if (s.equals("Nuovo"))
    {
      testo.setText("");
      setTitle("Nuovo");
    }
    if (s.equals("Apri")) apriTesto();
    if (s.equals("Salva")) salvaTesto();
    if (s.equals("Esci")) System.exit(0);
    if (s.equals("Taglia")) testo.cut();
    if (s.equals("Copia")) testo.copy();
    if (s.equals("Incolla")) testo.paste();
  }
}
```

PROGRAMMA JAVA (*EditorTesto.java*)

```
class EditorTesto
{
  public static void main(String argv[])
  {
    Editor f = new Editor();

    f.setTitle("Editor di testo");
    f.pack();
    f.setVisible(true);
  }
}
```



Le caratteristiche dell'area di testo sono impostate con le seguenti istruzioni:

```
testo.setEditable(true);
```

rende editabile il contenuto dell'area di testo,

```
testo.setFont(new Font("Arial", Font.PLAIN, 12));
```

imposta il font di visualizzazione,

```
testo.setLineWrap(true);
```

porta a capo una linea quando si raggiunge l'estremo destro dell'area di testo,

```
testo.setWrapStyleWord(true);
```

porta a capo la parola intera (determinata dallo spazio bianco tra i caratteri) quando si raggiunge l'estremo destro dell'area di testo.

L'esecuzione del metodo **showOpenDialog** sulla componente **FileChooser** restituisce un valore di stato che è uguale a **APPROVE_OPTION** se l'operazione è stata eseguita correttamente.
Il metodo **getSelectedFile** poi restituisce il file selezionato. A questo punto il flusso associato al file viene copiato nell'area di testo con il metodo **read** e il titolo della finestra è sostituito con il nome del file (**setTitle**):

```
int status = fileChooser.showOpenDialog(this);
if (status==JFileChooser.APPROVE_OPTION)
{
  try
  {
    File f = fileChooser.getSelectedFile();
    Reader fIN = new FileReader(f);
    testo.read(fIN,null);
    setTitle(f.getName());
  }
  catch(Exception e) {}
}
```

Considerazioni analoghe possono essere fatte per l'operazione di scrittura sul file con la scelta *Salva*.

Le operazioni del menu *Modifica* (*Taglia*, *Copia* e *Incolla*) sono gestite con i metodi predefiniti della componente *JTextArea*:
- **cut**
- **copy**
- **paste**.

AUTOVERIFICA

Problemi da 22 a 24 pag. 348

7. Simulatore di un miscelatore di acqua
8. Rilevazione di dati in un stazione meteorologica

PROBLEMI

Elementi base dell'interfaccia utente

1 Descrivere a parole e schematizzare con un disegno l'interfaccia grafica utilizzabile nelle seguenti situazioni, specificando quali sono gli elementi grafici usati: una finestra per gestire l'input di due numeri qualunque; una maschera per inserire una data composta da giorno, mese e anno; un programma per disegnare (per esempio il programma *Paint* di Windows); un programma che gestisce la spedizione di messaggi di posta elettronica.

2 Basandosi sui software conosciuti, elencare alcuni esempi di eventi che vengono generati quando si utilizza il mouse.

3 Definire, con il linguaggio Java, le seguenti finestre sia con la libreria Swing che con AWT:
a) una finestra di dimensione 320x240
b) una finestra di dimensione 100x100 contenente due pulsanti, *Ok* e *Annulla*
c) una finestra con due etichette contenenti due nomi di persona.

4 Scrivere un programma Java che acquisisca da riga di comando l'anno di nascita dell'utente (interfaccia a caratteri) e, successivamente, apra una finestra grafica per visualizzare l'età in un'etichetta (interfaccia grafica).

Programmazione visuale con NetBeans

5 Creare un progetto e inserire nella finestra di progettazione gli oggetti necessari per implementare le componenti seguenti:
a) un'etichetta con la scritta *Inserisci il nome*,
b) un'etichetta con la scritta *1.234,5* allineata a destra,
c) un'etichetta con la scritta *Programma 8.0* di colore blu su sfondo giallo,
d) un pulsante con la scritta *OK*,
e) un pulsante con la scritta *Annulla* disabilitato,
f) una casella di testo larga 200 pixel e alta 30 pixel,
g) una casella di testo contenente la scritta *Msg fisso* e resa non modificabile.

6 Modificare il Progetto 2 *CalcolatriceProject* presentato nell'inserto, aggiungendo un pulsante *Azzera*: quando l'utente fa clic su di esso, il programma pulisce le tre caselle di testo impostando una stringa vuota.

7 Creare un'applicazione con interfaccia grafica per calcolare il prodotto tra due numeri.

8 Creare un'applicazione con interfaccia grafica in cui l'utente può inserire la velocità in chilometri orari e, dopo aver fatto clic su un pulsante di conversione, ottiene il valore espresso in metri al secondo.

9 Creare un'applicazione grafica per effettuare calcoli con la legge di Ohm. Dati V (differenza di potenziale) e I (intensità della corrente), calcolare la resistenza R con la formula R=V/I. Descrivere anche quali componenti grafiche si intendono utilizzare per l'interfaccia utente.

10 Scrivere l'applicazione per la conversione di una misura da pollici a centimetri e per la conversione inversa, presentando all'utente due pulsanti per scegliere il verso della conversione.

Componenti grafiche

11 Scrivere il codice Java per inserire i seguenti oggetti in una finestra grafica:
a) un'area di testo di dimensioni 5x10
b) un'area di testo contenente tre numeri su ogni riga
c) una casella combinata per scegliere un giudizio tra i seguenti valori: ottimo, buono, sufficiente e insufficiente
d) una casella combinata per scegliere un numero tra 1 e 31
e) una casella di controllo per selezionare uno o più dei dodici mesi dell'anno.

12 Ripetere i problemi precedenti, usando l'ambiente di sviluppo visuale *NetBeans*.

13 Scrivere il programma Java per ciascuna delle seguenti interfacce, disponendo gli elementi grafici nei contenitori con un opportuno layout:
a) un pulsante *OK* e *Annulla* all'interno di un pannello
b) una casella di testo preceduta da una scritta
c) in una finestra con disposizione *BorderLayout*, un pulsante per ogni zona
d) nella parte alta di una finestra una casella di testo, nella parte bassa un'area di testo
e) tre bottoni in un pannello disposti verticalmente
f) una finestra contenente tre pannelli di colore verde, bianco e rosso.

14 Disegnare all'interno di una finestra una griglia 4x4 contenente 16 caselle di testo. Colorare le celle usando due colori diversi.

Gestione di eventi

15 Creare un'interfaccia grafica con un'area di testo formata da 60 colonne e 10 righe in cui l'utente può scrivere del testo. Aggiungere anche un pulsante: quando l'utente fa clic su di esso, il programma pulisce l'area di testo cancellando tutto il suo contenuto.

16 Creare un'interfaccia grafica composta da una casella di testo, un pulsante e un'area di testo. L'utente utilizza la casella di testo per inserire dei nomi. Quando fa clic sul pulsante, i nomi vengono ricopiati nell'area di testo.

17 Scrivere il programma per richiedere all'utente un numero e controllare che il numero inserito sia maggiore di 500; se il valore non è maggiore, il programma visualizza una finestra con un messaggio di avvertimento, altrimenti apre una finestra per visualizzare il valore inserito.

18 Scrivere il programma per consentire all'utente la scelta in un elenco di animali. In corrispondenza della scelta effettuata, viene visualizzata l'immagine dell'animale.

19 Scrivere il programma per consentire all'utente la scelta in un elenco di colori. Il colore scelto diventa lo sfondo di una casella di testo posizionata al centro della finestra. Controllare anche l'uscita dal programma chiedendone conferma all'utente.

20 Simulare la parte della finestra di dialogo per le stampe tipica delle applicazioni in Windows con la quale si può scegliere l'orientamento (orizzontale o verticale) della pagina da stampare: usare i pulsanti di opzione.

21 Creare una finestra che resta aperta finché l'utente non inserisce in una casella di testo la parola *FINE*.

Menu

22 Aprire un applicativo in ambiente Windows: osservare le voci presenti nei primi due menu e riprodurre in una finestra la struttura dei menu con le voci di ciascuno.

23 Costruire un programma che consenta di scegliere un nome di nazione all'interno di un elenco organizzato come menu a tendina. Il nome scelto viene scritto con un'etichetta all'interno della finestra.

24 In una finestra sono inseriti un pulsante e un'area di testo: facendo clic sul pulsante si apre la finestra di dialogo che consente di selezionare un file presente sul disco; il nome del file selezionato viene poi aggiunto all'area di testo.

parte quarta

Applicazioni Web

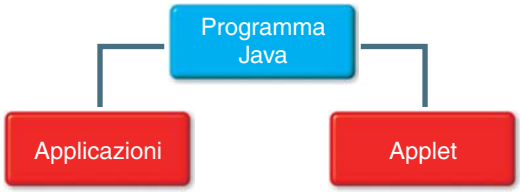
6 Applet

OBIETTIVI DI APPRENDIMENTO

In questo capitolo imparerai a costruire le applet in Java per gestire l'interazione tra l'utente e l'ambiente Web. Sarai in grado di utilizzare le componenti grafiche per ottenere interfacce utilizzabili nelle pagine Web. Imparerai anche a gestire gli eventi del mouse e a costruire applet con disegni o immagini.

1 Applicazioni e applet

I programmi che vengono scritti con il linguaggio Java si dividono in due grandi categorie:
- le applicazioni
- le applet.



Le **applicazioni** sono state presentate nei capitoli precedenti e costituiscono programmi Java completi a sé stanti.
Le **applet** (piccole applicazioni) sono anch'esse programmi Java, ma possono essere eseguite solo all'interno di un browser Web, cioè, per poter funzionare, devono appoggiarsi a un altro programma. Quasi tutto quello che si può fare con un'applicazione, può essere realizzato anche usando un'applet. Esistono però delle limitazioni per le applet, dovute a problemi di sicurezza, per evitare le possibili operazioni illecite che un programma, scaricato dalla rete ed eseguito sul computer dell'utente collegato, potrebbe compiere.

Il grande successo ottenuto dalle applet è legato allo sviluppo della rete Internet. Essa collega milioni di computer sparsi in tutto il mondo, consentendo la condivisione di diverse informazioni. Queste informazioni sono principalmente memorizzate in documenti collegati tra loro in modo ipertestuale usando la tecnologia **WWW** (*World Wide Web*); per questo motivo si chiamano comunemente **pagine Web**. In pratica, una pagina corrisponde a un file, che viene costruito usando un linguaggio chiamato **HTML** (*HyperText Markup Language*). Le prime versioni del linguaggio HTML permettevano di creare pagine formate solo da testi e immagini. Successivamente sono state aggiunte altre possibilità per inserire suoni, filmati, applicazioni eseguibili e interattive.

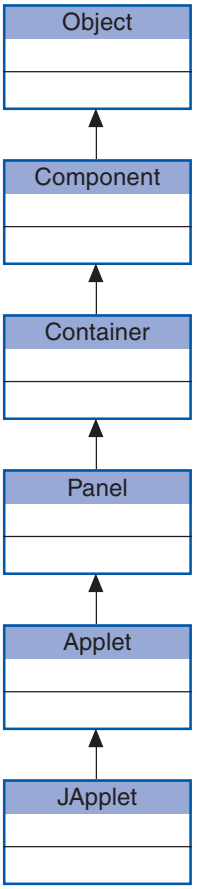
Per accedere alle informazioni di Internet si utilizzano i programmi applicativi per la visualizzazione delle pagine Web, chiamati **browser**. Essi sono in grado di interpretare il codice HTML e di mostrare il contenuto delle pagine in formato grafico.
Nei casi più semplici di pagine Web, i browser si limitano a posizionare il testo e le immagini all'interno della loro finestra, seguendo le indicazioni specificate nel file. Se le pagine HTML contengono un richiamo a un'applet, il browser Web ha il compito di eseguirla. Un'applet viene quindi eseguita automaticamente, se viene inserita in modo opportuno in una pagina HTML. Java è un linguaggio di programmazione che, attraverso le applet, si integra con il Web. Le applet favoriscono l'interattività e rendono le pagine Web più dinamiche e più attraenti. Le pagine, in cui sono state aggiunte le applet, consentono di ricevere ed elaborare l'input dell'utente oltre che di visualizzare grafici e animazioni.

Tra i browser Web più conosciuti ci sono *Google Chrome*, *Firefox, Internet Explorer, Safari* e *Opera*. È opportuno sottolineare che i browser Web eseguono le applet in modi leggermente diversi: si possono quindi ottenere dei comportamenti diversi a seconda del tipo di browser utilizzato. In alternativa ai browser Web, è possibile visualizzare un'applet usando il programma **appletviewer**, che è uno strumento software fornito insieme al JDK di *Oracle*. L'*appletviewer* viene eseguito indicando il nome di un file HTML. Se il file non contiene applet, l'*appletviewer* termina l'esecuzione, altrimenti apre una finestra ed esegue le applet contenute nel file. Questo strumento può risultare molto utile nella fase di collaudo delle applet.

MATERIALI ON LINE

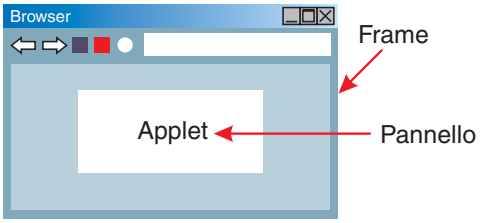
1. Richiami sugli elementi del linguaggio HTML

2 La classe Applet



Le applet vengono create estendendo la classe predefinita **Applet**, contenuta nel package **java.applet**. Osservando la gerarchia delle classi di Java, si vede che la classe Applet è una sottoclasse della classe **Panel** e ne eredita tutte le proprietà. Le applet si comportano, quindi, come dei contenitori: possono contenere più componenti e possono impostare un loro *Layout Manager*.

Un'applet è una particolare applicazione grafica perché non viene eseguita all'interno di una propria finestra, ma usa la finestra di un browser. Diversamente dalle applicazioni, non si deve creare un *Frame* dove inserire i pannelli e le componenti.
Il *Frame* è fornito dal browser mentre l'applet, che rappresenta un pannello, viene aggiunta a questa finestra. Questo pannello occupa una zona all'interno del browser, e le sue dimensioni vengono definite da un particolare tag HTML nella pagina Web.



Le applet non hanno bisogno di definire un loro *Window Listener* per gestire gli eventi legati alla finestra (per esempio la chiusura della finestra), perché è il browser che si occupa di questo.
Per il resto, un'applet può essere costruita allo stesso modo delle applicazioni, può contenere pulsanti, caselle di testo, aree di disegno e tutte le altre componenti grafiche di Java.

Nel linguaggio Java si possono usare due versioni di applet:

- la classe **Applet** del package AWT
- la classe **JApplet**, più recente, del package Swing (*javax.swing.JApplet*).

La classe *JApplet* è una sottoclasse di **java.applet.Applet**. Le applet che contengono componenti *Swing* devono essere realizzate con una sottoclasse che estende la classe *JApplet*. Negli esempi seguenti useremo la classe *JApplet*.

La struttura generale di un'applet è la seguente:

```
import java.awt.*;
import javax.swing.*;

public class NomeApplet extends JApplet
{
  // attributi
  // metodi
}
```

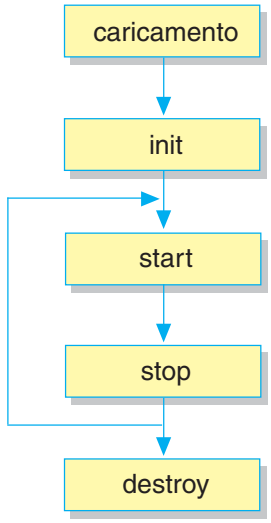
Le istruzioni *import* servono per rendere visibili la classe *JApplet* e le componenti grafiche all'interno del programma. Segue poi la dichiarazione della classe. Deve essere una classe dichiarata con la parola chiave *public* e deve essere sottoclasse della classe *JApplet*. Al suo interno sono presenti gli attributi e i metodi di cui è composta.
Come per le applicazioni, l'applet deve essere memorizzata in un file con estensione *.java* e con lo stesso nome della classe, corrispondente al nome dell'applet. La compilazione viene eseguita con il procedimento usato anche per le applicazioni, cioè con il comando *javac*. Il risultato della compilazione è la generazione di un file *.class*. Se un'applet è composta da più classi, si otterranno più file *.class*.

Nelle applet non è presente il metodo *main*, quindi serve un altro modo per indicare in quale punto ha inizio l'esecuzione. Al suo posto viene usato il metodo **init**, uno dei tanti metodi contenuti nella classe JApplet ed ereditato dalle sue sottoclassi. Questo metodo deve essere ridefinito nella sottoclasse che definisce la nuova applet. Solitamente, nel metodo *init*, si inseriscono le operazioni di inizializzazione, come per esempio il posizionamento delle componenti all'interno dell'applet e il caricamento di eventuali immagini.

Oltre al metodo *init*, la classe *JApplet* contiene altri metodi usati per gestire la sua esecuzione:

- start
- stop
- destroy.

L'esecuzione di questi metodi viene attivata direttamente dal browser Web che, in questo modo, può controllare il funzionamento dell'applet.
Appena un'applet viene caricata, il browser Web richiama il metodo *init*; successivamente invoca il metodo **start**. Ogni volta che ci si sposta su una pagina diversa, per esempio perché è stato fatto un clic su un link, il browser Web invoca il metodo **stop** delle applet che sta visualizzando. Se si ritorna alla pagina in cui c'era l'applet, viene invocato nuovamente il metodo *start*. I metodi *start* e *stop* possono essere eseguiti più volte dall'applet, mentre il metodo *init* viene eseguito solo una volta, subito dopo il caricamento dell'applet. Infine, quando il browser viene chiuso, viene invocato il metodo **destroy**. Il modo con cui questi metodi vengono richiamati, varia leggermente a seconda dei browser Web utilizzati.



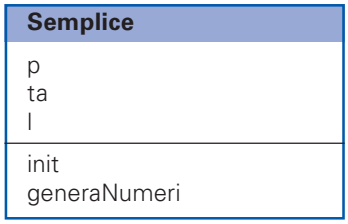
Il metodo più utilizzato è *init*. Se non ci sono operazioni da inserire negli altri metodi, è possibile non ridefinirli e quindi non inserirli nell'applet. I metodi *start* e *stop* sono usati principalmente per attivare e sospendere le eventuali animazioni eseguite dall'applet: quando una pagina viene caricata, con il metodo *start* si fa partire l'animazione, mentre il metodo *stop* consente di fermarla. La presenza dei metodi *start* e *stop* serve per evitare di tenere occupato il processore quando l'applet non è più visibile.

Un'applet può essere costruita usando tutte le componenti contenute nei package *java.awt* e *javax.swing*.
Nel seguente esempio viene costruita una semplice applet con alcuni elementi grafici.

PROGETTO 1

Costruire un'applet con un'etichetta e un'area di testo. Riempire poi l'area di testo con 10 numeri generati casualmente.

Ogni elemento grafico dell'applet viene dichiarato come attributo della classe, il cui diagramma è il seguente:



I tre elementi grafici sono il pannello *p*, l'area di testo *ta* e l'etichetta *l*.

Il metodo *init* crea i due oggetti grafici e li posiziona all'interno del pannello usando come Layout Manager il *BorderLayout*. Quest'ultimo viene poi aggiunto al centro, nel *content pane* dell'applet usando l'istruzione:

```
getContentPane().add(p, BorderLayout.CENTER);
```

In alternativa all'istruzione precedente, si può anche usare l'istruzione:

```
setContentPane(p);
```

che imposta il pannello *p* come *content pane* dell'applet.

Il metodo *generaNumeri* contiene il ciclo per calcolare un numero casuale e inserirlo nell'area di testo con il metodo **append**.

APPLET JAVA (*Semplice.java*)

```
import java.awt.*;
import javax.swing.*;

public class Semplice extends JApplet
{
  private JPanel p = new JPanel();
  private JTextArea ta;
  private JLabel l;
```

```
  public void init()
  {
    ta = new JTextArea(10,8);
    l = new JLabel("Elenco di numeri casuali", JLabel.CENTER);

    generaNumeri();

    // dispone le componenti all'interno del pannello
    p.setLayout(new BorderLayout());
    p.add(l, "North");
    p.add(ta, "Center");
    getContentPane().add(p, BorderLayout.CENTER);
  }

  public void generaNumeri()
  {
    int casuale;
    for(int i=1; i<=10; i++)
    {
      casuale = (int) (Math.random()*1000);
      ta.append(casuale+"\n");
    }
  }
}
```

Per compilare questa applet, bisogna eseguire il comando

```
javac Semplice.java
```

Il risultato è un file compilato chiamato *Semplice.class*. Questo programma non può essere eseguito come un'applicazione, usando il comando *java*, perché manca il metodo *main*. L'applet deve essere eseguita all'interno di un browser Web e, per farlo, si deve predisporre un opportuno file HTML con l'indicazione dell'applet che si vuole visualizzare. L'applet viene inserita in una pagina HTML attraverso il tag **<APPLET>**. Il tag può contenere diversi elementi che indicano al browser Web le modalità per visualizzare l'applet.
Il tag <APPLET> deve contenere alcuni elementi obbligatori e la sua struttura più semplice è la seguente:

```
<APPLET CODE="nomeapplet" WIDTH="larghezza" HEIGHT="altezza">
</APPLET>
```

L'elemento **CODE** specifica il nome della classe principale dell'applet, quella contenente il metodo *init*. Deve essere indicato il nome del file compilato *.class*.
Questo file deve trovarsi nella stessa directory in cui si trova la pagina HTML.
Gli elementi **WIDTH** e **HEIGHT** indicano le dimensioni in pixel di un'area rettangolare. Il browser Web riserva quest'area per inserire ed eseguire l'applet. Le dimensioni dell'applet non vengono fissate all'interno del programma, ma nel file HTML e l'applet è costretta a rispettare queste dimensioni.

Le versioni dei browser Web che non riconoscono il tag <APPLET>, lo ignorano; per segnalare questa situazione, si può far comparire un messaggio di avvertimento, inserendo il testo tra il tag di apertura e di chiusura.
Gli elementi *CODE*, *WIDTH* e *HEIGHT* sono obbligatori e devono essere sempre indicati.

La pagina HTML, da utilizzare per eseguire l'applet precedente, è la seguente:

```
<HTML>
<HEAD>
<TITLE>Semplice applet</TITLE>
</HEAD>
<BODY>
<APPLET CODE="Semplice.class" WIDTH="300" HEIGHT="200">
Applet non eseguibile.
</APPLET>
</BODY>
</HTML>
```

L'applet occupa un'area rettangolare avente le dimensioni di 300x200 pixel. La clausola CODE indica il nome del file compilato *Semplice.class*. Se il browser non riconosce il tag <APPLET> viene visualizzato il messaggio "*Applet non eseguibile*".

Esistono altri elementi opzionali che possono essere aggiunti all'interno del tag <APPLET>.

- **CODEBASE**: specifica l'URL dove è posizionata l'applet; se non è indicato, viene usato l'URL della pagina HTML contenente l'applet.
- **ALT**: contiene il messaggio da visualizzare quando il browser, pur riconoscendo il tag <APPLET>, in base alle sue impostazioni, non può eseguire le applet.
- **NAME**: permette di associare un nome all'applet.
- **ALIGN**: indica dove deve essere allineata l'applet; i possibili valori sono: *left*, *right*, *top*, *texttop*, *middle*, *absmiddle*, *baseline*, *bottom*, *absbottom*.

Per esempio:

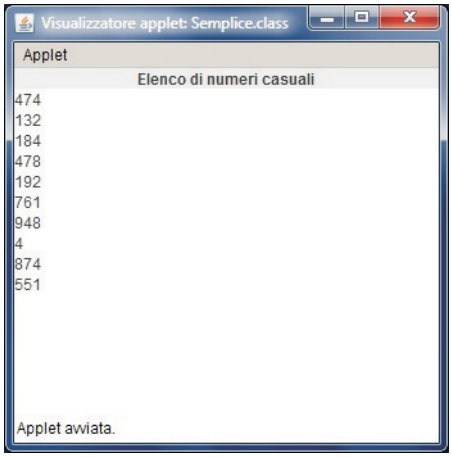
```
<APPLET CODEBASE="http://sito345.it/esercizi/"
CODE="Semplice.class"
WIDTH="250" HEIGHT="250"
ALT="Esecuzione vietata"
ALIGN="right"
NAME="esempio">
</APPLET>
```

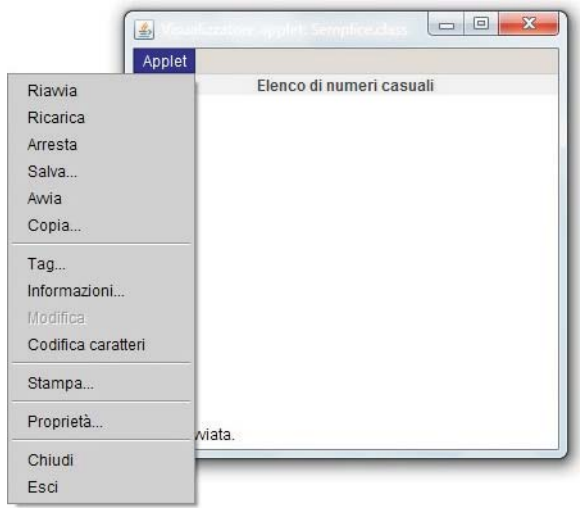
Se non si ha a disposizione un browser Web per eseguire le applet, è possibile usare il programma **appletviewer**, distribuito insieme al JDK di Oracle. Questo programma simula un browser Web nell'esecuzione delle applet ma non visualizza le pagine HTML. È un minibrowser che interpreta solo i tag <APPLET> e ignora gli altri. Per eseguire l'appletviewer occorre attivare la finestra del *Prompt dei comandi*. Si deve quindi digitare il comando *appletviewer* seguito dal nome del file HTML che contiene l'applet.

Per esempio, se il file HTML contenente l'applet si chiama *Semplice.htm,* e si trova nella directory *esempi* del disco C:, il comando per attivare l'appletviewer è il seguente:

```
C:\esempi>appletviewer Semplice.htm
```

L'appletviewer apre una finestra all'interno della quale viene eseguita l'applet.
Questa finestra possiede un menu con il quale è possibile controllare l'esecuzione dell'applet: si può interrompere e riattivare l'esecuzione dell'applet, si può ricaricare nuovamente l'applet e si possono impostare le proprietà con cui viene eseguito l'appletviewer.





AUTOVERIFICA

Domande da 1 a 6 pag. 383
Problemi da 1 a 4 pag. 385-386

3 I parametri

Il tag <APPLET> può anche contenere i **parametri** che devono essere passati all'applet. Questi sono utili perché permettono di costruire applet generiche che vengono poi adattate in base al valore dei parametri. Per esempio, un parametro può specificare il colore da utilizzare per lo sfondo dell'applet. Usando i parametri, è possibile inserire la stessa applet in più pagine usando diversi valori del parametro. In questo modo si evita di dover creare applet diverse.
La struttura del tag <APPLET> contenente i parametri è la seguente:

```
<APPLET CODE="nomeapplet" WIDTH="larghezza" HEIGHT="altezza">
<PARAM NAME= "nome del parametro" VALUE= "valore del parametro" >
<PARAM NAME= "nome del parametro" VALUE= "valore del parametro" >
. . .
</APPLET>
```

Per ogni parametro (**PARAM**) si deve indicare il nome dopo la parola **NAME** e il valore dopo la parola **VALUE**. Tutti i parametri vanno elencati all'interno del tag <APPLET>.

Per leggere il valore di un parametro in un'applet si usa il metodo **getParameter**.

Questo metodo è contenuto nella classe *Applet* e richiede che venga specificato il nome del parametro che si vuole leggere. Il nome deve corrispondere a un nome indicato nel tag <APPLET> dopo la parola *NAME*. Il metodo *getParameter* restituisce una stringa che contiene il valore del parametro. Se il parametro è un numero bisogna convertire la stringa usando il metodo opportuno. Se non esiste nessun parametro associato al nome, il metodo *getParameter* restituisce un valore *null*.

PROGETTO 2

Realizzare un'applet parametrica che mostra un messaggio generico.

I due parametri dell'applet vengono usati per indicare il testo da visualizzare e la grandezza dei caratteri del testo.

Il tag <APPLET> deve specificare i parametri come si vede nella seguente pagina Web:

PAGINA WEB (*Parametri.htm*)

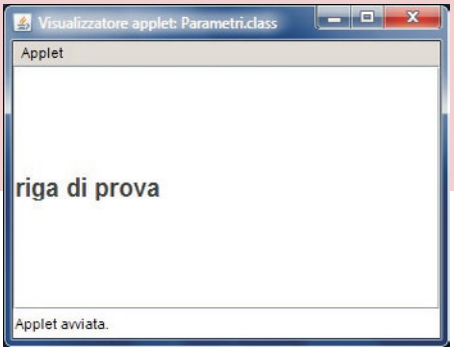
```
<HTML>
<HEAD>
<TITLE>Semplice applet</TITLE>
</HEAD>
<BODY>
<APPLET CODE="Parametri.class"
        WIDTH="300" HEIGHT="200">
<PARAM NAME="testo" VALUE="riga di prova">
<PARAM NAME="grandezza" VALUE="20">
Applet non eseguibile.
</APPLET>
</BODY>
</HTML>
```

I parametri vengono letti dall'applet e assegnati ai rispettivi attributi *messaggio* e *grandezza*. Per eseguire la conversione del secondo parametro si utilizza il metodo *Integer.parseInt* e, nel caso si verifichino degli errori di conversione, si assegna un valore predefinito per la grandezza.

APPLET JAVA (*Parametri.java*)

```
import java.awt.*;
import javax.swing.*;

public class Parametri extends JApplet
{
  private JPanel p = new JPanel();
  private JTextField tf;
  private Font f;
  private String messaggio;
  private int grandezza;
```

```
  public void init()
  {
    // riceve i parametri
    String messaggio = getParameter("testo");
    try
    {
      String g = getParameter("grandezza");
      grandezza = Integer.parseInt(g);
    }
    catch(Exception e)
    {
      grandezza = 10;
    }

    // costruisce l'interfaccia
    tf = new JTextField(30);
    tf.setText(messaggio);
    f = new Font("Arial", Font.BOLD, grandezza);
    tf.setFont(f);
    p.setLayout(new BorderLayout());
    p.add(tf, "Center");
    setContentPane(p);
  }
}
```



APPLET E BROWSER

Quando il browser incontra il tag <APPLET> nella pagina Web, viene lanciata l'esecuzione della *Java Virtual Machine* (JVM) incorporata nel browser. Occorre osservare però che le versioni del browser non sempre sono sincronizzate con le più recenti versioni di Java e si potrebbe verificare l'impossibilità di eseguire l'applet oppure una perdita di funzionalità. Per questo motivo Oracle mette a disposizione uno strumento denominato **Java Plug-in** che viene installato automaticamente con il JDK oppure con il JRE per l'esecuzione delle applicazioni Java (il termine **plug-in** indica, in generale, un piccolo programma aggiuntivo per il browser in grado di visualizzare gli oggetti creati con particolari applicazioni).
Nelle versioni più recenti del linguaggio HTML, il tag <APPLET> viene considerato obsoleto e, secondo le raccomandazioni del W3C (*World Wide Web Consortium*), dovrebbe essere sostituito con il tag **<OBJECT>** che indica, in generale, un oggetto multimediale o eseguibile da inserire in una pagina Web.
Il tag <OBJECT> provoca la chiamata del *Java Plug-in*, anziché della *Virtual Machine*, per l'esecuzione dell'applet.
Per comodità di lettura rivediamo la pagina Web utilizzata nel paragrafo precedente,che utilizza il tag <APPLET>. Tutte le versioni recenti dei browser interpretano correttamente questo tag.

PAGINA WEB

```
<HTML>
<HEAD>
<TITLE>Semplice applet</TITLE>
</HEAD>
<BODY>
<APPLET CODE="Semplice.class" WIDTH="300" HEIGHT="200">
Applet non eseguibile.
</APPLET>
</BODY>
</HTML>
```

La stessa pagina può essere riscritta per il browser *Mozilla Firefox*, usando il tag <OBJECT>, nel seguente modo:

PAGINA WEB

```
<HTML>
<HEAD>
<TITLE>Semplice applet</TITLE>
</HEAD>
<BODY>
<OBJECT CODETYPE="application/java"
        CLASSID="java:Semplice.class"
        WIDTH="300" HEIGHT="200">
Applet non eseguibile.
</OBJECT>
</BODY>
</HTML>
```

Il nome dell'applet viene indicato come valore dell'attributo **CLASSID** dell'oggetto. Il valore assegnato all'attributo **CODETYPE** specifica che si tratta di un oggetto di tipo *applet* in Java. Inoltre, analogamente al tag <APPLET>, le eventuali righe scritte tra <OBJECT> e </OBJECT> vengono visualizzate nel caso in cui il browser non sia in grado di interpretare il codice HTML. Per il browser *Internet Explorer* il codice HTML della pagina Web assume una forma leggermente diversa:

PAGINA WEB

```
<HTML>
<HEAD>
<TITLE>Semplice applet</TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="clsid:CAFEEFAC-0014-0002-FFFF-ABCDEFFEDCBA"
WIDTH="300" HEIGHT="200">
<PARAM NAME="code" VALUE="Semplice.class">
Applet non eseguibile.
</OBJECT>
</BODY>
</HTML>
```

All'attributo **CLASSID** è assegnato un valore prefissato che identifica la versione più recente del controllo *ActiveX* tra quelle installate sul computer; nel caso in cui questo identificativo corrisponda a una vecchia versione del *Plug-in*, all'utente viene inviato un messaggio per suggerire il download dell'aggiornamento.
Il nome dell'applet viene passato come valore del parametro **code**.

Come si vede, la sintassi del tag <OBJECT> per il passaggio dei **parametri**, attraverso l'elemento PARAM, è identica a quella utilizzata per il tag <APPLET>.

Attraverso un artificio è possibile costruire una pagina Web che possa funzionare con qualsiasi browser. La pagina Web usa i **commenti condizionali** che funzionano solo con *Internet Explorer* e che vengono interpretati come normali commenti dagli altri browser.

La loro struttura di base è uguale ai commenti HTML:

```
<!-- ... -->
```

ma *Internet Explorer* riconosce anche la seguente sintassi speciale:

```
<!--[if IE]> ... <![endif]-->
```

oppure

```
<!--[if !IE]> ... <![!endif]-->
```

che controlla se il browser è *Internet Explorer* (IE) o altro (! sta per *diverso da*).

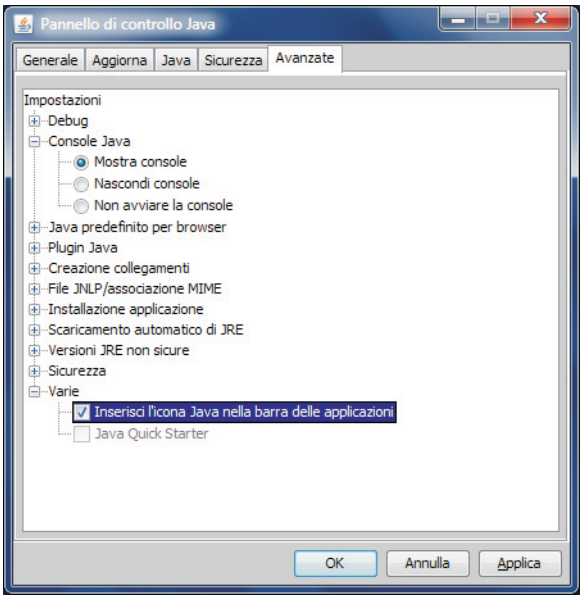
PAGINA WEB

```
<HTML>
<HEAD>
<TITLE>Semplice applet</TITLE>
</HEAD>
<BODY>
<!--[if IE]>
<OBJECT CLASSID="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
        WIDTH="300" HEIGHT="200" >
<PARAM NAME="code" VALUE="Semplice.class">
Applet non eseguibile.
</OBJECT>
<![endif]-->
<!--[if !IE]>-->
<OBJECT CLASSID="java:Semplice.class"
        CODETYPE="application/java"
        WIDTH="300" HEIGHT="200">
Applet non eseguibile.
</OBJECT>
<!--[!endif]-->
</BODY>
</HTML>
```

La prima parte viene interpretata da *Internet Explorer* come commento condizionale e viene eseguita, mentre viene considerata un insieme di righe di commento normale dagli altri browser perché sono delimitate da <!– – e – –>.
La seconda parte viene interpretata da *Internet Explorer* come commento condizionale e non viene eseguita perché c'è la condizione *!IE*, mentre gli altri browser considerano solo la riga iniziale e la riga finale come commento (perché ci sono i simboli – –> di chiusura del commento) e possono eseguire le righe comprese tra esse.

4 Interazione con il browser Web

Come già visto in precedenza, il browser Web controlla le applet invocando i metodi *init*, *start*, *stop* e *destroy*. Le applet, a loro volta, possono eseguire comandi che hanno effetti sul browser: possono inviare messaggi sul video (*console*) o sulla barra di stato, oppure possono indicare quale pagina HTML deve essere visualizzata dal browser. Le applicazioni Java usano normalmente il metodo *System.out.println* per visualizzare i messaggi sul video. Con le applet è possibile utilizzare questo metodo, visualizzando i messaggi in una particolare finestra chiamata **Java Console**. In Windows questa finestra può essere abilitata scegliendo la voce *Java* da *Pannello di controllo*, *Programmi*: nella scheda *Avanzate*, si deve impostare *Mostra console* e richiedere l'inserimento dell'icona di Java nella barra delle applicazioni.



La *Java Console* è utile perché mostra anche tutti i possibili errori che si possono verificare durante l'esecuzione dell'applet. È usata principalmente da chi programma le applet per verificare che funzionino correttamente.
Può essere utile inserire nel programma le istruzioni *System.out.println* per eseguire il debug nel caso si verifichino degli errori. Per esempio si può usare questo metodo per stampare il valore delle variabili in particolari momenti oppure per stampare messaggi al termine dell'esecuzione di una parte del programma.
È possibile utilizzare questa tecnica per capire in quale ordine il browser richiama i metodi *init*, *start*, *stop* e *destroy*: basta inserire in ognuno di questi metodi un'istruzione di stampa; eseguendo l'applet, nella *Java Console* viene stampato l'ordine con cui vengono richiamati i metodi. L'ordine varia a seconda del browser usato.

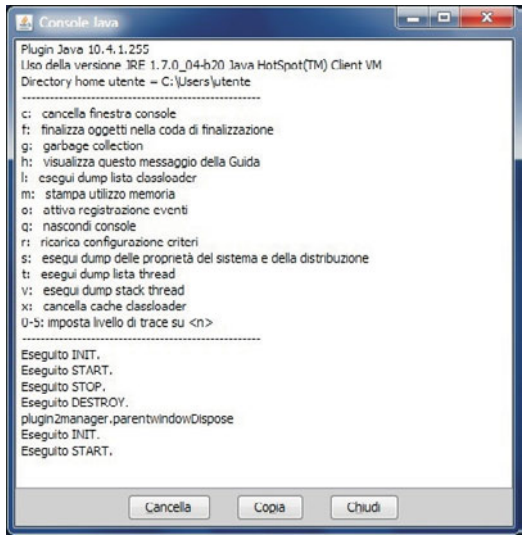
APPLET JAVA (*Ordine.java*)

```
import javax.swing.*;

public class Ordine extends JApplet
{
  public void init()
  {
    System.out.println("Eseguito INIT.");
  }
  public void start()
  {
    System.out.println("Eseguito START.");
  }
  public void stop()
  {
    System.out.println("Eseguito STOP.");
  }
  public void destroy()
  {
    System.out.println("Eseguito DESTROY.");
  }
}
```

PAGINA WEB (*Ordine.htm*)

```
<HTML>
<HEAD>
<TITLE>Console Java</TITLE>
</HEAD>
<BODY>
<APPLET CODE="Ordine.class"
       WIDTH="300" HEIGHT="500">
Applet non eseguibile.
</APPLET>
</BODY>
</HTML>
```

Si apra ora con il browser il file *htm* che richiama l'esecuzione dell'applet: sulla barra delle applicazione compare l'icona della Java Console, che può essere aperta con un clic sull'icona. La figura mostra un possibile risultato dell'esecuzione di questa applet per due volte consecutive (dopo aver aperto la pagina Web, fare clic sul pulsante *Aggiorna* del browser).



I browser Web (ma anche *appletviewer*) possiedono una **barra di stato** posizionata nella parte bassa della finestra, dove vengono mostrati i messaggi che forniscono le informazioni all'utente oppure l'URL dei file che sono in fase di caricamento.
Un'applet Java può scrivere i messaggi sulla barra di stato usando il metodo **showStatus**.

Esso è un metodo contenuto nella classe *Applet* e riceve come parametro la stringa da visualizzare. Per esempio:

```
showStatus("Applet Java vers. 1.7");
```

Un'applet può indicare al browser quale pagina HTML deve visualizzare. Grazie a questa possibilità si possono costruire pagine HTML nelle quali la **gestione dei link** è eseguita graficamente da un'applet. Per caricare una nuova pagina viene usato il metodo **showDocument**, che appartiene alla classe **AppletContext**. Il parametro passato è l'indirizzo della pagina da visualizzare. L'indirizzo deve essere specificato usando la classe **URL** contenuta nel package **java.net**. Un oggetto di classe URL può essere creato con due costruttori.
Nell'esempio seguente l'oggetto *indirizzo* è creato con il costruttore che ha come unico parametro l'indirizzo completo della pagina HTML.

```
try
{
  URL indirizzo = new URL("http://www.oracle.com/technetwork/java/
index.html");
}
catch (MalformedURLException e)
{
  System.out.println("Errore nella creazione dell'URL.");
}
```

La creazione deve gestire la possibile eccezione che si verifica se l'URL non è specificato correttamente, utilizzando una struttura *try* ... *catch*. Un URL, creato usando il costruttore precedente, indica un riferimento assoluto alla pagina HTML specificata.
Se si vuole utilizzare un riferimento relativo, si deve usare un secondo tipo di costruttore che ha due parametri: il primo indica il valore restituito dal metodo *getDocumentBase*, mentre il secondo è il nome del file HTML. Il metodo **getDocumentBase** restituisce un riferimento dell'host e della directory su cui è memorizzata la pagina attualmente caricata. L'URL viene costruito concatenando l'indirizzo restituito dal metodo *getDocumentBase* con il riferimento relativo indicato come secondo parametro.
Per esempio:

```
try
{
  URL indirizzo = new URL(getDocumentBase(),"index.htm");
}
catch (MalformedURLException e)
{
  System.out.println("Errore nella creazione dell'URL.");
}
```

A ogni applet è associato un oggetto di classe *AppletContext* che può essere recuperato usando il metodo **getAppletContext()**. Disponendo di questo oggetto, si può richiamare il metodo *showDocument* per visualizzare una nuova pagina HTML. Per poter utilizzare la classe *AppletContext* si deve importare il package **java.applet**.
Per esempio, se si vuole visualizzare l'indirizzo creato precedentemente, si devono usare le seguenti istruzioni:

```
AppletContext ac = getAppletContext();
ac.showDocument(indirizzo);
```

Se la pagina Web richiesta deve essere aperta in una nuova finestra basta aggiungere un secondo parametro con l'indicazione del target "**_blank**":

```
ac.showDocument(indirizzo, "_blank");
```

PROGETTO 3

Creare un semplice menu di navigazione composto da tre pulsanti.

I tre pulsanti del menu aprono rispettivamente le pagine *index.htm*, *applicazioni.htm* e *applet.htm*. Questi link sono da considerarsi come riferimenti relativi rispetto alla directory in cui è memorizzata la pagina del menu.
Sia i pulsanti che i relativi URL sono gestiti con array di tre elementi, dichiarati nel seguente modo:

```
private URL indirizzo[] = new URL[3];
private JButton but[] = new JButton[3];
```

Il clic sui pulsanti viene gestito nel metodo *actionPerformed*, nello stesso modo con cui è stato gestito nelle applicazioni del capitolo precedente.
Il codice della pagina HTML e dell'applet sono riportati di seguito.

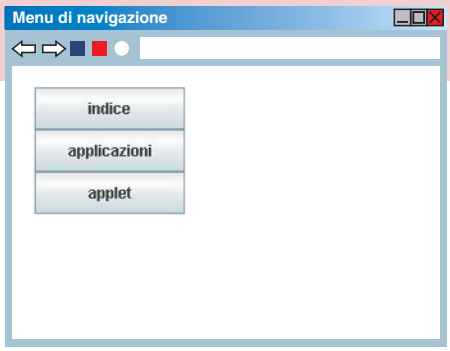
PAGINA WEB (*Menu.htm*)

```
<HTML>
<HEAD>
<TITLE>Menu di navigazione</TITLE>
</HEAD>
<BODY>
<APPLET CODE="Link.class" WIDTH="120" HEIGHT="100">
Applet non eseguibile.
</APPLET>
</BODY>
</HTML>
```

APPLET JAVA (*Link.java*)

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;

public class Link extends JApplet implements ActionListener
{
  private JPanel p = new JPanel();
  private URL indirizzo[] = new URL[3];
  private JButton but[] = new JButton[3];

  public void init()
  {
    try
    {
      indirizzo[0]=new URL(getDocumentBase(), "index.htm");
```

```
      indirizzo[1]=new URL(getDocumentBase(), "applicazioni.htm");
      indirizzo[2]=new URL(getDocumentBase(), "applet.htm");
    }
    catch (MalformedURLException e)
    {
      System.out.println("Errore nella creazione degli URL.");
    }
    but[0]=new JButton("indice");
    but[1]=new JButton("applicazioni");
    but[2]=new JButton("applet");

    // aggiunge le componenti e registra il loro gestore
    p.setLayout(new GridLayout(3,1));
    for(int i=0; i<3; i++)
    {
      p.add(but[i]);
      but[i].addActionListener(this);
    }
    setContentPane(p);
  }

  // controlla quale tasto viene premuto
  public void actionPerformed(ActionEvent e)
  {
    String bottone = e.getActionCommand();
    AppletContext ac = getAppletContext();

    if (bottone.equals("indice"))
    {
      ac.showDocument(indirizzo[0], "_blank");
      showStatus("Indice");
    }
    else if (bottone.equals("applicazioni"))
    {
      ac.showDocument(indirizzo[1], "_blank");
      showStatus("Applicazioni");
    }
    else if (bottone.equals("applet"))
    {
      ac.showDocument(indirizzo[2], "_blank");
      showStatus("Applet");
    }
  }
}
```



RESTRIZIONI E PROBLEMI DI SICUREZZA

La differenza principale tra un'applicazione Java e un'applet è legata alla capacità di quest'ultima di essere scaricata da Internet ed eseguita in locale. Questa operazione può essere molto pericolosa, infatti il programma scaricato potrebbe contenere dei virus oppure essere in grado di violare la sicurezza del sistema locale. Per evitare questi problemi, Java ha messo a punto un particolare meccanismo di **sicurezza**. Tutte le applet che vengono scaricate da Internet vengono filtrate da questo meccanismo; se l'applet tenta di eseguire operazioni non consentite, viene generata un'eccezione che ne provoca la terminazione.

Tra le operazioni che un'applet non può compiere, ci sono:
- leggere e scrivere i file locali
- aprire una connessione con un server diverso da quello che le ospita
- lanciare l'esecuzione di altri programmi sulla macchina locale
- invocare il metodo *System.exit()*
- accedere alle proprietà del sistema.

Questi divieti vengono fatti rispettare dai browser che, in questo modo, assicurano una certa sicurezza nei confronti delle applet scaricate dalla rete. È possibile anche modificare le impostazioni dei browser per non permettere in nessun caso l'esecuzione delle applet. L'*appletviewer* è meno restrittivo per quanto riguarda questi divieti: si possono impostare le sue proprietà per stabilire se è permesso eseguire o meno una particolare applet; si possono quindi superare le barriere imposte dal meccanismo di sicurezza.
In generale, anche le applet che vengono caricate dal file system locale vengono trattate in modo meno restrittivo, perché si ipotizza che l'utente che le sta eseguendo sia la stessa persona che le ha create e di conseguenza esse vengono considerate più sicure.

AUTOVERIFICA

Domande da 7 a 12 pag. 383-384
Problemi da 5 a 11 pag. 386

MATERIALI ON LINE

2. Chiamata delle funzioni JavaScript da un'applet

5 Gli eventi del mouse

Nelle applet è importante gestire l'interazione con gli **eventi del mouse**. Si possono verificare diversi eventi legati al movimento del mouse:
- l'evento che segnala quando il mouse entra in un'area dell'applet
- gli eventi di pressione del bottone del mouse, del clic e del suo rilascio
- l'evento generato quando il mouse viene spostato fuori dall'applet.

L'interfaccia che permette di gestire questi eventi si chiama **MouseListener**. L'applet può definire un gestore di eventi che implementa questa interfaccia.
L'elenco dei metodi della *MouseListener* usati per gestire gli eventi sono i seguenti:

```
public void mouseClicked(MouseEvent e) {}
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
```

Tutti i metodi hanno come parametro un oggetto di classe **MouseEvent**. Le informazioni più importanti che si possono ricavare da questo oggetto sono le coordinate del punto in cui si fa clic con il tasto del mouse: esse si ottengono usando i metodi **getX** e **getY** applicati all'oggetto *e*.
Una classe che si vuole registrare come gestore, deve implementare l'interfaccia *MouseListener* usando nella sua intestazione la parola chiave **implements**. Al suo interno si devono ridefinire tutti i cinque metodi precedenti, anche se non eseguono alcuna operazione.

PROGETTO 4

Creare un'applet che modifica il colore dello sfondo sulla base degli eventi del mouse.

L'applet è composta da un oggetto *JPanel* il cui colore di sfondo viene modificato con la seguente istruzione:

```
p.setBackground(Color.red);
```

Ogni evento del mouse causa il cambiamento del colore e la visualizzazione di un messaggio nella barra di stato del browser. Nel caso di un clic del mouse, la barra di stato visualizza le coordinate della posizione del mouse.

PAGINA WEB (*Colore.htm*)

```
<HTML>
<HEAD>
<TITLE>Piccola animazione</TITLE>
</HEAD>
<BODY>
<APPLET CODE=Anim.class WIDTH=300 HEIGHT=300>
Applet non eseguibile.
</APPLET>
</BODY>
</HTML>
```

APPLET JAVA (*Anim.java*)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Anim extends JApplet implements MouseListener
{
  private JPanel p = new JPanel();

  public void init()
  {
    addMouseListener(this);
    p.setBackground(Color.red);
    setContentPane(p);
  }
```

```
  // metodi che gestiscono gli eventi del mouse
  public void mouseClicked(MouseEvent e)
  {
    p.setBackground(Color.blue);
    showStatus("Coordinate: " + e.getX() + ", " + e.getY());
  }
  public void mousePressed(MouseEvent e)
  {
    p.setBackground(Color.black);
    showStatus("Impostato il colore NERO");
  }
  public void mouseReleased(MouseEvent e)
  {
    p.setBackground(Color.white);
    showStatus("Impostato il colore BIANCO");
  }
  public void mouseEntered(MouseEvent e)
  {
    p.setBackground(Color.green);
    showStatus("Impostato il colore VERDE");
  }
  public void mouseExited(MouseEvent e)
  {
    p.setBackground(Color.red);
    showStatus("Impostato il colore ROSSO");
  }
}
```

6 Disegni

Come abbiamo visto, le applet possono essere costruite usando le stesse componenti GUI usate per le applicazioni. Nel capitolo precedente sono state presentate molte di queste componenti. Esaminiamo ora con attenzione i metodi della classe **Graphics** con i quali si può gestire la grafica.
Le aree di disegno vengono create definendo una classe che estende **JPanel** e, in particolare, per modificare il loro contenuto bisogna ridefinire il metodo **paintComponent**.

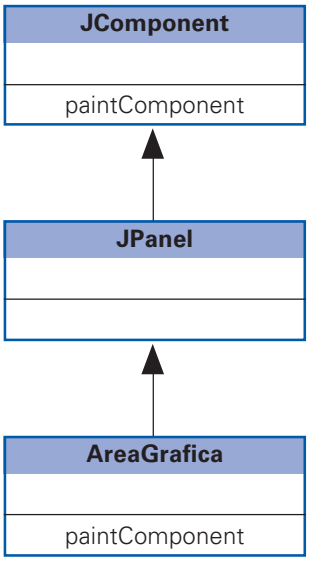
```
public void paintComponent(Graphics g)
{
  // metodi per disegnare nell'area
}
```

Il metodo *paintComponent* riceve come parametro un oggetto di classe *Graphics*. È proprio in questa classe che si trovano i metodi per gestire i colori, tracciare le linee e visualizzare i testi.

PROGETTO 5

Visualizzare una scritta all'interno di un'area di disegno.

L'area di disegno viene creata come una nuova classe che estende *JPanel* e esegue l'*override* del metodo *paintComponent* ereditato dalla classe *JComponent*. Il seguente diagramma mostra la relazione tra queste tre classi.



L'oggetto *area* viene creato come istanza della classe *AreaGrafica* e viene successivamente posizionato all'interno dell'applet con le seguenti istruzioni:

```
area = new AreaGrafica();
setContentPane(area);
```

Il codice della pagina HTML e dell'applet sono riportati di seguito.

PAGINA WEB (*Disegno.htm*)

```
<HTML>
<HEAD>
<TITLE>Area grafica </TITLE>
</HEAD>
<BODY>
<APPLET CODE=Disegno.class WIDTH=300 HEIGHT=300>
Applet non eseguibile.
</APPLET>
</BODY>
</HTML>
```

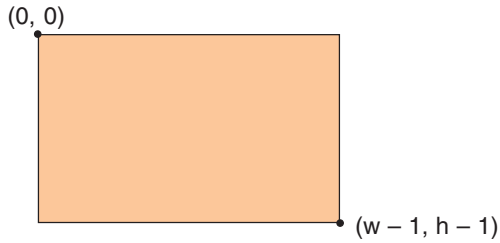
APPLET JAVA (*Disegno.java*)

```
import java.awt.*;
import javax.swing.*;
```

```
public class Disegno extends JApplet
{
  AreaGrafica area;

  public void init()
  {
    area = new AreaGrafica();
    setContentPane(area);
  }
}

class AreaGrafica extends JPanel
{
  public void paintComponent(Graphics g)
  {
    g.drawString("JAVA", 10,10);
  }
}
```

Si noti che nel file *Disegno.java* sono state definite due classi, una per l'applet e l'altra per l'area di disegno. Nel momento in cui si esegue la compilazione del file vengono generati i due file *Disegno.class* e *AreaGrafica.class*.

Il metodo *paintComponent* viene richiamato automaticamente dal sistema quando il pannello o l'applet devono essere visualizzati. È anche possibile richiedere che l'area di disegno venga ridisegnata usando il metodo **repaint()**. Questo metodo va utilizzato tutte le volte che vengono compiute delle modifiche e si vuole ridisegnare l'applet per aggiornarla.
L'area di disegno è una zona rettangolare le cui dimensioni possono essere espresse in *pixel*. All'interno di quest'area si possono individuare i singoli punti (o pixel) usando una coppia di numeri che rappresentano le coordinate del punto nel sistema di riferimento dello schermo. Il punto in alto a sinistra ha coordinate (0,0). Per sapere quali sono le coordinate del punto in basso a destra bisogna conoscere le dimensioni dell'area di disegno.
Le dimensioni di un'applet vengono stabilite nel file HTML, usando gli elementi WIDTH e HEIGHT. Queste dimensioni possono essere conosciute dall'applet usando il metodo **getSize()**. In particolare, la larghezza dell'area si ottiene usando **getSize().width**, mentre l'altezza con **getSize().height**. Entrambi i metodi restituiscono un numero intero.



In figura vengono mostrate le coordinate dei punti estremi dell'area di disegno, in cui le variabili *w* e *h* assumono i seguenti valori:

```
int w = getSize().width;
int h = getSize().height;
```

Il metodo **drawLine** della classe *Graphics* che permette di tracciare le linee è:

```
drawLine(int x1, int y1, int x2, int y2);
```

I quattro parametri rappresentano le coordinate dei due punti tra cui deve essere tracciata la linea. Il primo punto ha coordinate (x1,y1) mentre il secondo punto ha coordinate (x2,y2). Il colore usato per il tracciamento è quello attualmente impostato nella classe *Graphics*.

Il seguente esempio, basato sulle classi del progetto precedente, mostra un'applet che traccia le diagonali dell'area di disegno.

```
import java.awt.*;
import javax.swing.*;

public class Disegno2 extends JApplet
{
  AreaGrafica area;

  public void init()
  {
    area = new AreaGrafica();
    setContentPane(area);
  }
}

class AreaGrafica extends JPanel
{
  private int w, h;

  public void paintComponent(Graphics g)
  {
    w = getSize().width;
    h = getSize().height;
    g.drawLine(0, 0, w-1, h-1);
    g.drawLine(0, h-1, w-1, 0);
  }
}
```

Per disegnare un rettangolo si deve usare il metodo **drawRect**:

```
drawRect(int x, int y, int larghezza, int altezza);
```

I primi due parametri sono le coordinate del punto in alto a sinistra, il terzo parametro specifica la larghezza del rettangolo, mentre il quarto specifica l'altezza.

Di conseguenza il punto in basso a destra assumerà le coordinate (x+larghezza, y+altezza).

```
g.drawRect(10,10,50,60);
```

Il metodo *drawRect* disegna solo il bordo. Se si vuole disegnare un rettangolo con un colore di riempimento si deve usare il metodo **fillRect** che possiede gli stessi parametri del metodo precedente.
Il colore utilizzato è sempre quello attualmente impostato nella classe *Graphics*.

```
g.fillRect(10,10,50,60);
```

Le figure vengono disegnate usando il colore che è attualmente impostato come attributo della classe *Graphics*.
L'impostazione di un colore diverso viene fatta usando il metodo **setColor(*Color*)**, passando un oggetto della classe *Color*. Esistono alcuni colori predefiniti che sono stati elencati nel capitolo precedente: per esempio il bianco è indicato con l'attributo statico *Color.white*.
Oltre a questi, si possono definire nuovi colori. I colori gestiti da Java usano il formato **RGB**. In questo formato ogni colore è rappresentato da tre componenti: il rosso (**R**, *red*), il verde (**G**, *green*) e il blu (**B**, *blue*). Ognuna di queste componenti può assumere un valore numerico compreso tra 0 e 255. Un valore basso indica che la componente incide poco sul colore, mentre un valore alto significa che il colore usa molto quella componente. A ogni tripletta di valori viene fatto corrispondere un particolare colore.

Per esempio:

R	G	B	Colore
0	0	0	Nero
255	255	255	Bianco
255	0	0	Rosso
0	255	0	Verde
0	0	255	Blu
255	100	100	Rosso chiaro

Per definire un nuovo colore si deve creare un oggetto di classe *Color*. Il costruttore di questa classe utilizza tre parametri, ognuno dei quali specifica il valore delle componenti R, G e B.
Per esempio la creazione del colore rosso chiaro e la sua impostazione come colore corrente si realizzano con le seguenti istruzioni:

```
Color c = new Color(255, 100, 100);
g.setColor(c);
```

Per disegnare le figure circolari, una circonferenza oppure un'ellisse, si usa il metodo **drawOval** secondo la sintassi seguente:

```
drawOval(int x, int y, int larghezza, int altezza);
```

I parametri servono per definire il quadrato o il rettangolo dove viene inscritta la figura circolare. Se la larghezza è uguale all'altezza verrà tracciata una circonferenza, altrimenti verrà tracciata un'ellisse.
Per esempio, la seguente istruzione disegna un cerchio che ha il diametro di 20 pixel:

```
g.drawOval(0,0,20,20);
```

Il metodo **drawArc** consente di disegnare solo un arco di circonferenza o di ellisse:

```
drawArc(int x, int y, int larghezza, int altezza,
        int angIniziale, int angolo);
```

Il metodo ha sei parametri: i primi quattro hanno lo stesso significato di quelli del metodo *drawOval*; gli ultimi due indicano l'angolo da cui inizia l'arco e l'ampiezza dell'arco stesso. Il valore usato per indicare gli angoli è un numero intero che varia tra 0 e 360. Al valore 0 viene fatta corrispondere la posizione che nell'orologio hanno le ore tre. L'arco viene disegnato in senso antiorario.

Un arco di circonferenza di diametro 10 che parte da 45 gradi e arriva a 135 gradi viene disegnato con il seguente metodo:

```
g.drawArc(0, 0, 10, 10, 45, 90);
```

I metodi *drawOval* e *drawArc* disegnano solo il bordo delle figure circolari. Per disegnare le figure piene, si usano i corrispondenti metodi **fillOval** e **fillArc**, che hanno gli stessi parametri visti in precedenza.
Per inserire titoli o messaggi all'interno di un grafico si usa il metodo **drawString**, che, come già visto, riceve tre parametri: la stringa da visualizzare e le coordinate del punto di partenza della stringa.
La scritta viene mostrata usando il colore e il font correntemente impostati. Il colore può essere impostato usando il precedente metodo *setColor*, mentre per il font si usa il metodo **setFont*(Font)***, come già visto nel Progetto 3, applicato a una componente di tipo *JTextField*. Il parametro passato deve essere un oggetto di classe **Font**. Il costruttore di questa classe permette di impostare nuovi font specificando tre elementi:
- il tipo di font (uno tra *Arial*, *Helvetica*, *TimesRoman*, *Courier* e *Symbol*)
- lo stile (normale, corsivo e grassetto)
- la dimensione.

Per specificare lo stile si possono usare tre attributi statici: *Font.PLAIN* (normale), *Font.BOLD* (grassetto), *Font.ITALIC* (corsivo).
Per esempio le seguenti istruzioni impostano un tipo di font *TimesRoman*, normale e di dimensione 12:

```
Font personalizzato = new Font("TimesRoman", Font.PLAIN, 12);
g.setFont(personalizzato);
```

Dopo questa istruzione, tutti i metodi *drawString* useranno il nuovo font.

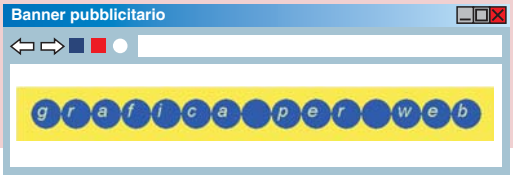
PROGETTO 6

Creare un banner pubblicitario con una scritta, separando tra loro i caratteri ed inserendo ciascuno di essi in un cerchio colorato. L'utente inoltre può attivare un link facendo clic sul banner.

I **banner** sono immagini o animazioni inserite, a scopo pubblicitario, di solito nella parte alta delle pagine Web.

PAGINA WEB (*Banner.htm*)

```
<HTML>
<HEAD>
<TITLE>Banner pubblicitario</TITLE>
</HEAD>
<BODY>
<CENTER>
<APPLET CODE="Banner.class" WIDTH="800" HEIGHT="100">
Applet non eseguibile.
</APPLET>
</CENTER>
</BODY>
</HTML>
```



APPLET JAVA (*Banner.java*)

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;

public class Banner extends JApplet implements MouseListener
{
  AreaGrafica area;
  URL indirizzo;

  public void init()
  {
    try
    {
      indirizzo = new URL("http://www.oracle.com");
    }
    catch (MalformedURLException e)
    {
      System.out.println("URL errato");
    }
    area = new AreaGrafica();
    addMouseListener(this);
    setBackground(Color.yellow);
    setContentPane(area);
  }

  // metodi che gestiscono gli eventi del mouse
  public void mouseClicked(MouseEvent e)
  {
    AppletContext ac = getAppletContext();
    ac.showDocument(indirizzo);
  }
  public void mousePressed(MouseEvent e) {}
  public void mouseReleased(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
}

class AreaGrafica extends JPanel
{
  private String scritta = "grafica per web";

  public void paintComponent(Graphics g)
  {
    g.setFont(new Font("Arial", Font.ITALIC, 30));
    Color testo = new Color(190, 240, 160);
    for(int i=0; i<scritta.length(); i++)
    {
      g.setColor(Color.blue);
      g.fillOval(25+50*i, 20, 50, 50);
      g.setColor(testo);
      g.drawString(""+ scritta.charAt(i), 35+50*i, 50);
    }
  }
}
```

Per inserire ciascuna lettera della scritta in un cerchio, è stata usato il metodo **charAt** che, come già visto nel Capitolo 4, estrae il carattere che si trova nella stringa alla posizione indicata come parametro (la prima posizione corrisponde al valore 0).

I metodi per disegnare le figure, descritti in precedenza, sono solo alcuni di quelli presenti nella classe *Graphics*. Esistono dei metodi più sofisticati che consentono di disegnare rettangoli con il bordo arrotondato, rettangoli con effetti tridimensionali, poligoni con più lati e così via.

MATERIALI ON LINE

3. Grafici statistici

7 Immagini nelle applet

In un'applet possono essere visualizzate le immagini che vengono caricate dalla rete. I formati che sono supportati da Java sono gli stessi di quelli usati normalmente in Internet per le pagine Web: *gif*, *jpeg* e *png*.

Le immagini vengono gestite dagli oggetti della classe **Image** che è contenuta nella libreria *java.awt*. Per caricare un'immagine in un'applet si usa il metodo **getImage** fornito dalla classe *Applet*. Si devono specificare due parametri che sono simili a quelli usati per il costruttore della classe URL: il primo parametro indica un indirizzo assoluto, mentre il secondo indica il nome del file relativo al primo indirizzo.
Per esempio:

```
Image img = getImage(getDocumentBase(), "foto.jpg");
```

In questo caso viene richiesta l'immagine *foto.jpg* che è posizionata nella stessa directory del file HTML.

Solitamente il caricamento di tutte le immagini utilizzate da un'applet viene fatto all'interno del metodo *init*; quando il caricamento è completato, negli oggetti di classe *Image* sono memorizzate le immagini. Per visualizzarle nell'applet bisogna usare il metodo **drawImage** all'interno del metodo *paintComponent*. È possibile richiamare il metodo *drawImage* in due formati diversi, usando 4 o 6 parametri.
I parametri del primo formato, comuni anche al secondo, sono:

- l'immagine
- il punto (x,y) dove posizionare l'immagine (punto in alto a sinistra)
- il riferimento a un particolare oggetto che ne controlla il caricamento. Quest'ultimo valore è solitamente impostato a *this*, perché è già previsto all'interno dell'applet.

Per esempio, il seguente metodo *paintComponent* visualizza l'immagine *img* nel punto (0,0):

```
public void paintComponent(Graphics g)
{
  g.drawImage(img, 0, 0, this);
}
```

Questo metodo disegna l'immagine usando le sue dimensioni originali.

Il secondo formato di *drawImage* consente di allungare, schiacciare o scalare un'immagine: i due parametri aggiuntivi indicano i valori dell'altezza e della larghezza dell'immagine.
Per esempio, volendo visualizzare l'immagine *img* nel punto (0,0), forzandola ad assumere una larghezza di 100 pixel e un'altezza di 50, si deve scrivere il seguente metodo:

```
public void paintComponent(Graphics g)
{
  g.drawImage(img, 0, 0, 100, 50, this);
}
```

PROGETTO 7

Creare un banner pubblicitario con un'immagine: facendo clic su di esso si apre la pagina di un sito Web.

Il problema viene risolto usando come traccia il progetto del paragrafo precedente.
In questo caso il contenuto del banner è un'immagine in formato *jpg*.

L'immagine da visualizzare si trova nella sottodirectory *images*.
L'immagine viene acquisita all'interno del metodo *init* dell'applet e viene passata come parametro al costruttore di *AreaGrafica*.
Il metodo *drawImage* deforma l'immagine per adattarla alle dimensioni dell'applet.
Le dimensioni originali dell'immagine sono 500x180 pixel.

Il codice della pagina HTML e dell'applet sono riportati di seguito.

PAGINA WEB (*Banner2.htm*)

```
<HTML>
<HEAD>
<TITLE>Banner pubblicitario</TITLE>
</HEAD>
<BODY>
<CENTER>
<APPLET CODE="Banner2.class" WIDTH="800" HEIGHT="100">
Applet non eseguibile.
</APPLET>
</CENTER>
</BODY>
</HTML>
```

APPLET JAVA (*Banner2.java*)

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;

public class Banner2 extends JApplet implements MouseListener
{
  AreaGrafica area;
  URL indirizzo;

  public void init()
  {
    // URL del link sul banner
    try
    {
      indirizzo = new URL("http://www.ferrariworld.com");
    }
    catch (MalformedURLException e)
    {
      System.out.println("URL errato");
    }

    // immagine del banner
    Image foto = getImage(getDocumentBase(), "images/Ferrari.jpg");
    area = new AreaGrafica(foto);
    addMouseListener(this);
    setBackground(Color.yellow);
    setContentPane(area);
  }

  // metodi che gestiscono gli eventi del mouse
  public void mouseClicked(MouseEvent e)
  {
    AppletContext ac = getAppletContext();
    ac.showDocument(indirizzo);
  }
  public void mousePressed(MouseEvent e) {}
  public void mouseReleased(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
}

class AreaGrafica extends JPanel
{
  private Image img;

  public AreaGrafica(Image img)
  {
    this.img = img;
  }

  public void paintComponent(Graphics g)
  {
    g.drawImage(img, 0, 0, 800, 200, this);
  }
}
```

8 Riproduzione di suoni

Il package *java.applet* fornisce un supporto di base per la riproduzione di file sonori. Attualmente, i file sonori interpretabili in Java devono avere uno dei seguenti formati: AIFF, AU, WAV, MIDI, RMF. Altri formati possono essere convertiti in uno di questi con programmi di utilità per i file audio.
I file sonori sono gestiti dall'interfaccia **AudioClip**. Per utilizzare un file sonoro si usa il metodo **getAudioClip** ereditato dalla classe *Applet*. Questo metodo riceve due parametri: il primo è l'URL relativo alla posizione del file e il secondo è il nome del file. Il metodo restituisce un oggetto *AudioClip*.

```
AudioClip suono;
String nome = "Vivaldi.wav";
suono = getAudioClip(getDocumentBase(), nome);
```

Il metodo *getDocumentBase* restituisce un riferimento dell'host e della directory su cui è memorizzata l'applet. Questo significa che il file sonoro *Vivaldi.wav* deve essere posizionato nella stessa directory della classe dell'applet.
Il file sonoro viene realmente caricato solo quando l'applet avvia la riproduzione.

L'interfaccia *AudioClip* definisce i seguenti metodi:
- **loop**: riproduce ripetutamente il file;
- **play**: riproduce il file una sola volta, ogni volta viene fatto ripartire dall'inizio;
- **stop**: interrompe la riproduzione del file.

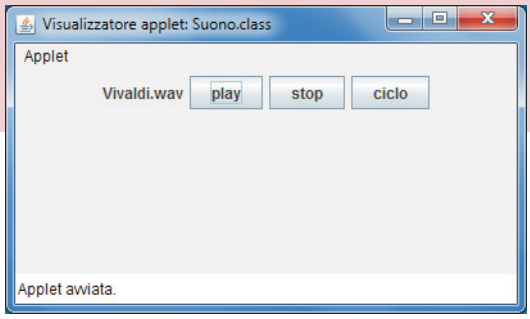
PROGETTO 8

Realizzare un'applet per caricare e riprodurre un file sonoro.

Essa contiene una semplice interfaccia, composta da tre pulsanti, per gestire le tre operazioni relative ad un file sonoro: riproduzione singola, riproduzione ciclica e interruzione. Sullo schermo compare anche il nome del file audio che viene riprodotto.

PAGINA WEB (*Suono.htm*)

```
<HTML>
<HEAD>
<TITLE>Riproduzione </TITLE>
</HEAD>
<BODY>
<APPLET CODE="Suono.class" WIDTH="300" HEIGHT="100">
Applet non eseguibile.
</APPLET>
</BODY>
</HTML>
```



APPLET JAVA (*Suono.java*)

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Suono extends JApplet implements ActionListener
{
  AudioClip suono;
  JButton b1,b2,b3;
  JLabel l1;
  JPanel p;

  public void init()
  {
    p = new JPanel();
    String nome = "Vivaldi.wav";
    suono = getAudioClip(getDocumentBase(), nome);
    l1 = new JLabel(nome, Label.RIGHT);
    b1 = new JButton("play");
    b2 = new JButton("stop");
    b3 = new JButton("ciclo");

    b1.addActionListener(this);
    b2.addActionListener(this);
    b3.addActionListener(this);

    p.add(l1);
    p.add(b1);
    p.add(b2);
    p.add(b3);
    setContentPane(p);
  }

  public void actionPerformed(ActionEvent e)
  {
    String bottone = e.getActionCommand();
    if (bottone.equals("play"))
    {
      suono.play();
    }

    if (bottone.equals("stop"))
    {
      suono.stop();
    }

    if (bottone.equals("ciclo"))
    {
      suono.loop();
    }
  }
}
```

CONVERSIONE DI APPLICAZIONI IN APPLET

Le applicazioni Java possono essere convertite in applet attraverso poche modifiche. Si consideri, per esempio l'applicazione Java, presentata nel Progetto 11 dell'inserto sulle interfacce grafiche, precedente a questo capitolo, per richiedere cognome e nome di una persona e controllare i dati inseriti.

Le operazioni da compiere per costruire l'applet sono le seguenti:

- prima di tutto la classe *DatiAnagrafici* con il *main* non serve più, perché l'applet usa come frame la finestra del browser;
- deve essere eliminata l'eventuale chiamata al metodo *setSize*, perché le dimensioni della finestra sono impostate nella pagina Web, con il tag <APPLET> e gli attributi WIDTH e HEIGHT;
- deve essere eliminata la chiamata al metodo *setTitle*, perché il titolo dell'applet è il titolo assegnato alla pagina Web con il tag <TITLE> del linguaggio HTML;
- nella classe *Anagrafe* occorre sostituire *JFrame* con *JApplet* e impostare la classe come *public*, in modo che l'applet possa essere avviata dal browser; quindi la riga:

```
class Anagrafe extends JFrame implements ActionListener
```

diventa:

```
public class Anagrafe extends JApplet implements ActionListener
```

- occorre eliminare la chiamata al metodo che gestisce l'evento della chiusura della finestra, perchè l'esecuzione di un'applet termina quando l'utente chiude la finestra del browser; quindi le seguenti righe dell'applicazione devono essere eliminate:

```
// uscita dal programma quando la finestra viene chiusa
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

- occorre sostituire il costruttore della classe *Anagrafe* con il metodo *init*; l'applet, infatti, può avere un suo costruttore, ma i parametri possono essere ricevuti dalla pagina Web tramite il metodo *getParameter* solo all'interno del metodo *init*. Quindi la riga:

```
public Anagrafe()
{
....
}
```

deve essere sostituita con:

```
public void init()
{
....
}
```

Le considerazioni precedenti possono essere viste nel seguente progetto che, per comodità di lettura, ripropone lo stesso problema dell'inserto precedente a questo capitolo, con il risultato della conversione dell'applicazione in applet.

PROGETTO 9

Creare una maschera nel browser per richiedere all'utente il cognome e il nome.

I dati da inserire devono essere considerati obbligatori, per cui se l'utente lascia una delle caselle di input vuote, il programma visualizza una finestra con un messaggio di avvertimento, altrimenti visualizza una finestra di informazione che riassume i dati inseriti.

PAGINA WEB (*RichiestaDati.htm*)

```
<HTML>
<HEAD>
<TITLE>Dati anagrafici</TITLE>
</HEAD>
<BODY>
<CENTER>
<APPLET CODE="Anagrafe.class" WIDTH="400" HEIGHT="100">
Applet non eseguibile.
</APPLET>
</CENTER>
</BODY>
</HTML>
```

APPLET JAVA (*Anagrafe.java*)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Anagrafe extends JApplet implements ActionListener
{
  private JPanel p = new JPanel();

  private JPanel dati = new JPanel();
  private JLabel l1 = new JLabel ("Cognome ", JLabel.RIGHT);
  private JTextField cognome = new JTextField(10);
  private JLabel l2 = new JLabel ("Nome ", JLabel.RIGHT);
  private JTextField nome = new JTextField(10);

  private JPanel bottoni = new JPanel();
  private JButton annulla = new JButton("Annulla");
  private JButton ok = new JButton("OK");

  public void init()
  {
    p.setLayout(new BorderLayout());
```

```
    // imposta le stringhe di comando
    // e gli ascoltatori per gli eventi sui pulsanti
    annulla.setActionCommand("annulla");
    ok.setActionCommand("ok");
    annulla.addActionListener(this);
    ok.addActionListener(this);

    // inserisce le componenti
    dati.add(l1);
    dati.add(cognome);
    dati.add(l2);
    dati.add(nome);
    bottoni.add(annulla);
    bottoni.add(ok);
    p.add(dati, "Center");
    p.add(bottoni,"South");

    // aggiunge il pannello al frame
    this.getContentPane().add(p, BorderLayout.CENTER);
  }

  public void actionPerformed(ActionEvent e)
  {
    if ("annulla".equals(e.getActionCommand()))
    {
      cognome.setText("");
      nome.setText("");
    }
    if ("ok".equals(e.getActionCommand()))
    {
      String cognomeLetto = cognome.getText();
      String nomeLetto = nome.getText();
      if ((cognomeLetto.equals("")) || (nomeLetto.equals("")))
      {
        JOptionPane.showMessageDialog(this, "I dati sono obbligatori",
            "Attenzione", JOptionPane.WARNING_MESSAGE);
      }
      else
      {
        String messaggio = "Dati inseriti: " + cognomeLetto +
            " " + nomeLetto;
        JOptionPane.showMessageDialog(this, messaggio,
            "Conferma", JOptionPane.INFORMATION_MESSAGE);
      }
    }
  }
}
```

AUTOVERIFICA

Domande da 13 a 19 pag. 384-385
Problemi da 12 a 25 pag. 386-387

MATERIALI ON LINE

4. Grafico della parabola

DOMANDE

Applet

1 Quali di queste affermazioni, riferite ad un'applet, sono vere (V) e quali false (F)?
a) È uguale ad un'applicazione Java V F
b) Può essere eseguita in un browser Web V F
c) Deve essere definita come sottoclasse della classe *Applet* V F
d) Può essere considerata come un pannello V F
e) Viene inserita in una pagina HTML con il tag <IMG> V F

2 Quale di questi tag è correttamente dichiarato?
a) <APPLET CODE="graf.class"></APPLET>
b) <APPLET CODE="graf.class" WIDTH="300"></APPLET>
c) <APPLET CODE="graf.class" WIDTH="300" HEIGHT="300"></APPLET>
d) <APPLET CODEBASE="graf.class" WIDTH="300" HEIGHT="300"></APPLET>

3 Quale dei seguenti metodi viene eseguito subito dopo che l'applet è stata caricata?
a) init
b) start
c) stop
d) destroy

4 Quali di queste affermazioni, riferite all'*appletviewer*, sono vere (V) e quali false (F)?
a) È un minibrowser V F
b) Esegue anche le applicazioni Java V F
c) Interpreta tutti i tag dell'HTML V F
d) Interpreta solo il tag <APPLET> V F
e) Viene eseguito dal *Prompt dei comandi* V F

5 Quale tra i seguenti attributi del tag <APPLET> specifica l'URL dove è posizionata l'applet?
a) CODEBASE
b) ALT
c) NAME
d) ALIGN

6 Quale tra i seguenti attributi del tag <APPLET> contiene il messaggio da visualizzare quando il browser non può eseguire le applet?
a) CODEBASE
b) NAME
c) ALT
d) ALIGN

Interazione col browser

7 Qual è il metodo che legge il valore di un parametro in un'applet?
a) setParameter
b) getParameter
c) Param
d) ParamValue

AUTOVERIFICA

8 Quali delle seguenti frasi rappresenta meglio la definizione di *plug-in*?
a) Un'applet che esegue altre applet.
b) Un piccolo programma aggiuntivo per il browser.
c) Un minibrowser.
d) Un'applicazione Java.

9 Quale tag HTML deve essere usato al posto di <APPLET> per inserire in una pagina Web un oggetto multimediale o eseguibile?
a) <OBJECTAPPLET>
b) <JAPPLET>
c) <APPLETOBJECT>
d) <OBJECT>

10 Quale delle seguenti frasi rappresenta l'oggetto restituito dal metodo *getDocumentBase*?
a) Un URL indicante l'indirizzo della pagina attualmente caricata.
b) Il nome della pagina attualmente caricata.
c) Un URL indicante il nome del server e della directory in cui è memorizzata la pagina.
d) La directory dove si trovano i file di installazione di Java.

11 Completa le frasi seguenti utilizzando uno tra i metodi elencati alla fine della domanda.
a) Il metodo ..... scrive i messaggi sulla barra di stato.
b) Il metodo ..... visualizza una nuova pagina HTML.
c) La finestra ..... mostra i messaggi in output dell'esecuzione di un'applet.
d) La classe ..... specifica l'indirizzo di una pagina.
setFont, setColor, paint, repaint, drawString, Java Console, getImage, URL, getSize, showDocument, showStatus, getAppletContext

12 Quali delle seguenti affermazioni riferite alla sicurezza nell'uso delle applet sono vere (V) e quali false (F)?
a) Un'applet viene scaricata da Internet ed eseguita in locale V F
b) Un'applet può leggere e scrivere i file locali V F
c) È possibile modificare le impostazioni del browser per non permettere in nessun caso l'esecuzione delle applet V F
d) Il programma *appletviewer* è più restrittivo del browser per quanto riguarda la sicurezza nell'esecuzione di un'applet V F

Grafica e Suoni

13 Quale dei seguenti metodi non fa parte dell'interfaccia *MouseListener*?
a) mouseDrag
b) mouseClicked
c) mousePressed
d) mouseExited

14 Quali di queste affermazioni, riferite alla classe *Graphics*, sono vere (V) e quali false (F)?
a) Identifica una componente V F
b) Il parametro del metodo *paintComponent* appartiene a questa classe V F
c) Contiene i metodi per disegnare le figure V F
d) Contiene il metodo *setColor* V F
e) Contiene il metodo *getSize* V F

15 Associa a ciascun metodo della colonna a sinistra la corrispondente definizione scegliendo dalla colonna a destra.

a) drawLine	1) rettangoli con l'area colorata
b) drawRect	2) colore per i disegni
c) fillRect	3) traccia archi
d) setColor	4) settore circolare con area colorata
e) drawOval	5) scrivere caratteri
f) drawArc	6) tracciare linee
g) fillOval	7) font per i caratteri
h) fillArc	8) cerchio o ellisse con area colorata
i) drawString	9) cerchio o ellisse
l) setFont	10) tracciare rettangoli

16 Completa le frasi seguenti utilizzando uno tra i metodi elencati alla fine della domanda.

a) Il metodo ..... stabilisce che l'area di disegno deve essere ridisegnata.

b) Il metodo ..... imposta il colore usato per disegnare le figure e il testo.

c) Il metodo ..... disegna un arco di circonferenza o di ellisse.

d) Il metodo ..... inserisce titoli e messaggi all'interno di un grafico.

setFont, setColor, paint, repaint, drawString, drawImage, getImage, getSize, showDocument, showStatus, getAppletContext

17 Quale dei seguenti metodi carica un'immagine in un'applet?
a) showImage
b) setImage
c) drawImage
d) getImage

18 Quale dei seguenti metodi *non* è presente nell'interfaccia *AudioClip*?
a) rec
b) loop
c) play
d) stop

19 Quale classe da estendere deve essere sostituita in un'applicazione per convertirla in applet?
a) sostituire JApplet con JFrame
b) sostituire JFrame con JApplet
c) sostituire JFrame con JPanel
d) sostituire JApplet con JPanel

PROBLEMI

Applet

1 Sapendo che il file principale di un'applet è *titolo.class*, mostrare il tag HTML che consente di visualizzare l'applet in un'area di dimensione 100x300.

2 Creare un'applet che mostra tre caselle di testo disposte una sotto l'altra.

3 Costruire un'applet con un'etichetta del titolo e sei caselle di testo contenenti sei numeri generati casualmente tra 1 e 90, per simulare l'estrazione dell'Enalotto.

4 Scrivere un'applet che calcola la somma di due numeri. I due numeri vengono inseriti in due caselle di testo e il risultato è visualizzato usando un'altra componente.

Interazione col browser

5 Definire, usando il linguaggio HTML, un parametro da passare ad un'applet. Il parametro ha come nome *peso* e il suo valore è 30.

6 Acquisire una stringa contenuta in un parametro e visualizzarla nell'applet in un'etichetta.

7 Una pagina Web richiede l'esecuzione di un'applet passando come parametri una stringa e un numero. L'applet scrive la stringa all'interno di un'etichetta usando il font Times New Roman e la grandezza dei caratteri uguale al numero ricevuto.

8 Creare un'applet che scrive un messaggio sulla Java Console.

9 Scrivere un'applet che consente di gestire, con un menu verticale di pulsanti, un elenco di 5 link a pagine web.

10 Scrivere un'applet che consente di gestire, con una barra di menu di etichette disposte in orizzontale, un elenco di link a pagine web.

11 Costruire una pagina Web che richiede l'esecuzione di un'applet, usando il tag <OBJECT> e verificarne il funzionamento con diversi browser.

Grafica e suoni

12 Ridefinire il metodo *mouseEntered* per visualizzare sulla barra di stato del browser il messaggio "Inserisci i valori nella casella di testo".

13 Scrivere un'applet che tiene traccia degli eventi generati dal mouse scrivendo sulla Java Console quale evento si verifica in ogni istante. Per esempio, se il pulsante del mouse viene premuto all'interno dell'applet, sulla Java Console deve comparire la scritta "Il pulsante del mouse è stato premuto". Se il pulsante del mouse viene rilasciato, si deve stampare la scritta "Il pulsante del mouse è stato rilasciato". Si proceda così anche con gli altri eventi.

14 Dati i punti P1(10,10) e P2(100,50), scrivere il metodo per tracciare una linea che unisce i due punti e il metodo che disegna un rettangolo avente come spigoli opposti i due punti.

15 Indicare il metodo con cui si può disegnare una circonferenza che ha centro nel punto di coordinate (50,50) e il diametro di 20 pixel.

16 Visualizzare in un'applet la stringa *Benvenuto* usando il seguente tipo di font: Courier, grassetto e dimensione 40.

17 Creare un'applet che mostra la scritta *Italia* all'interno di una cornice composta da tre rettangoli di colore verde, bianco e rosso.

18 Stabilire qual è il colore che ha le seguenti componenti: R=150, G=150, B=150.

19 Scrivere un'applet che disegna un quadrato avente un vertice nel punto (10,10) e il lato uguale a 100.

20 Creare un banner per una pagina Web contenente un'immagine e una scritta pubblicitaria.

21 Creare un'applet di dimensione 200x200 e riempirla usando una sola immagine di dimensioni 20x20. Occorre visualizzare tante copie della stessa immagine, una vicino all'altra, per riempire tutto lo sfondo dell'applet.

22 Realizzare un'applet per simulare una raccolta musicale in cui la riproduzione avviene scegliendo i brani da una casella combinata.

23 Realizzare un'applet che reagisce ai movimenti del mouse nel seguente modo: quando il mouse entra nell'applet viene attivata la riproduzione di un suono che si interrompe non appena il mouse viene spostato fuori dall'applet. Il clic del mouse serve per cambiare il brano.

24 Convertire un'applicazione Java, tra quelle realizzate nell'inserto sulle interfacce grafiche, precedente a questo capitolo, in un'applet.

25 Convertire un'applet in un'applicazione Java.

Focus notes

THE APPLET LIFECYCLE

Every applet can live only within a Web browser. In order to insert an applet into a Web page, a specific tag has to be used. The HTML <APPLET> tag informs the browser that an HTML page contains an applet.

The Web browser manages an *Applet Lifecycle* through four methods:
- *init()*: called once, when the Applet is first loaded;
- *start()*: called when the Applet becomes visible;
- *stop()*: called when the Applet becomes invisible;
- *destroy()*: called once, when the Applet is closed.

These methods represent the lifecycle of an Applet. Both the start and the stop methods can be called multiple times, when the applet is shown or is hidden by other windows.

THE APPLET TAG

The APPLET tag is made of several attributes (mandatory and optional). These are shown in the following template. Below, you can find a description of each attribute.

```
<APPLET CODE    = appletFile
        ALT     = text
        NAME    = appletName
        WIDTH   = pixels
        HEIGHT  = pixels
        ALIGN   = alignment
        VSPACE  = pixels
        HSPACE  = pixels>
    <PARAM NAME = param1 VALUE = v1>
    <PARAM NAME = param2 VALUE = v2>
This browser doesn't support Applets.
</APPLET>
```

The only mandatory attributes are CODE, WIDTH, and HEIGHT.
- CODE: is the name of the *.class* file, containing the applet;
- ALT: is an optional text that is shown when the browser recognizes the APPLET tag but can't run applets;
- NAME: is an optional name, used to identify other applets within the same web page;
- WIDTH, HEIGHT: define a rectangular area used by the applet;
- ALIGN: is an optional attribute and can be one of these keywords: left, right, top, texttop, middle, absmiddle, baseline, bottom, absbottom;
- VSPACE, HSPACE: are optional attributes and are used to define an empty space around the applet;
- PARAM: defines a parameter for the applet.

Focus notes

Glossary

appletviewer
A command line program to run applets outside a web browser.

height
A vertical distance along the y-axis.

Java Console
A debugging tool for applets.

plug-in
An additional component for browsers, enabling extra functionality.

RGB
A model for representing the colors on a computer.

URL
Address of an Internet resource.

WAV
A file format for storing an audio file.

width
A horizontal distance along the x-axis.

Acronyms

AWT	Abstract Window Toolkit
HTML	HyperText Markup Language
IE	Internet Explorer
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
RGB	Red, Green, Blue
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WWW	World Wide Web

SCHEDA DI AUTOVALUTAZIONE

CONOSCENZE

- ☐ Caratteristiche generali di un'applet
- ☐ Classi Applet e JApplet
- ☐ Metodi della classe Applet
- ☐ Attivazione dell'applet da una pagina Web
- ☐ Passaggio di parametri all'applet
- ☐ Differenza tra il tag <applet> e il tag <object> nella pagina Web
- ☐ Interazione dell'applet con il browser
- ☐ Restrizioni e problemi di sicurezza nell'uso delle applet
- ☐ Differenza tra applicazione e applet Java
- ☐ Eventi del mouse
- ☐ Metodi per disegni
- ☐ Metodi per le immagini

ABILITÀ

- ☐ Usare il programma appletviewer per il collaudo delle applet
- ☐ Creare un'applet con la gestione dei suoi metodi principali
- ☐ Scrivere una pagina Web che richiama l'esecuzione di un'applet
- ☐ Scrivere una pagina Web che passa i parametri a un'applet
- ☐ Usare il tag <object> nella pagina Web
- ☐ Gestire i link all'interno di un'applet
- ☐ Convertire un'applicazione Java in un'applet
- ☐ Inserire all'interno dell'applet la gestione degli eventi del mouse
- ☐ Inserire scritte e forme grafiche
- ☐ Creare un banner pubblicitario
- ☐ Inserire immagini

SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 539

7

parte quarta

Applicazioni Web

Accesso ai database con JDBC

OBIETTIVI DI APPRENDIMENTO

In questo capitolo imparerai ad usare gli strumenti Java per l'accesso ai database.

Sarai in grado di effettuare la connessione al database e le operazioni di manipolazione e interrogazione.

I driver per la connessione al database

La tecnologia JDBC

Manipolazione dei dati

Interrogazioni

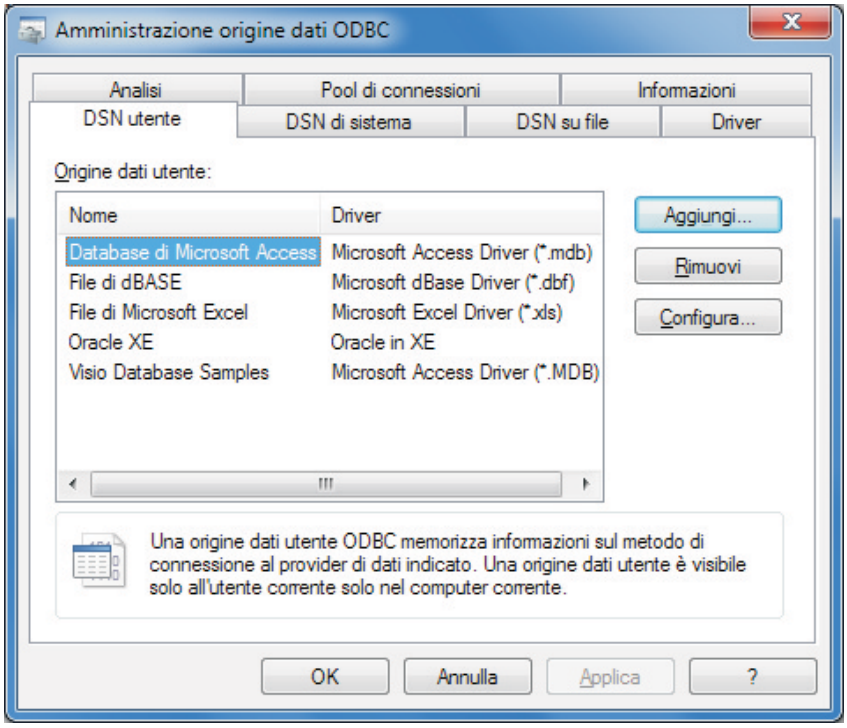
1 I driver per la connessione al database

In informatica il termine **driver** indica, in generale, un programma che consente a una periferica hardware di comunicare con l'unità centrale di un computer: il programma driver ritrova i dati richiesti e li trasferisce al programma richiedente. Analogamente, il **driver per database** è un programma standard che consente di trasferire i dati presenti in un database: quando un programma vuole accedere al database creato con un altro applicativo, deve utilizzare il driver per quel database.
ODBC (*Open Database Connectivity*) è l'interfaccia software standard in ambiente Windows che consente ai programmatori di scrivere in modo semplice le applicazioni per connettersi ai database e ritrovare i dati in essi contenuti. I database possono essere creati con prodotti DBMS diversi: ODBC possiede i driver software per database ed evita quindi di dover scrivere un programma diverso per ciascun tipo di database.

Per visualizzare le proprietà di ODBC in Windows e per fissarne le impostazioni, occorre fare le seguenti operazioni:

1. Nel *Pannello di controllo* scegliere *Sistema e sicurezza, Strumenti di amministrazione* e infine fare doppio clic sull'icona *Origine dati (ODBC)*.

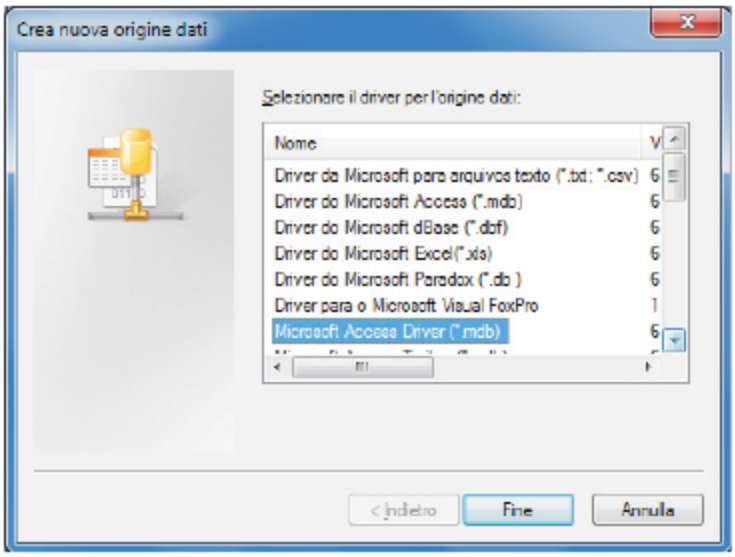
2. Fare clic sulla scheda *DSN utente*, *DSN di sistema* o *DSN su file*, a seconda del tipo di origine dati.



3. Per modificare la definizione di un'origine dati esistente:
- selezionare l'origine dati dall'elenco,
- scegliere il pulsante *Configura*,
- completare le finestre di dialogo.

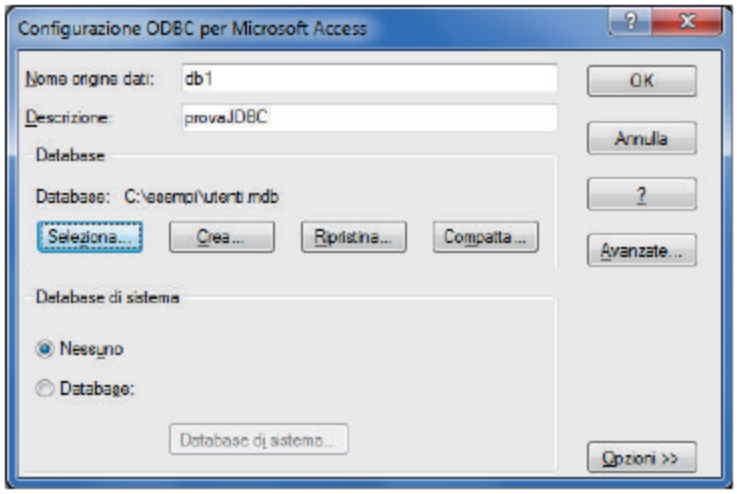
4. Per definire una nuova origine dati per un driver installato, nella finestra iniziale, scegliere il pulsante *Aggiungi*; nella nuova finestra che si apre (*Crea nuova origine dati*) scegliere il driver adatto per il database e poi fare clic sul pulsante *Fine*.

La sigla **DSN** (*Data Source Name*) indica il nome della sorgente di dati; è il nome che identifica il database quando viene utilizzato da un'applicazione Web in Internet o in una Intranet aziendale.



L'origine dei dati può essere:

- DSN utente (*User DSN*) per fissare una sorgente di dati per uno specifico utente;
- DSN di sistema (*System DSN*) per le sorgenti di dati disponibili per molti utenti che accedono allo stesso computer di rete;
- DSN su file (*File DSN*) per configurare un DSN come file con estensione *.dsn* che può essere condiviso da più utenti che dispongono dello stesso driver installato.

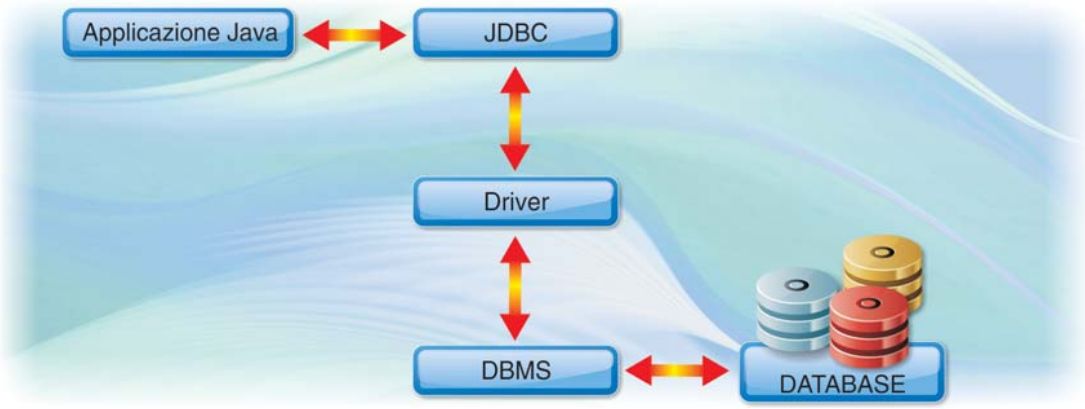


MATERIALI ON LINE

1. Richiami sui database e il linguaggio SQL

2 La tecnologia JDBC

JDBC (*Java DataBase Connectivity*) mette a disposizione una libreria di classi Java per interfacciare l'accesso ai database che usano il linguaggio standard SQL, con l'obiettivo di fornire un'uniformità di accesso ai prodotti DBMS relazionali.
Attraverso JDBC è possibile richiamare comandi SQL direttamente, fornendo la base per costruire applicazioni per l'accesso ai database.



Secondo la terminologia di Java il software JDBC è classificato come una **API** (*Application Programming Interface*), perché fornisce un'interfaccia analoga a quella offerta dall'API per i sistemi operativi moderni: in questo caso fornisce i driver ai database per la connessione, l'esecuzione dei comandi SQL e le transazioni verso i database gestiti con i più diffusi prodotti DBMS.
L'API JDBC è contenuta nel JDK di Oracle.
JDBC comprende anche il **JDBC-ODBC Bridge**, un ponte verso i driver standard ODBC, che rende possibile la connessione ai database creati con prodotti DBMS molto diffusi, tra i quali *Microsoft Access*.

Le operazioni di manipolazione e di interrogazione possono essere eseguite su un database locale usando **applicazioni Java** oppure sul database di un server di rete, usando le tecnologie del Web.
I programmi che usano JDBC devono specificare l'uso del package **java.sql** del linguaggio Java e quindi devono contenere l'istruzione

```
import java.sql.*;
```

La classe **DriverManager** tiene traccia dei driver disponibili e li gestisce stabilendo la connessione tra un driver e un particolare database.
Il programmatore carica il driver esplicitandolo con il comando

Class.forName("*nome del driver*"**)**

indicando il nome della specifica classe del driver JDBC, cioè la classe che consente a Java di accedere al database.
Per esempio:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

I comandi e i nomi delle classi sono *case-sensitive*, si faccia quindi attenzione all'uso delle lettere minuscole e maiuscole.

La classe indicata come parametro contiene un driver fornito con JDK per accedere ai database che utilizzano lo standard ODBC (tra cui Access) e quindi è in grado di convertire le richieste al DBMS dal formato JDBC a quello ODBC.
È buona norma inserire la **gestione delle eccezioni** nella fase di caricamento del driver utilizzando la seguente struttura:

```
try
{
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch(ClassNotFoundException e)
{
  System.err.print("ClassNotFoundException: ");
  System.err.println(e.getMessage());
}
```

L'eccezione **ClassNotFoundException** viene rilevata se non è stata trovata la classe del driver. Usando il metodo **getMessage()** si ottiene una descrizione completa dell'eccezione che si è verificata.
Il programma deve poi richiedere alla classe **DriverManager** di creare la connessione a un particolare database specificando il nome del database, ed eventualmente il nome dell'utente (*username*) e la *password* per l'autorizzazione nell'accesso.
Il nome del database è specificato all'interno di una stringa che contiene tre informazioni:
- protocollo
- sottoprotocollo
- nome del database secondo il formato previsto dal driver.

Per esempio:

```
String url = "jdbc:odbc:db1";
```

In questa assegnazione viene indicato il protocollo *jdbc*, il sottoprotocollo *odbc* (driver ODBC) e il nome *db1* definito come DSN nel caso si utilizzi ODBC.
La stringa viene in genere chiamata **URL** di JDBC perché richiama il concetto di indirizzo Internet: in effetti identifica il database attraverso il protocollo *jdbc*, così come un browser distingue un sito Internet tramite i protocolli *http* o *ftp*.
Il *DriverManager* controlla che ci sia un driver disponibile in grado di gestire la richiesta di connessione; quando lo trova può effettuare la connessione, con l'istruzione:

```
Connection con = DriverManager.getConnection(url, "username", "password");
```

L'oggetto **Connection** è l'oggetto restituito dal metodo **getConnection** della classe *DriverManager*.
Se non sono stati fissati username e password, l'istruzione viene usata nel seguente formato:

```
Connection con = DriverManager.getConnection(url, "", "");
```

La connessione rappresenta una sessione di lavoro con uno specifico database: all'interno di una connessione vengono eseguiti i comandi SQL e si ottengono i risultati con i dati richiesti.

AUTOVERIFICA
Domande da 1 a 3 pag. 408
Problemi da 1 a 2 pag. 409

3 Manipolazione dei dati

Per gli esempi presentati successivamente si farà riferimento a un database di nome *Utenti*, contenente la tabella *Anag* con i dati delle persone. Il tracciato del record anagrafico è il seguente:

Nome	Tipo	Dimensione
ID	Contatore	Intero lungo
Cognome	Testo	40
Nome	Testo	40
DataNascita	Data/ora	8
Luogo	Testo	40
Sesso	Testo	1
Via	Testo	50
CAP	Testo	5
Città	Testo	30
Provincia	Testo	2
Telefono	Testo	12
DataAggio	Data/ora	8
CodiceFiscale	Testo	16
Banca	Testo	30
Agenzia	Testo	30
NumeroConto	Testo	15
CAB	Testo	6
ABI	Testo	6

Il database, implementato in Access, è contenuto nel file *Utenti.mdb* ed è associato tramite ODBC al nome **db1**, specificato come nome della sorgente di dati (DSN).

Dopo aver stabilito la connessione, come visto nel precedente paragrafo, si possono inviare al database i comandi SQL.
Per far questo, prima si predispone l'istruzione SQL in una stringa, per esempio:

```
String query = "SELECT * FROM Anag";
```

Poi si predispone il comando JDBC (*statement*) con il metodo **createStatement** che crea un'istanza della classe **Statement**.
Per esempio, l'istruzione seguente usa la connessione (oggetto *con*) per costruire un comando SQL da inviare successivamente al database:

```
Statement stmt = con.createStatement();
```

I comandi SQL sono poi attivati con i due principali metodi applicati allo *statement*:

- metodo **executeUpdate** per i comandi SQL di inserimento (*Insert*), aggiornamento (*Update*) e cancellazione (*Delete*); si usa anche per i comandi SQL relativi alla creazione (*Create Table*), cancellazione (*Drop Table*), modifica della struttura (*Alter Table*) delle tabelle nel database;
- metodo **executeQuery** per le interrogazioni.

Il primo metodo restituisce il numero di righe che sono state sottoposte all'operazione di manipolazione della tabella, il secondo metodo restituisce un insieme di risultati (*ResultSet*) contenente le righe della tabella che soddisfano la condizione specificata nell'interrogazione SQL.

La **gestione delle eccezioni** per l'accesso al database tramite i comandi SQL viene gestita con una struttura come la seguente:

```
try
{
  ..............
  ..............
}
catch(SQLException ex)
{
  System.err.print("SQLException: ");
  System.err.println(ex.getMessage());
}
```

L'esempio seguente rappresenta il programma per l'inserimento di un nuovo record nella tabella del database, utilizzando il comando *Insert* di SQL.

PROGETTO 1

Inserire i dati di una nuova persona nella tabella delle anagrafiche.

PROGRAMMA JAVA (*Inserimento.java*)

```
/* Inserimento di un nuovo record */
import java.sql.*;

public class Inserimento
{
  public static void main(String args[])
  {
    // crea un URL specificando il DSN (data source name) di ODBC
    String url = "jdbc:odbc:db1";
    Connection con;
    String cmdSQL = "INSERT INTO Anag " +
      " (ID, Cognome, Nome, DataNascita, Luogo, Sesso, " +
      " Via, CAP, Città, Provincia, Telefono, DataAggio, " +
      " CodiceFiscale, Banca, Agenzia, NumeroConto, CAB, ABI) " +
      " VALUES (005, 'Rossi', 'Angelo', '12/02/65', 'Bologna', " +
      " 'M', 'Via Lunga 7', '20102', 'Milano', 'MI', '02/2828217', " +
      " '13/05/01', 'RSSNGL65B12R342Z', 'BIPOP', 'BO2', '23456', " +
      " '078977', '023411')";
    Statement stmt;

    // connessione al database
    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch(ClassNotFoundException e)
    {
      System.err.print("ClassNotFoundException: ");
      System.err.println(e.getMessage());
    }
```

```
    try
    {
      con = DriverManager.getConnection(url,"", "");
      stmt = con.createStatement();
      // esegue il comando SQL
      stmt.executeUpdate(cmdSQL);
      System.out.print("Record registrato");
      stmt.close();
      con.close();
    }
    catch(SQLException ex)
    {
      System.err.print("SQLException: ");
      System.err.println(ex.getMessage());
    }
  }
}
```

Per comodità di lettura il comando SQL è stato costruito con una stringa indicata con il nome *cmdSQL* e formata da diverse parti concatenate con l'operatore +. Si faccia attenzione all'uso corretto dei doppi apici per delimitare le parti di stringa e degli apici per delimitare i valori alfanumerici dei campi della tabella.

Al termine del programma, viene utilizzato il metodo **close** sia con l'oggetto *stmt*, per rilasciare le risorse usate dal comando SQL, che con l'oggetto *con* per interrompere la connessione al database.

Nel progetto seguente viene presentata un'applicazione Java per eliminare una persona dall'archivio anagrafico utilizzano il comando *Delete* di SQL.
In essa viene utilizzato il valore di ritorno del metodo *executeUpdate* per controllare se il record è stato effettivamente eliminato.

PROGETTO 2

Cancellare dall'archivio anagrafico i dati della persona avente il codice = 5.

PROGRAMMA JAVA (*Cancellazione.java*)

```
/* Cancellazione di un record */
import java.sql.*;

public class Cancellazione
{
  public static void main(String args[])
  {
    String url = "jdbc:odbc:db1";
    Connection con;
    String cmdSQL = "DELETE FROM Anag WHERE ID = 5";
    Statement stmt;
    int righe;
```

```
    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch(java.lang.ClassNotFoundException e)
    {
      System.err.print("ClassNotFoundException: ");
      System.err.println(e.getMessage());
    }
    try
    {
      con = DriverManager.getConnection(url,"", "");
      stmt = con.createStatement();
      righe = stmt.executeUpdate(cmdSQL);
      if (righe > 0)
      {
        System.out.print("Record eliminato");
      }
      else
      {
        System.out.print("Record da eliminare non trovato");
      }
      stmt.close();
      con.close();
    }
    catch(SQLException ex)
    {
      System.err.print("SQLException: ");
      System.err.println(ex.getMessage());
    }
  }
}
```

Il metodo *createStatement*, utilizzato negli esempi precedenti, in genere serve per attivare semplici comandi SQL che non contengono alcun parametro.
JDBC mette poi a disposizione anche il metodo **prepareStatement**, il quale crea un'istanza della classe **PreparedStatement**, che è una sottoclasse di **Statement** e che quindi ne eredita tutte le funzionalità.
Il metodo *prepareStatement* risulta particolarmente utile quando si devono eseguire istruzioni SQL con uno o più parametri, oppure istruzioni SQL semplici che però devono essere eseguite più volte.
I parametri sono indicati nel comando SQL con un punto di domanda.
Per esempio per indicare come parametri il codice della persona da aggiornare e il nuovo numero telefonico, si scrive il comando SQL in questo modo:

```
PreparedStatement stmt = con.prepareStatement(
    "UPDATE Anag SET Telefono = ? WHERE ID = ?");
```

Per assegnare poi i valori ai parametri si usano i metodi **set*XXX***, sostituendo a *XXX* uno dei tipi di dato previsti nel linguaggio Java, con l'indicazione del numero del parametro da sostituire (i parametri indicati con il punto di domanda nel comando SQL sono numerati da sinistra verso destra a partire da 1).

Per esempio:

```
stmt.setString(1, "02/8883917");
stmt.setInt(2, 365);
```

essendo *365* il valore del codice del record da aggiornare e *02/8883917* il nuovo numero telefonico.
Il telefono è il parametro 1 e il codice della persona è il parametro 2 nel comando SQL.
Il metodo *setXXX* converte i parametri nei tipi appropriati del linguaggio SQL.
A questo punto si può eseguire il comando SQL con il metodo **executeUpdate**, seguito dalle parentesi tonde vuote:

```
stmt.executeUpdate();
```

I dati utilizzati dai metodi *setXXX*, anziché essere indicati come costanti nel programma, possono essere anche acquisiti da tastiera, come mostrato nel progetto seguente.

PROGETTO 3

Aggiornare il numero di telefono nell'archivio anagrafico per una persona della quale viene fornito da tastiera il codice. Anche il nuovo numero di telefono deve essere inserito dall'utente.

PROGRAMMA JAVA (*Aggiornamento.java*)

```
/* Aggiorna un record con i dati inseriti da tastiera */
import java.io.*;
import java.sql.*;

class Aggiornamento
{
  public static void main(String argv[])
  {
    int codice;
    String telef;

    // impostazione dello standard input
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);

    try
    {
      // crea un URL specificando il DSN (data source name) di ODBC
      String url = "jdbc:odbc:db1";

      // connessione al database
      try
      {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      }
      catch(ClassNotFoundException e)
      {
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
      }
```

```
      Connection con = DriverManager.getConnection(url, "", "");
      // statement di preparazione all'aggiornamento con due parametri
      PreparedStatement stmt = con.prepareStatement(
          "UPDATE Anag SET Telefono = ? WHERE ID = ?");

      // inserimento da tastiera dei dati
      try
      {
        System.out.println("Codice da aggiornare ");
        String numeroLetto = tastiera.readLine();
        codice = Integer.valueOf(numeroLetto).intValue();
      }
      catch(Exception e)
      {
        System.out.println("\nNumero di codice non corretto!");
        return;
      }
      try
      {
        System.out.println("Nuovo numero telefonico ");
        telef = tastiera.readLine();
      }
      catch(Exception e)
      {
        System.out.println("\nInserimento errato");
        return;
      }

      // esegue il comando SQL
      stmt.setString(1, telef);
      stmt.setInt(2, codice);
      stmt.executeUpdate();
      System.out.println("Aggiornamento record: OK");
      stmt.close();
      con.close();
    }
    catch(SQLException ex)
    {
      System.err.print("SQLException: ");
      System.err.println(ex.getMessage());
    }
  }
}
```

Anche in questo programma si potrebbe usare il valore di ritorno del metodo *executeUpdate* per controllare che l'aggiornamento abbia avuto esito positivo, come visto nel Progetto 2 per la cancellazione.

AUTOVERIFICA

Domande da 4 a 5 pag. 408
Problemi da 3 a 7 pag. 409

4 Interrogazioni

Le interrogazioni al database vengono eseguite tramite il metodo **executeQuery**.
Per esempio, per effettuare una proiezione della *Tabella1* sulle colonne *a*, *b*, *c* occorre scrivere la seguente istruzione:

```
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Tabella1");
```

oppure, si può passare al metodo *executeQuery,* come parametro, una stringa *cmdSQL* che contiene il comando SQL.

```
String cmdSQL = "SELECT a, b,c FROM Tabella1";
...........
ResultSet rs = stmt.executeQuery(cmdSQL);
```

Il valore di ritorno del metodo *executeQuery* è un singolo **ResultSet**, cioè un insieme di risultati, contenente le righe della tabella che soddisfano la condizione specificata nell'interrogazione SQL. Il risultato di una *Select* su una tabella nei database relazionali infatti è a sua volta una tabella formata dalle righe selezionate e dalle colonne scelte.
Per scorrere questa tabella si deve far riferimento a un **cursore** che indica la riga della tabella. Dopo l'esecuzione del comando *executeQuery*, il cursore si trova automaticamente posizionato sulla prima riga del *ResultSet*: ci si può muovere sulle righe successive con il metodo **next()**.
Si supponga che i campi *a*, *b*, *c* siano rispettivamente di tipo *int*, *float*, *String*; la visualizzazione dei risultati contenuti nel *ResultSet* si ottiene con il seguente ciclo di lettura sequenziale che utilizza i metodi **get*XXX***, dove *XXX* va sostituito con uno dei tipi di dato di Java.

```
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Tabella1");
while (rs.next())
{
  int i = rs.getInt("a");
  float f = rs.getFloat("b");
  String s = rs.getString("c");
  System.out.println(i);
  System.out.println(f);
  System.out.println(s);
}
```

In alternativa, anziché il nome del campo (etichetta della colonna), si può indicare, come argomento dei metodi *getXXX* il numero della colonna contenente il dato, come nell'esempio seguente.

```
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Tabella1");
while (rs.next())
{
  int i = rs.getInt(1);
  float f = rs.getFloat(2);
  String s = rs.getString(3);
  System.out.println(i);
  System.out.println(f);
  System.out.println(s);
}
```

Il progetto seguente rappresenta un'applicazione Java che esegue un'interrogazione al database, utilizzando un comando Select di SQL.

PROGETTO 4

Visualizzare il codice, il cognome, il nome e il codice fiscale delle persone che sono residenti nella provincia di Milano.

PROGRAMMA JAVA (*Selezione.java*)

```
/* Visualizza i dati ottenuti da un'interrogazione */
import java.sql.*;

public class Selezione
{
  public static void main(String args[])
  {
    String url = "jdbc:odbc:db1";
    Connection con;
    String query = "SELECT ID, Cognome, Nome, CodiceFiscale " +
                   "FROM Anag " +
                   "WHERE Provincia = 'MI'";
    Statement stmt;

    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch(ClassNotFoundException e)
    {
      System.err.print("ClassNotFoundException: ");
      System.err.println(e.getMessage());
    }
    try
    {
      con = DriverManager.getConnection(url,"", "");
      stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(query);
      while (rs.next())
      {
        int cod = rs.getInt(1);
        String cogn = rs.getString(2);
        String nom = rs.getString(3);
        String cf = rs.getString(4);
        // visualizza i campi
        System.out.println("Codice = " + cod);
        System.out.println("Cognome = " + cogn);
        System.out.println("Nome = " + nom);
        System.out.println("Codice Fiscale = "+ cf);
        System.out.print("\n");
      }
      stmt.close();
      con.close();
    }
    catch(SQLException ex)
    {
      System.err.print("SQLException: ");
      System.err.println(ex.getMessage());
    }
  }
}
```

Il progetto seguente esegue la stessa interrogazione, selezionando però le persone sulla base della provincia fornita dall'utente del programma tramite tastiera. In questo caso occorre usare il comando **prepareStatement** per indicare un comando *Select* di SQL con il parametro.

PROGETTO 5

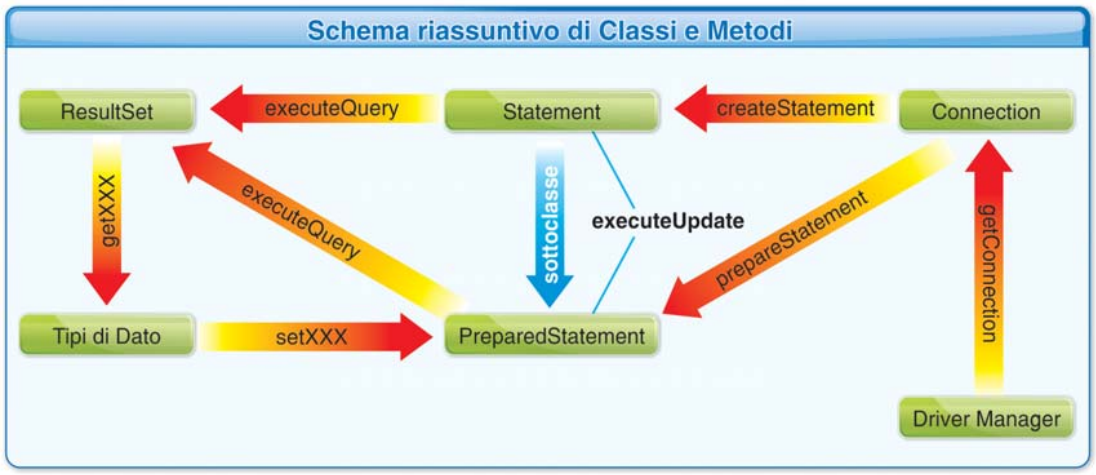
Visualizzare il codice, il cognome, il nome e il codice fiscale delle persone che sono residenti in una provincia fornita da tastiera.

PROGRAMMA JAVA (*SelezParam.java*)

```
/* Interrogazione con parametro */
import java.io.*;
import java.sql.*;

public class SelezParam
{
  public static void main(String args[])
  {
    String prov;
    // impostazione dello standard input
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);
    String url = "jdbc:odbc:db1";
    Connection con;

    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch(ClassNotFoundException e)
    {
      System.err.print("ClassNotFoundException: ");
      System.err.println(e.getMessage());
    }

    try
    {
      con = DriverManager.getConnection(url,"", "");
      // statement di preparazione all'aggiornamento con un parametro
      PreparedStatement stmt = con.prepareStatement(
          "SELECT ID, Cognome, Nome, CodiceFiscale " +
          "FROM Anag " +
          "WHERE Provincia = ? ");

      // inserimento da tastiera della provincia
      try
      {
        System.out.println("Provincia richiesta ");
        prov = tastiera.readLine();
      }
```

```
      catch(Exception e)
      {
        System.out.println("\nInserimento errato");
        return;
      }

      // esegue il comando SQL
      stmt.setString(1, prov);
      ResultSet rs = stmt.executeQuery();
      while (rs.next())
      {
        int cod = rs.getInt(1);
        String cogn = rs.getString(2);
        String nom = rs.getString(3);
        String cf = rs.getString(4);
        // visualizza i campi
        System.out.println("Codice = " + cod);
        System.out.println("Cognome = " + cogn);
        System.out.println("Nome = " + nom);
        System.out.println("Codice Fiscale = " + cf);
        System.out.print("\n");
      }

      stmt.close();
      con.close();
    }
    catch(SQLException ex)
    {
      System.err.print("SQLException: ");
      System.err.println(ex.getMessage());
    }
  }
}
```

Il seguente schema riassume le classi e i metodi utilizzati per la connessione al database e per le operazioni di manipolazione e interrogazione.



METADATI

JDBC mette a disposizione del programmatore un'importante classe, chiamata **DatabaseMetaData**, che fornisce i metodi per conoscere tutte le informazioni sulla struttura del database (cioè i **metadati**) come un insieme unico, restituendo un oggetto **ResultSetMetaData** per ogni *ResultSet* ottenuto con una query.

Il comando **ResultSet.getMetaData** restituisce un oggetto *ResultSetMetaData* che fornisce numero, tipo e proprietà delle colonne ottenute con l'oggetto *ResultSet*.
Il seguente frammento di programma Java visualizza, per ogni riga dell'insieme *ResultSet* ottenuto con una query, il numero della colonna e il suo contenuto.

```
ResultSet rs = stmt.executeQuery(query);
ResultSetMetaData rsmd = rs.getMetaData();
int numberOfColumns = rsmd.getColumnCount();
int rowCount = 1;
while (rs.next())
{
  System.out.println("Riga " + rowCount + ": ");
  for (int i = 1; i <= numberOfColumns; i++)
  {
    System.out.print(" Colonna " + i + ": ");
    System.out.println(rs.getString(i));
  }
  System.out.println("");
  rowCount++;
}
```

I metodi forniti dalla classe *DatabaseMetaData*, applicati all'oggetto *ResultSetMetaData*, possono essere opportunamente utilizzati per visualizzare la struttura di una tabella del database, ottenendo l'elenco dei campi, con numero, nome e tipo di ciascuna colonna della tabella.

PROGETTO 6

Visualizzare il nome e il tipo per ciascuna colonna di una tabella del database.

PROGRAMMA JAVA (*Struttura.java*)

```
/* Visualizza la struttura di una tabella */
import java.sql.*;

public class Struttura
{
  public static void main(String args[])
  {
    String url = "jdbc:odbc:db1";
    Connection con;
    String query = "SELECT * FROM Anag ";
    Statement stmt;
```

```
    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch(ClassNotFoundException e)
    {
      System.err.print("ClassNotFoundException: ");
      System.err.println(e.getMessage());
    }

    try
    {
      con = DriverManager.getConnection(url,"", "");
      stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(query);
      ResultSetMetaData rsmd = rs.getMetaData();
      int numberOfColumns = rsmd.getColumnCount();
      for (int i = 1; i <= numberOfColumns; i++)
      {
        String n = rsmd.getColumnName(i) ;
        String t = rsmd.getColumnTypeName(i) ;
        System.out.print(" Colonna " + i + ": \t" + n);
        System.out.println("      Tipo: " + t);
      }
      stmt.close();
      con.close();
    }
    catch(SQLException ex)
    {
      System.err.print("SQLException: ");
      System.err.println(ex.getMessage());
    }
  }
}
```

AUTOVERIFICA

Domande da 6 a 7 pag. 408
Problemi da 8 a 12 pag. 409
Problemi di riepilogo da 13 a 15 pag. 409

MATERIALI ON LINE

2. Dati storici sulle vendite per reparto
3. Forum per internet

AUTOVERIFICA

DOMANDE

Connessione ai database

1 Completa le frasi seguenti utilizzando una tra le sigle elencate alla fine della domanda.

a) La sigla ..... indica i driver per database che evitano di dover scrivere un programma diverso per ciascun tipo di database.

b) La sigla ..... indica il nome che identifica il database quando viene utilizzato da un'applicazione Web.

c) La sigla ..... indica una libreria di classi per interfacciare l'accesso ai database che usano lo standard SQL.

DSN, DBJC, DBOC, JDBC, ODBC, DNS

2 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?

a) Il software JDBC è classificato come una API V F

b) I programmi che usano JDBC devono necessariamente specificare l'uso del package *java.awt* V F

c) Per essere utilizzato in un'applicazione il database deve essere associato a un'origine dati ODBC V F

d) L'URL di JDBC indica il nome del driver per il database V F

e) La connessione rappresenta una sessione di lavoro con uno specifico database V F

3 Quale delle seguenti frasi dichiara correttamente un URL di JDBC?

a) String url ="jdbcodbc:db1";
b) String url ="odbc:jdbc:db1";
c) String url ="jdbc:odbc:db1";
d) String url ="db1:odbc:jdbc";

Operazioni di manipolazione

4 Quale dei seguenti metodi esegue un comando di manipolazione?

a) updateExecute
b) executeSQL
c) executeQuery
d) executeUpdate

5 Quale comando risulta particolarmente utile quando si devono eseguire istruzioni SQL con uno o più parametri?

a) prepareStatement
b) statementPrepare
c) createStatement
d) parameterStatement

Interrogazioni

6 Quale delle seguenti frasi rappresenta il risultato ottenuto con un comando *executeQuery*?

a) Il valore di ritorno è un ResultSet, contenente le righe della tabella ottenute con l'interrogazione SQL.
b) Il valore di ritorno è il numero di righe del ResultSet ottenuto con l'interrogazione SQL.
c) Il valore di ritorno è il numero di colonne del ResultSet ottenuto con l'interrogazione SQL.
d) Il valore di ritorno è il nome delle colonne del ResultSet ottenuto con l'interrogazione SQL.

7 Quale delle seguenti frasi corrisponde alla descrizione della classe *DatabaseMetaData*?

a) La classe che fornisce i metodi per effettuare le operazioni di manipolazione
b) La classe che fornisce i metodi per effettuare le operazioni di interrogazione
c) La classe che fornisce i metodi per conoscere tutte le informazioni sulla struttura del database
d) La classe che fornisce i metodi per conoscere i valori contenuti nelle colonne di una tabella

PROBLEMI

Connessione ai database

1 Creare un database di Access, formato da due tabelle per registrare i dati sulle *automobili* e sui *motocicli*.

2 Usando il database del problema precedente, stabilire una connessione ODBC nel Pannello di Controllo di Windows, scegliendo il driver opportuno e indicando un nome DSN utente.

Operazioni di manipolazione

3 Creare una nuova tabella in un database di Access con i campi: Codice, Descrizione e Importo. Scrivere l'applicazione Java per inserire alcuni dati di prova da programma.

4 Scrivere l'applicazione Java per inserire una nuova riga nella tabella, acquisendo i dati da tastiera.

5 Modificare l'importo, diminuendolo del 10%, di una riga della tabella della quale viene fornito da tastiera il codice.

6 Cancellare dalla tabella tutte le righe per le quali l'importo risulta inferiore a 100.

7 Cancellare dalla tabella tutte le righe per le quali l'importo risulta inferiore a una cifra prefissata e fornita da tastiera.

Interrogazioni

8 Costruire una query che estrae dalla tabella i record aventi nel campo Importo valori superiori a 150.

9 Costruire una query parametrica che estrae dalla tabella i record aventi nel campo Importo valori superiori a una cifra prefissata e fornita da tastiera.

10 Costruire un'applicazione Java che restituisce la somma totale degli importi di tutte le righe della tabella.

11 Costruire una query parametrica che estrae dalla tabella i record aventi nel campo Codice valori inferiori a un codice prefissato e fornito da tastiera.

12 Scrivere l'applicazione Java per visualizzare la struttura della tabella, con il nome e il tipo dei campi.

PROBLEMI DI RIEPILOGO

Per ciascuno dei seguenti esercizi, dopo aver creato il database contenente la tabella indicata e aver definito una connessione ODBC, scrivere l'applicazione Java per eseguire l'interrogazione in SQL.

13 Data la tabella con i dati dei *contribuenti*, contenente quattro colonne con CodiceFiscale, Cognome, Nome, Reddito, produrre l'elenco dei contribuenti aventi il codice fiscale che inizia con 'RSS'.

14 Data la tabella con i *motivi musicali*, contenente per ogni riga il Nome dell'artista, il Titolo e l'Anno di produzione, fornire l'elenco dei motivi musicali in ordine decrescente di anno, solo con il nome dell'artista e l'anno di produzione.

15 Data la tabella dei *voli aerei* con Numero del volo, Descrizione e Compagnia, dopo aver fornito da tastiera i nomi di due compagnie, si vuole calcolare quanti sono i voli appartenenti a ciascuna di esse.

Focus notes

DATABASE

A database is any collection of related information about a subject, organised in a way that enables rapid access to selected data. Computer databases are organised as collections of data files, a data file is a set of data records organised as a group of data fields. The smallest unit of information is the data field. A data field that uniquely identifies a specific record is called a primary key.
Usually, some level of quality in terms of accuracy, availability, usability is required which implies the use of a *database management system* (DBMS). Typically, a DBMS is a software system that meets a number of requirements. The use of databases is now spread to such a wide degree that virtually every technology and product relies on databases. Simple databases consist of a single data file and one or more related index files. The data file contains the data with the information of interest while the index files, which are usually hidden from the user, provide high-speed access to specific records in the data file using a system of pointers: like when using the index of a book to reach the pages with the requested information.

RELATIONAL DATABASE

A relational database is based on the relational data model first proposed by E. F. Codd in 1970. According to the relational model, a database is a set of tables called relations. Each table is made up of named attributes or columns of data. A row of data in the table contains as many attributes as the number of columns in the table. A great strength of the relational model is the simple logical structure, but behind this simple structure lies a strong theoretical foundation. In the terminology of the relational model a table has a degree and a cardinality. The degree of a relation is the number of attributes it contains, the cardinality of a table is the number of rows it contains and a relational database is a collection of tables with different names.

SQL

SQL stands for *Structured Query Language*. SQL is a language used to work with databases. You can send an SQL Select command to the database to retrieve data from it, or you can insert new records in a table with the Insert command, modify one or more attribute values with the Update command and delete one or more records from a table with the Delete command. The SQL commands that modify the data in the database are identified as the DML (*Data Manipulation Language*) subset of SQL. There are also commands for creating, declaring and modifying the structure of a table or other objects of the database. These commands represent the DDL (*Data Definition Language*) part of the SQL language.

JDBC

The JDBC is a Java API that can access any kind of tabular data, especially data stored in a relational database. JDBC helps to create tables, insert values into them, query the tables, retrieve the results of the queries, and update the tables.
JDBC helps the programmer to write Java applications that manage these programming activities:
- connect to a database
- create tables and insert values into them
- update the tables or delete rows from a table
- query the tables and process the results received from the database in answer to a query.

Focus notes

Glossary

API
A software to be used as an interface by software components to communicate with each other.

driver
A software program designed to allow a device to communicate with a computer.
A JDBC driver is a software component enabling a Java application to interact with a database.

field
The element of a database having settings that determine the type of data they can store and how the data is displayed. One important setting for fields is the data type, including number, text, currency (money), and date/time. The data type limits and describes the kind of information in the field. The data type also determines how much memory the data uses.

manipulation
Command or action to insert, update or delete database information.

metadata JDBC
A collection of information about the database's structure (table, columns and column properties).

query
A request for information from a database that meets a set of criteria.

query language
A language for querying, or retrieving data from, a database or other file system. The de-facto standard query language to query a database is SQL (*Structured Query Language*).

Acronyms

API	Application Programming Interface
DB	DataBase
DBMS	DataBase Management System
DSN	Data Source Name
JDBC	Java DataBase Connectivity
JDK	Java Development Kit
ODBC	Open Database Connectivity
RDBMS	Relational DataBase Management System
SQL	Structured Query Language
URL	Uniform Resource Locator

SCHEDA DI AUTOVALUTAZIONE

CONOSCENZE

- ☐ Driver per la connessione ai database
- ☐ Caratteristiche di JDBC
- ☐ Applicazioni Java per operazioni di manipolazione
- ☐ Interrogazioni in SQL
- ☐ Metadati del database

ABILITÀ

- ☐ Fissare le impostazioni di ODBC
- ☐ Codificare in Java le operazioni di manipolazione usando JDBC
- ☐ Utilizzare i parametri nelle operazioni di manipolazione
- ☐ Gestire le eccezioni durante le operazioni di connessione e di accesso al database
- ☐ Rappresentare le interrogazioni SQL
- ☐ Utilizzare i parametri nelle query
- ☐ Visualizzare le informazioni sulle tabelle e sui campi del database

SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 539

8

parte quarta

Applicazioni Web

Servlet e pagine JSP

L'architettura Client/Server

Le servlet

Compilazione ed esecuzione della servlet

Interazione con il client

Le servlet per la connessione ai database

Le pagine JSP

Attivazione di una pagina JSP

Passaggio di parametri alla pagina JSP

Accesso ai database con JSP

OBIETTIVI DI APPRENDIMENTO

In questo capitolo conoscerai gli strumenti per gestire le interazioni nel Web.

Sarai in grado di costruire pagine Web dinamiche e imparerai a usare gli oggetti e i comandi per effettuare manipolazioni e interrogazioni ai database nel Web.

1 L'architettura Client/Server

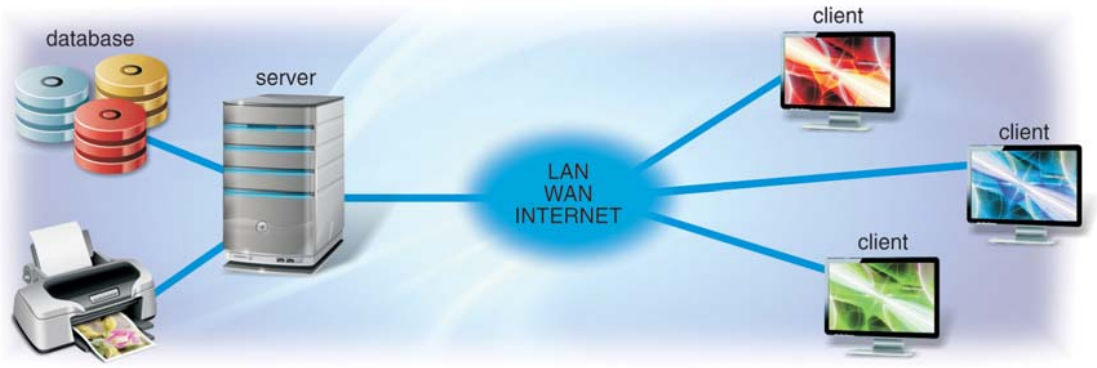
L'**architettura Client/Server** (C/S) permette a diversi calcolatori collegati in rete di utilizzare e condividere le risorse. In questa situazione, i calcolatori, così come le risorse, sono distribuiti su un'area che non ha estensioni predefinite e sono collegati tra loro per formare le reti. Si parla di *rete locale* (LAN) se il collegamento tra i calcolatori è limitato a un unico edificio, altrimenti se l'estensione riguarda una zona più ampia si parla di *rete geografica* (WAN).
Internet è un'insieme di reti, anche di tipo diverso, interconnesse tra loro.
I calcolatori collegati in rete sono solitamente diversi tra loro, ognuno ha la sua dotazione hardware e software che può decidere di condividere con gli altri.

Alla base dell'architettura C/S c'è il concetto di **condivisione delle risorse**. Con il termine *risorse* si fa riferimento sia alle apparecchiature hardware che ai programmi software o ai dati. Esse sono solitamente costose e devono essere sfruttate da vari utenti. Esempi di risorse sono le CPU veloci, le unità di backup, le stampanti, i database e i programmi applicativi.

In un'architettura C/S si possono distinguere due entità:
- il **server**: il computer che mette a disposizione la risorsa e assume il ruolo di servente;
- il **client**: il computer che utilizza la risorsa condivisa e assume il ruolo di cliente.

Le entità server e client sono collegate tramite LAN o WAN e comunicano attraverso la rete usando un protocollo di comunicazione. Il **protocollo** è l'insieme di regole conosciute da entrambe le entità che stabilisce la forma e il significato da attribuire ai messaggi scambiati.

Si può fare riferimento all'approccio C/S sia da un punto di vista hardware che da un punto di vista software.
A livello hardware, una rete basata su un'architettura C/S è caratterizzata dalla presenza di computer potenti, chiamati *server*, dotati di veloci CPU, di unità per memorizzare grandi quantità di dati, di periferiche costose da condividere tra molti utenti. Questi server sono collegati in rete con altri calcolatori meno potenti, chiamati *client*, identificabili con i personal computer. I client sfruttano le capacità dei server per accedere ai database, per usare le stampanti e le altre risorse condivise o per demandare i calcoli alle CPU veloci del server.



Anche a livello software si può parlare di architettura C/S. In questo caso le entità vengono identificate dai programmi che sono in esecuzione su macchine diverse e possono comunicare tra loro attraverso la rete.

I **programmi server** (o semplicemente i *server*) sono le applicazioni che offrono un servizio e sono attivi sui computer potenti che rivestono il ruolo di server.

Il compito delle applicazioni server è quello di restare in attesa delle richieste inviate da altri programmi e, non appena ne arriva una, di generare una risposta. La struttura dei programmi server è fondamentalmente la seguente:

```
while (true)
{
  // aspetta la richiesta

  // esegue la richiesta
}
```

I **programmi client** (o semplicemente i *client)* sono quelli che formulano le richieste al server volte a ottenere un certo servizio.

Le applicazioni client sono solitamente in esecuzione sui personal computer dei vari utenti collegati alla rete e si preoccupano della gestione dell'interfaccia grafica per richiamare i servizi del server.
La comunicazione tra il server e il client avviene attraverso la spedizione di messaggi di richiesta e messaggi di risposta. Sia le richieste che le risposte devono essere codificate usando un particolare protocollo conosciuto sia dal server che dal client.

Un'applicazione che gestisce l'accesso a un database condiviso può essere realizzata usando un'architettura Client/Server. Si può immaginare una situazione in cui vari utenti, collegati a diversi personal computer, interagiscono con un database condiviso memorizzato su un server. Un'applicazione di questo tipo sarà composta da due moduli: un client e un server. Il modulo client si occupa della gestione dell'interfaccia grafica e dell'interpretazione dei comandi dell'utente. Il modulo server ha il compito di eseguire le interrogazioni ed effettuare gli aggiornamenti al database.

La rete Internet e le applicazioni che vengono utilizzate da chi si collega alla rete, sono basate su un'architettura C/S. Il *World Wide Web* o **WWW** è uno tra i servizi più conosciuti offerti da Internet. Il servizio WWW consente a varie persone, sparse in diverse località del mondo, di condividere un insieme di documenti chiamati **pagine Web**. In questo contesto, il ruolo di client è assunto dal personal computer degli utenti che visitano le pagine Web, invece il ruolo di server spetta ai calcolatori che memorizzano i documenti e decidono di condividerli. Anche a livello software si può fare un analogo parallelo: da un lato c'è il **browser** (client) che viene utilizzato per richiedere le pagine Web, dall'altro c'è un programma, chiamato **server Web**, che ha il compito di rispondere inviando il documento richiesto.

Il server Web è un'applicazione che rimane sempre in esecuzione in attesa delle richieste inviate dai browser. La richiesta è rappresentata dal nome del file corrispondente a una pagina Web. Ricevuta una richiesta, il server Web cerca sul suo disco il file richiesto: se lo trova risponde inviando il file, altrimenti segnala un messaggio di errore.
I server Web più diffusi sono *Apache* per il sistema operativo Linux e *Internet Information Services (IIS)* per il sistema operativo Windows.
Il server Web può essere installato anche sul proprio computer personale, eventualmente collegato a una rete locale.

Per esempio IIS per Windows, si installa da *Pannello di controllo*, *Programmi e funzionalità*, *Attivazione o disattivazione delle funzionalità di Windows*. Il processo di installazione crea una directory *Inetpub* e, al suo interno, la sottocartella *wwwroot* per le pagine Web.
Il server Web può essere poi avviato o arrestato con la voce *Gestione Internet Information Services (IIS)* presente in *Strumenti di amministrazione* del *Pannello di controllo*.
Per controllare che il server Web sia attivo, basta aprire il browser e scrivere nella casella dell'indirizzo

```
http://localhost
```

essendo **localhost** il nome che identifica il server Web della macchina locale. Il server *localhost* corrisponde all'indirizzo IP **127.0.0.1**, per cui si può scrivere in modo equivalente

```
http://127.0.0.1
```

In output compare una pagina di test, che conferma l'attivazione del server Web.
Il nome *localhost* viene usato anche per accedere al server Web dalla stessa macchina sulla quale è installato il server Web per una rete.

La comunicazione tra il server Web e il browser Web si basa sull'utilizzo di un protocollo chiamato **HTTP** (*HyperText Transfer Protocol*). Questo protocollo stabilisce come devono essere formulate le richieste da parte del client e le risposte da parte del server.
Il servizio HTTP è normalmente associato alla **porta 80**. Il termine *porta* viene usato, in questo contesto, per indicare una connessione logica al computer dall'esterno tramite un programma software.
Esistono vari tipi di richieste fatte dal browser e vari tipi di risposte che il server restituisce: ognuna è definita esattamente dalle specifiche del protocollo HTTP.
Le principali richieste che possono essere fatte da un browser sono:
- richiesta di una pagina Web o di un altro file, indicata dal comando **GET**
- richiesta di una pagina Web con la possibilità di spedire i dati al server, indicata dal comando **POST**
- richiesta della sola intestazione della pagina Web, indicata dal comando **HEAD.**

Le risposte che un server Web genera sono formate da un'intestazione contenente le informazioni sul risultato della richiesta, seguite dall'eventuale pagina Web. Esistono vari codici per indicare il risultato di una richiesta, per esempio il codice 200 segnala che tutto ha funzionato correttamente, mentre i codici il cui numero è del tipo 4xx indicano la presenza di errori. Il codice 400 indica una richiesta non corretta mentre 404 indica una pagina inesistente: quest'ultimo è l'errore che corrisponde a un errato indirizzo di una pagina Web.

ESTENSIONI DEL SERVER WEB

Il contenuto dei documenti WWW ha subìto un'evoluzione nel corso degli anni: si è passati da un contenuto di tipo statico a un contenuto di tipo dinamico.
Con il termine **contenuto statico** si fa riferimento a documenti formati da testo con al più immagini non animate.

Bisogna ricordare che, agli inizi di Internet, tutte le informazioni erano presentate in modo testuale e i primi browser consentivano la sola visualizzazione di caratteri. Successivamente furono creati i browser grafici: questi permettevano l'inserimento di immagini all'interno dei documenti, oltre alla possibilità di formattare il testo scegliendo il tipo di carattere, la dimensione, il colore e l'allineamento.

Chi visualizzava le pagine continuava però a vedere documenti fissi e statici e questo poteva essere ritenuto sufficiente considerando che le informazioni presentate si limitavano ad articoli scientifici e a materiale consultato solo da poche persone.
La diffusione di Internet e il suo utilizzo da parte di un crescente numero di navigatori, ha portato a un cambiamento nel modo di presentare i contenuti.
Sono stati introdotti in continuazione nuovi elementi per rendere la navigazione più piacevole e per stimolare l'interattività che è tipica di questo mezzo di comunicazione.
Nei documenti WWW hanno trovato spazio le immagini animate, le applet scritte in Java e molti altri elementi multimediali. Anche la possibilità di interagire con i database memorizzati sul server ha contribuito a rendere le pagine più dinamiche.

I **contenuti dinamici** possono essere classificati in due categorie a seconda del punto in cui intervengono la dinamicità e l'elaborazione dei dati:
- dal lato client,
- dal lato server.

I *contenuti dinamici dal lato client* si manifestano con gli elementi multimediali collegati alle pagine Web, tra cui le immagine animate, i suoni e i filmati video.
Inoltre esistono anche i **linguaggi di scripting**, così chiamati perché consentono di creare piccoli programmi, che non devono essere compilati, ma che vengono inseriti direttamente in formato testuale nelle pagine Web. Questi programmi si chiamano **script** e l'interpretazione dei loro comandi viene fatta direttamente dal browser Web. Il linguaggio di scripting più conosciuto e utilizzato è **Javascript**.
Tramite questo linguaggio si può agire direttamente su tutti gli elementi presenti in una pagina Web. Per esempio è possibile controllare il contenuto dei campi di un formulario oppure si possono modificare le immagini rendendole sensibili al passaggio del mouse.

I *contenuti dinamici dal lato server* si sono sviluppati come estensioni del server Web. Il compito primario del server Web è quello di rispondere alle richieste dei browser inviando la corrispondente pagina Web così come è memorizzata. Le estensioni del server Web sono state introdotte per permettere un'elaborazione della pagina prima che venga inviata al client. Un server Web a cui sono state aggiunte queste estensioni non restituisce più una pagina statica, ma risponde alle richieste con pagine che possono cambiare ogni volta.
Alcune estensioni consistono nel posizionare all'interno delle pagine HTML i tag aggiuntivi e il codice che il server Web deve interpretare, come accade per le **pagine ASP** (*Active Server Page*) sviluppate per il server Web della Microsoft.
Utilizzando queste pagine è possibile accedere a database relazionali ed eseguire modifiche o interrogazioni agli stessi. Anche il linguaggio **PHP**, molto usato in ambiente *Linux*, consente di creare pagine dinamiche per il Web, con le quali gestire l'interazione con il browser dell'utente e l'accesso ai database in rete.
Simili alle pagine ASP e PHP, sono le **pagine JSP** (*Java Server Page*) il cui codice viene scritto utilizzando i costrutti del linguaggio Java e dei quali si parlerà nei paragrafi successivi di questo capitolo.

AUTOVERIFICA

Domande da 1 a 5 pag. 452
Problemi da 1 a 2 pag. 454

MATERIALI ON LINE

1. Installare e configurare Apache Web Server

2 Le servlet

La **servlet** è un particolare programma scritto in Java il cui scopo è quello di estendere le funzionalità del server Web.

Ogni volta che il server Web riceve una richiesta che fa riferimento a una servlet, invoca l'esecuzione di un metodo particolare della servlet stessa con lo scopo di soddisfare la richiesta. Le servlet consentono di trattare i dati spediti tramite i form e sono utili per gestire i database condivisi che si trovano sul server.
La programmazione di un servizio dal lato server, usando le servlet, consente di ereditare tutti i vantaggi del linguaggio di programmazione Java, tra cui una metodologia orientata agli oggetti, la portabilità del codice e le numerose librerie disponibili.
Le servlet rappresentano il terzo tipo di programmi che può essere realizzato con Java. I primi due tipi, trattati nei precedenti capitoli, sono le applicazioni e le applet. Le applicazioni sono programmi a sé stanti che possono essere mandati in esecuzione invocando l'interprete Java. Le applet sono anch'esse programmi, ma il loro ambiente di esecuzione è il browser Web.
È possibile fare un **confronto tra applet e servlet**. Entrambe hanno lo scopo di introdurre elementi dinamici all'interno delle pagine Web. Le applet lo fanno dal lato client, estendendo le funzionalità del browser Web. Le servlet sono un'estensione del server Web e introducono dinamicità dal lato del server.
Sia le applet che le servlet non sono programmi autonomi, ma vengono comandati da un altro programma che ne gestisce l'inizializzazione, l'esecuzione e la terminazione.
La differenza riguarda l'interfaccia grafica: le servlet non hanno bisogno di interfaccia grafica, mentre per le applet essa è molto importante.

Tutto ciò che riguarda le servlet è racchiuso nelle due API **javax.servlet** e **javax.servlet.http**. Queste non fanno parte delle librerie standard di Java ma sono rese disponibili insieme ad altri prodotti per l'estensione dei server Web.

Per poter creare una servlet si deve specificare l'uso dei due package relativi alle servlet usando i comandi:

```
import javax.servlet.*;
import javax.servlet.http.*;
```

Una servlet è rappresentata in pratica da una classe che viene definita estendendo la classe *HttpServlet*:

```
public class NomeServlet extends HttpServlet
```

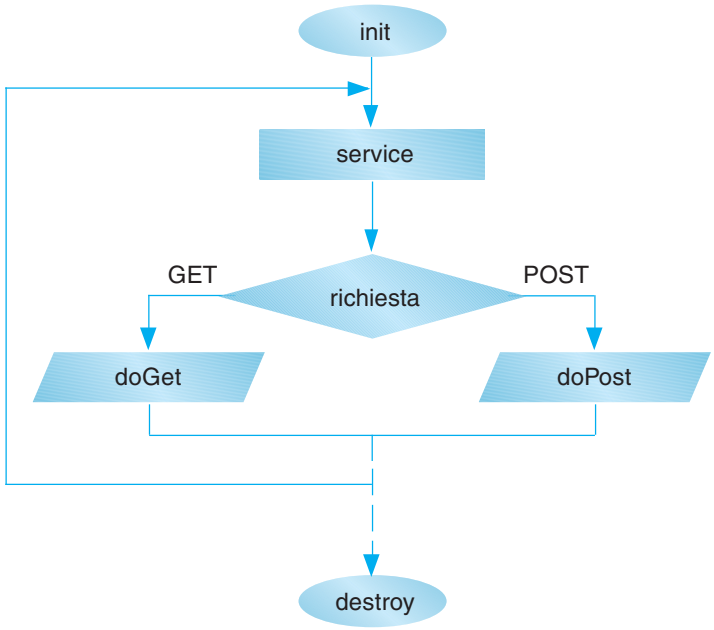
La classe *HttpServlet* contiene vari metodi che vengono ereditati dalla servlet e possono essere da questa ridefiniti. I metodi principali sono **init**, **doGet**, **doPost** e **destroy**.

Prima di vedere il significato di questi metodi è utile capire come viene trattata una servlet dal server Web. L'esecuzione delle servlet avviene su richiesta di un client, tipicamente un browser Web, che richiede al server l'attivazione di un particolare servizio. Alla prima esecuzione di una servlet, il server Web si occupa di caricarla e di inizializzarla. Le successive richieste della stessa servlet non ripeteranno più queste fasi, ma si limiteranno ad attivare la servlet già inizializzata per soddisfare il servizio.

Il server Web deve invocare particolari metodi che vengono definiti nella classe della servlet. Nella fase di inizializzazione viene invocato il metodo **init**: tutte le operazioni che vanno eseguite una sola volta, quali la connessione con un database, devono essere inserite in questo metodo. Successivamente, a ogni richiesta, il server Web invoca il metodo **service** che ha il compito di capire il tipo di richiesta e indirizzarla ai metodi opportuni.

I due principali tipi di richieste sono:
- richiesta di tipo **GET**: usata per richiedere una pagina Web, può spedire in allegato anche dati, ma non di grandi dimensioni;
- richiesta di tipo **POST**, usata per spedire grandi quantità di informazioni al server.

Se il metodo *service* riceve una richiesta di tipo GET manda in esecuzione il metodo **doGet**, se riceve una richiesta di tipo POST invoca il metodo **doPost**. Questi due metodi avranno il compito di generare una risposta, solitamente sotto forma di pagina Web, e spedirla al client.



Il server Web può infine decidere di eliminare una servlet invocando il metodo **destroy**.
Le servlet funzionano secondo un'architettura di tipo Client/Server in cui il client invia una richiesta e il server restituisce una risposta. La comunicazione si basa sul protocollo HTTP e viene mediata dal server Web.

Le richieste e le risposte sono gestite rispettivamente dalle classi **HttpServletRequest** e **HttpServletResponse**.
Gli oggetti generati dalla classe *HttpServletRequest* contengono tutte le informazioni relative alla richiesta ricevuta dal client, in particolare i valori delle variabili di ambiente, i parametri e i dati spediti dal client e il tipo di richiesta. Gli oggetti generati dalla classe *HttpServletResponse* consentono di controllare la risposta che verrà restituita al client.

Il codice che rappresenta il cuore di una servlet deve essere inserito nel metodo *doGet* oppure nel metodo *doPost* a seconda del tipo di richiesta che si vuole gestire.
Questi due metodi possiedono come parametri un oggetto che fa riferimento alla richiesta, di classe *HttpServletRequest*, e un oggetto relativo alla risposta, di classe *HttpServletResponse*.

La struttura per entrambi i metodi è la seguente:

```
public void doGet(HttpServletRequest richiesta,
                  HttpServletResponse risposta)
throws IOException, ServletException
{
  . . .
}
```

Il costrutto **throws** serve per indicare quali sono le eccezioni che il metodo potrebbe generare ma che non gestisce al suo interno.

Per impostare la risposta da restituire al client si deve specificare il tipo con l'istruzione:

```
risposta.setContentType("text/html");
```

Tramite un oggetto di classe **PrintWriter**, contenuta nel package *java.io*, si può aprire un canale per inserire tutti i dati testuali che vengono restituiti al client in risposta alla sua richiesta. Questo canale viene creato richiamando il seguente metodo:

```
PrintWriter out = risposta.getWriter();
```

Avendo a disposizione l'oggetto **out** si può utilizzare il metodo *println* per aggiungere tutte le stringhe che fanno parte della risposta.

Le stringhe inviate possono essere anche tag HTML per la formattazione della pagina inviata al browser.
Per esempio:

```
out.println("<HTML><BODY></BODY></HTML>");
```

Questa istruzione invia come risposta una pagina Web vuota. Per chiudere il canale attraverso il quale vengono spediti i dati si deve usare l'istruzione:

```
out.close();
```

Il seguente progetto mostra una servlet che gestisce le richieste di tipo GET, sovrascrive il metodo *doGet*, ma non modifica il metodo *init*.

PROGETTO 1

Generare, tramite una servlet, una pagina Web contenente un titolo e un'immagine fotografica.

La servlet deve essere dichiarata come sottoclasse di *HttpServlet* nel seguente modo:

```
public class Fotografia extends HttpServlet
```

L'unico metodo implementato è *doGet* che ha il compito di inviare, come risposta al browser, un insieme di righe contenenti codice HTML per la generazione della pagina Web.

La fotografia da visualizzare è registrata in una sottodirectory *images*.

Il codice completo della servlet è mostrato di seguito.

SERVLET (*Fotografia.java*)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Fotografia extends HttpServlet
{
  public void doGet(HttpServletRequest richiesta,
                    HttpServletResponse risposta)
  throws IOException, ServletException
  {
    risposta.setContentType("text/html");
    PrintWriter out = risposta.getWriter();

    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Fotografia</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<H1>Fotografia 1</H1>");
    out.println("<IMG SRC=images/foto1.jpg WIDTH=200 HEIGHT=200>");
    out.println("</BODY></HTML>");
    out.close();
  }
}
```

Tentando di compilare la precedente classe con le sole librerie standard di Java, si incorre nel seguente errore di compilazione:

```
Fotografia.java:2: error: package javax.servlet does not exist
```

L'errore ci avvisa che la libreria *javax.servlet* non è stata trovata tra l'insieme delle librerie disponibili al compilatore. Nel paragrafo successivo verrà mostrata la corretta modalità di compilazione.

L'AMBIENTE DI ESECUZIONE DELLE SERVLET

Dopo aver creato una servlet, occorre disporre di un **ambiente** adatto per poterla eseguire. È necessaria la presenza di un server Web che sia abilitato per la gestione delle servlet. Se non si dispone di un server Web adatto ad eseguire le servlet, all'indirizzo Internet *http://tomcat.apache.org/*, è possibile trovare un software chiamato *Apache Tomcat*, distribuito liberamente, utile per sviluppare e testare le servlet.
Tomcat è un software gratuito e *open source* che rappresenta l'implementazione delle tecnologie Java Servlet e JSP e che crea l'ambiente all'interno del quale le servlet e le pagine JSP vengono eseguite (*servlet engine*). Tomcat consente di provare le servlet sul proprio computer locale, anche se non si dispone di un server Web in rete.
Oltre che essere un ambiente di prova utilizzabile localmente, Tomcat si integra con i più diffusi server Web ed estende le funzionalità di questi ultimi per l'esecuzione delle servlet.
Per l'installazione di Tomcat nel sistema operativo Windows, è opportuno scaricare dal sito di *Apache Tomcat* il pacchetto *Windows Installer* che configura e installa il software in modo automatico. Durante l'installazione viene richiesta la creazione di un utente amministratore protetto da password. Viene inoltre richiesto di indicare la posizione della *Java Virtual Machine.*
Tra i programmi installati si possono trovare:

- *Configure Tomcat* consente di variare i parametri standard di configurazione;
- *Monitor Tomcat* crea un'icona in basso a destra con la quale si può velocemente avviare (*Start*) e fermare (*Stop*) il server;
- *Tomcat Manager* gestisce le applicazioni create tramite il browser.

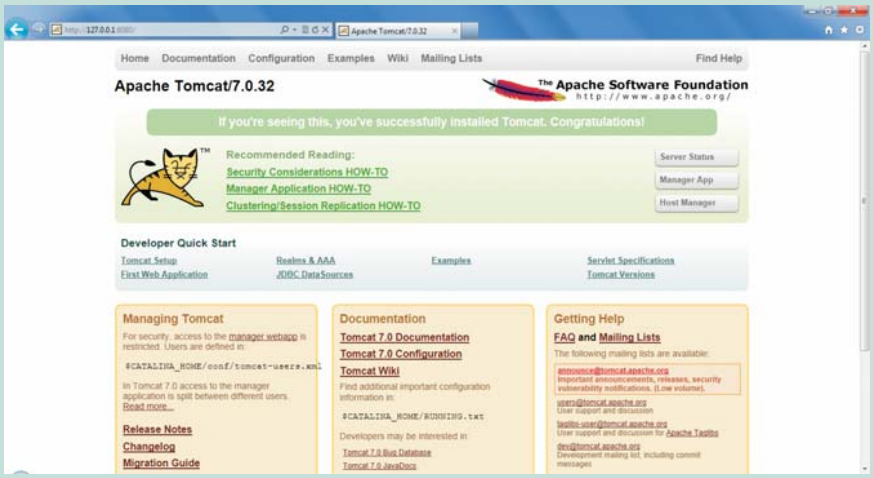
Il server Tomcat è associato alla **porta 8080**. Questo significa che Tomcat può funzionare anche se è attivo un altro server Web, come IIS o Apache, che usa normalmente la porta 80. Per verificare che il server sia stato attivato in modo corretto, occorre quindi scrivere nella casella del browser il seguente indirizzo:

```
http://127.0.0.1:8080/
```

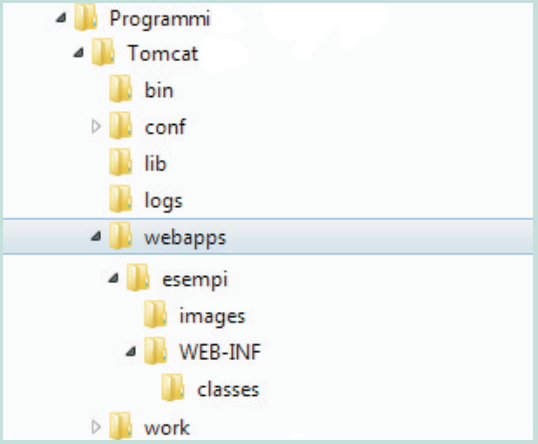
oppure

```
http://localhost:8080/
```

Viene aperta la pagina iniziale del server Tomcat.



La struttura delle sottodirectory di Tomcat è illustrata nella figura seguente, supponendo di aver installato il software nella directory *Programmi*.



La directory personale per le applicazioni, per esempio la directory *esempi*, deve stare obbligatoriamente all'interno della sottodirectory **webapps**. All'interno di *esempi* ci deve essere la directory **WEB-INF** e all'interno di questa la sottodirectory **classes**, dove devono essere messe le servlet compilate, cioè i file *.class*.

Le pagine HTML che richiamano le servlet devono essere memorizzate nella directory *esempi*, creata all'interno di *webapps*.
Le informazioni sulle applicazioni Web del server sono memorizzare in un file speciale, in formato XML, denominato **web.xml**, che deve essere posizionato nella sottodirectory WEB-INF.
(Il linguaggio XML è uno standard, utilizzato in modo esteso nel Web, per la rappresentazione delle informazioni attraverso tag di identificazione.)
Il file *web.xml* si chiama **deployment descriptor** (letteralmente, descrittore del dispiegamento), cioè descrittore dei componenti utilizzati nell'applicazione Web.

La struttura essenziale del file *web.xml* è la seguente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>

  <servlet>
    <servlet-name>Esempio di servlet</servlet-name>
    <servlet-class>Prova</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Esempio di servlet<servlet-name>
    <url-pattern>/Prova1</url-pattern>
  </servlet-mapping>

</web-app>
```

dove:

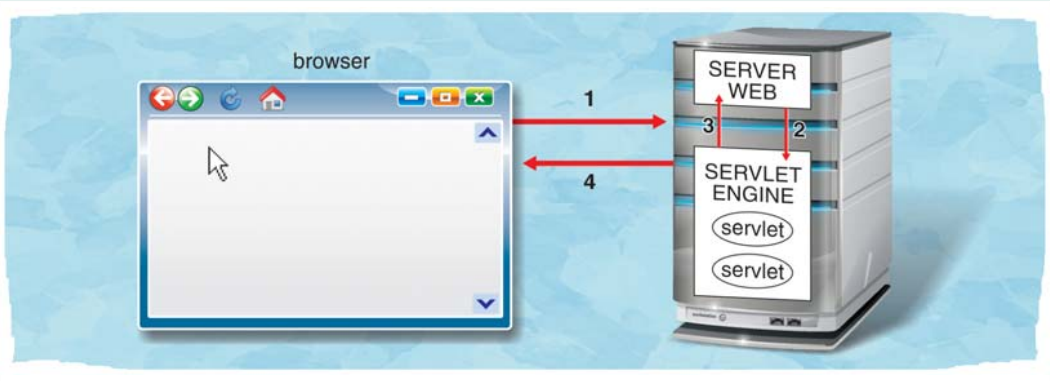
- *<web-app>* è il tag radice che indica l'elenco delle informazioni sulle applicazioni Web
- il tag *<servlet-name>* indica un titolo identificativo della servlet
- il tag *<servlet-class>* indica il nome fisico del file *.class* contenente la servlet compilata
- il tag *<url-pattern>* indica l'URL da scrivere nella casella indirizzo del browser per eseguire la servlet.

Per ogni servlet occorre inserire i due elementi *<servlet>* e *<servlet-mapping>*.

Per creare velocemente il file *web.xml*, si può fare una copia di quello fornito insieme agli esempi installati da Tomcat (sottodirectory *examples* di *webapps*).
I file *.xml* possono essere modificati con un normale editor di testi.
Quindi, supponendo di aver creato il file *Prova.class* e di averlo memorizzato nella sottodirectory *C:\Programmi\Tomcat\webapps\esempi\WEB-INF\classes*, per eseguire la servlet nella casella indirizzo del browser occorre scrivere:

```
http://localhost:8080/esempi/Prova1
```

La figura seguente riassume il funzionamento delle servlet con tutti i passaggi, dalla richiesta da parte dell'utente fino alla risposta del server Web.



1) L'esecuzione delle servlet avviene su richiesta di un client, tipicamente un browser Web, che richiede al server l'attivazione di un particolare servizio.
2) Il server Web comunica questa richiesta al contenitore delle servlet, chiamato anche *servlet engine*, che può essere il software *Tomcat*. Il *servlet engine* carica la servlet e la inizializza. Le successive richieste della stessa servlet non ripeteranno più queste fasi, ma si limiteranno ad attivare la servlet già inizializzata per soddisfare il servizio.
3) I risultati prodotti dalla servlet vengono restituiti al server Web.
4) Il server Web invia i risultati dell'elaborazione, sotto forma di pagina Web, al browser.

3 Compilazione ed esecuzione della servlet

Usiamo come riferimento la servlet creata nel paragrafo precedente per la generazione di una pagina Web con un titolo e una fotografia.
La **compilazione** di una servlet richiede particolare attenzione, perché la libreria che contiene l'API relativa alle servlet non fa parte dell'insieme di librerie fornite con il JDK. Si deve indicare al compilatore dove trovare le librerie specificando il *classpath* come parametro nel seguente modo:

```
javac –classpath C:\Programmi\Tomcat\lib\servlet-api.jar Fotografia.java
```

La libreria necessaria si chiama **servlet-api.jar**: nell'esempio si ipotizza di utilizzare quella fornita da *Tomcat*, installata nella cartella *c:\Programmi\Tomcat\lib*.
Per non ripetere ogni volta la specificazione di *classpath*, in ambiente Windows si possono usare due impostazioni alternative.

- scrivere da linea comandi

```
set CLASSPATH=.;C:\Programmi\Tomcat\lib\servlet-api.jar
```

  (si osservi il punto iniziale per indicare la directory corrente e il punto e virgola di separazione) oppure inserire questa riga in un file *.bat;*
- modificare la variabile di ambiente *CLASSPATH* di Windows: pulsante destro su *Computer*, *Proprietà*, *Impostazioni di sistema avanzate*, scheda *Avanzate*, pulsante *Variabili d'ambiente*, sezione *Variabili di sistema*.

Si deve quindi copiare il codice compilato della servlet, cioè il file con estensione *.class*, in una particolare directory del server Web, che contiene tutte le servlet.
Dopo aver creato la propria directory personale *esempi* all'interno di *webapps*, le servlet compilate devono essere posizionate nella cartella

C:\Programmi\Tomcat\webapps\esempi\WEB-INF\classes.

Occorre poi modificare il file *web.xml* che si trova in

C:\Programmi\Tomcat\webapps\esempi\WEB-INF,

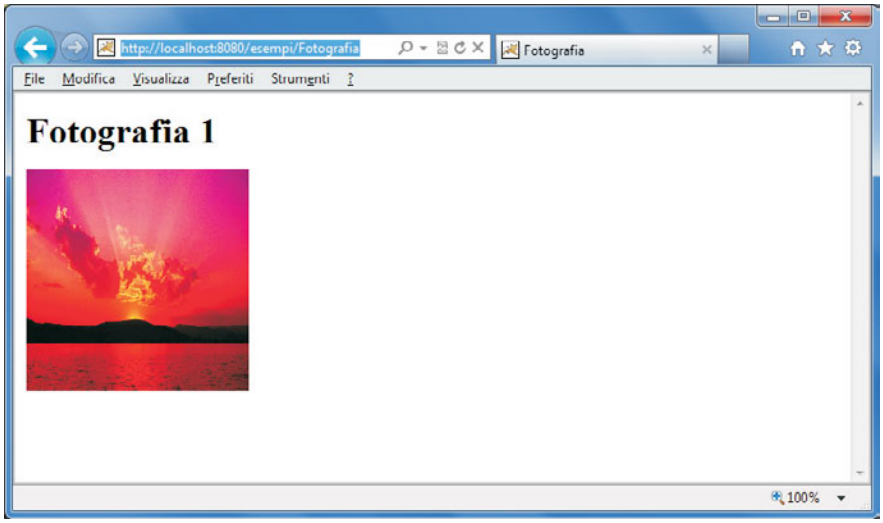
inserendo le seguenti righe nel tag *<web-app>:*

```
<servlet>
  <servlet-name>Fotografia</servlet-name>
  <servlet-class>Fotografia</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>Fotografia</servlet-name>
  <url-pattern>/Fotografia</url-pattern>
</servlet-mapping>
```

Si osservi l'uso della barra / prima di *Fotografia* nel tag *<url-pattern>*.

Con queste impostazioni, per lanciare l'esecuzione di una servlet si deve aprire il browser Web e indicare l'indirizzo corretto della servlet. Se si sta utilizzando *Tomcat* l'indirizzo da scrivere è il seguente:

```
http://localhost:8080/esempi/Fotografia
```



Oltre a specificare direttamente l'indirizzo nella casella del browser, è possibile richiamare le servlet creando un link che punta all'indirizzo della servlet, utilizzando il seguente codice HTML:

```
<A HREF="http://localhost:8080/esempi/Fotografia">Visualizza</A>
```

Si osservi che non si può eseguire la servlet facendo doppio clic sulla sua icona: si deve scrivere l'URL nella casella indirizzo del browser.
Si osservi anche che, facendo clic su *HTML* nel menu *Visualizza del browser*, non viene mostrato il codice della servlet, ma solo il codice HTML della pagina generata.
Il codice della servlet rimane nascosto all'utente finale che vede le pagine tramite il browser.

```
<HTML>
<HEAD>
<TITLE>Fotografia</TITLE>
</HEAD>
<BODY>
<H1>Fotografia 1</H1>
<IMG SRC=images/foto1.jpg WIDTH=200 HEIGHT=200>
</BODY>
</HTML>
```

Nel seguente progetto viene mostrato come creare una pagina dinamica, che possiede contenuti diversi a ogni sua chiamata.

PROGETTO 2

Realizzare, tramite una servlet, un contatore per tenere traccia del numero di visite relative a una certa pagina Web generata dalla stessa servlet.

La classe che descrive questa servlet utilizza l'attributo privato *conta* per memorizzare il numero di volte che la servlet viene richiamata. L'attributo *conta* viene inizializzato con il valore zero all'interno del metodo *init*, in quanto verrà eseguito solo alla prima attivazione della servlet e non più nelle successive esecuzioni.

Il metodo *init*, come anche il metodo *doGet*, utilizza il costrutto *throws* per indicare che l'eccezione non viene gestita al suo interno. Il metodo *init* possiede un argomento che fa riferimento a un oggetto di classe *ServletConfig*, il cui compito è quello di fornire le informazioni sull'ambiente in cui si trova la servlet.

Il metodo *doGet* si occupa di incrementare il valore dell'attributo e restituire la pagina Web generata in modo dinamico.

Il codice completo della servlet è mostrato di seguito.

SERVLET (*Contatore.java*)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class Contatore extends HttpServlet
{
  private int conta;

  public void init(ServletConfig config)
  throws ServletException
  {
    conta = 0;
  }

  public void doGet(HttpServletRequest richiesta,
                    HttpServletResponse risposta)
  throws IOException, ServletException
  {
    conta++;
    risposta.setContentType("text/html");
    PrintWriter out = risposta.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Pagina con contatore</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("Questa pagina e' stata visualizzata ");
    out.println("<H1>" + conta + "</H1> volte.");
    out.println("</BODY>");
    out.println("</HTML>");
    out.close();
  }
}
```

La servlet deve essere compilata e salvata nella directory

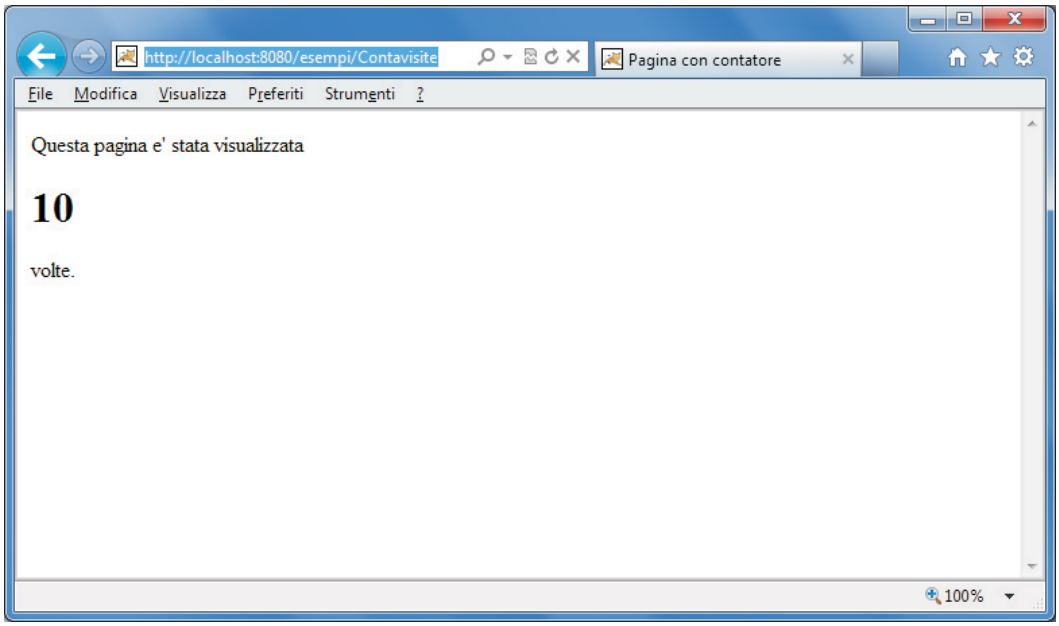
C:\Programmi\Tomcat\webapps\esempi\WEB-INF\classes.

Le righe da inserire nel file *web.xml* sono le seguenti:

```
<servlet>
  <servlet-name>ContaVisite</servlet-name>
  <servlet-class>Contatore</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>ContaVisite</servlet-name>
  <url-pattern>/Contavisite</url-pattern>
</servlet-mapping>
```

Con queste impostazioni, l'esecuzione della servlet si ottiene scrivendo nella casella del browser il seguente indirizzo:

http://localhost:8080/esempi/Contavisite



Ogni volta che si richiama la servlet, si ottiene una pagina Web con il valore aggiornato del contatore.
In questo progetto la variabile che memorizza il numero di accessi alla pagina viene mantenuta in memoria. Questo significa che il valore viene perso quando il server viene spento oppure interrotto. Tutti i valori che si vorrebbero rendere persistenti, andrebbero ovviamente memorizzati su un file.

AUTOVERIFICA

Domande da 6 a 10 pag. 452-453
Problemi da 3 a 7 pag. 454-455

MATERIALI ON LINE

2. Creare le servlet con l'ambiente di sviluppo NetBeans
3. Application Server per sviluppare applicazioni Java Enterprise

4 Interazione con il client

L'utente che utilizza un browser Web può interagire con le servlet, oltre che invocandole direttamente, anche fornendo le informazioni che vengono passate alla servlet sotto forma di parametri. I dati che il client passa a una servlet come parametri sono rappresentati da coppie nome/valore. La servlet riesce a recuperare il valore di un parametro facendo riferimento al nome dello stesso.
Il passaggio di **parametri** si concretizza tramite l'utilizzo di **form** che, come azione associata, prevedano l'attivazione di una servlet.
Si possono usare i campi di un form per raccogliere l'input dell'utente. Ognuno di questi campi viene identificato da un nome che rappresenta il nome del parametro e che deve essere usato all'interno della servlet per recuperare il valore associato.

Per esempio, un form che deve invocare la servlet *Aggiungi* passando come parametro un nome, viene costruito usando il seguente frammento di codice HTML.

```
<FORM METHOD="POST" ACTION="/esempi/Aggiungi">
Nome: <INPUT TYPE="text" NAME="nome" SIZE="20"><BR>
<INPUT TYPE="submit" VALUE="Invia">
</FORM>
```

Quando l'utente preme il pulsante *Invia*, viene richiesta l'esecuzione della servlet *Aggiungi* e viene spedito il contenuto del campo *nome* che verrà interpretato dalla servlet come un parametro.
Le servlet riescono a recuperare il valore di un certo parametro usando il metodo **getParameter** all'interno dei metodi *doGet* o *doPost*. Il metodo *getParameter* è richiamabile dagli oggetti di classe *HttpServletRequest*, ha come argomento il nome del parametro di cui si vuole ottenere il valore e restituisce una stringa. Se il parametro corrisponde a un numero, bisogna effettuare la conversione da stringa a numero usando lo stesso procedimento già visto per il caso di lettura di numeri da tastiera.
Il recupero del parametro *nome* viene eseguito all'interno del metodo *doPost* con la seguente istruzione:

```
String nome = richiesta.getParameter("nome");
```

PROGETTO 3

Si vogliono raccogliere, tramite una servlet, i voti ottenuti dagli studenti in un esame per calcolarne la media. Realizzare una pagina Web e una servlet per permettere la raccolta dei voti e la visualizzazione della media.

PAGINA WEB (*Votazione.htm*)

```
<HTML>
<HEAD>
<TITLE>Votazione</TITLE>
</HEAD>
<BODY>
<H3>Votazione esame </H3>
<P>
<FORM ACTION="/esempi/Voti" METHOD="post">
Inserisci votazione:
<INPUT TYPE="text" SIZE="10" NAME="voto"><BR>
<INPUT TYPE="submit" VALUE="Invia">
</FORM>
</P>
</BODY>
</HTML>
```

La pagina Web deve essere salvata nella directory
C:\Programmi\Tomcat\webapps\esempi

La classe della servlet è formata da due attributi: il valore intero *numeroVoti* conta il numero di voti inseriti, mentre il valore decimale *votoMedio* mantiene aggiornata la media dei voti.
Il metodo *doPost* legge il valore del parametro proveniente dal modulo Web e lo converte in un valore intero con le seguenti istruzioni:

```
votoStr = richiesta.getParameter("voto");
voto = Integer.valueOf(votoStr).intValue();
```

Si noti, nel metodo *doPost*, che la media viene aggiornata ad ogni inserimento tramite la seguente istruzione:

```
votoMedio += (voto - votoMedio) / numeroVoti;
```

Nel seguito è riportato il codice completo della servlet.

SERVLET (*Voti.java*)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Voti extends HttpServlet
{
  private int numeroVoti;
  private float votoMedio;

  public void init(ServletConfig config)
  throws ServletException
  {
     numeroVoti = 0;
     votoMedio = 0f;
  }

  public void doPost(HttpServletRequest richiesta,
                     HttpServletResponse risposta)
  throws IOException, ServletException
  {
     String votoStr;
     int voto;

     votoStr = richiesta.getParameter("voto");
     voto = Integer.valueOf(votoStr).intValue();
     numeroVoti++;
     votoMedio += (voto - votoMedio) / numeroVoti;
     risposta.setContentType("text/html");
     PrintWriter out = risposta.getWriter();
     out.println("<HTML>");
     out.println("<HEAD>");
     out.println("<TITLE>Votazione</TITLE>");
     out.println("</HEAD>");
     out.println("<BODY>");
     out.println("<H3>Votazione media:</H3>");
     out.println("<H1>" + votoMedio + "</H1>");
     out.println("<P>Voti inseriti: " + numeroVoti);
     out.println("<P><A  HREF=/esempi/Votazione.htm>altro  inserimento</A>");
     out.println("</BODY>");
     out.println("</HTML>");
     out.close();
  }
}
```

Al termine dell'esecuzione, la servlet inserisce, alla fine della pagina Web generata, il link alla pagina Web contenente il form per l'inserimento dei dati (*/esempi/Votazione.htm*).

La servlet deve essere compilata e salvata nella directory

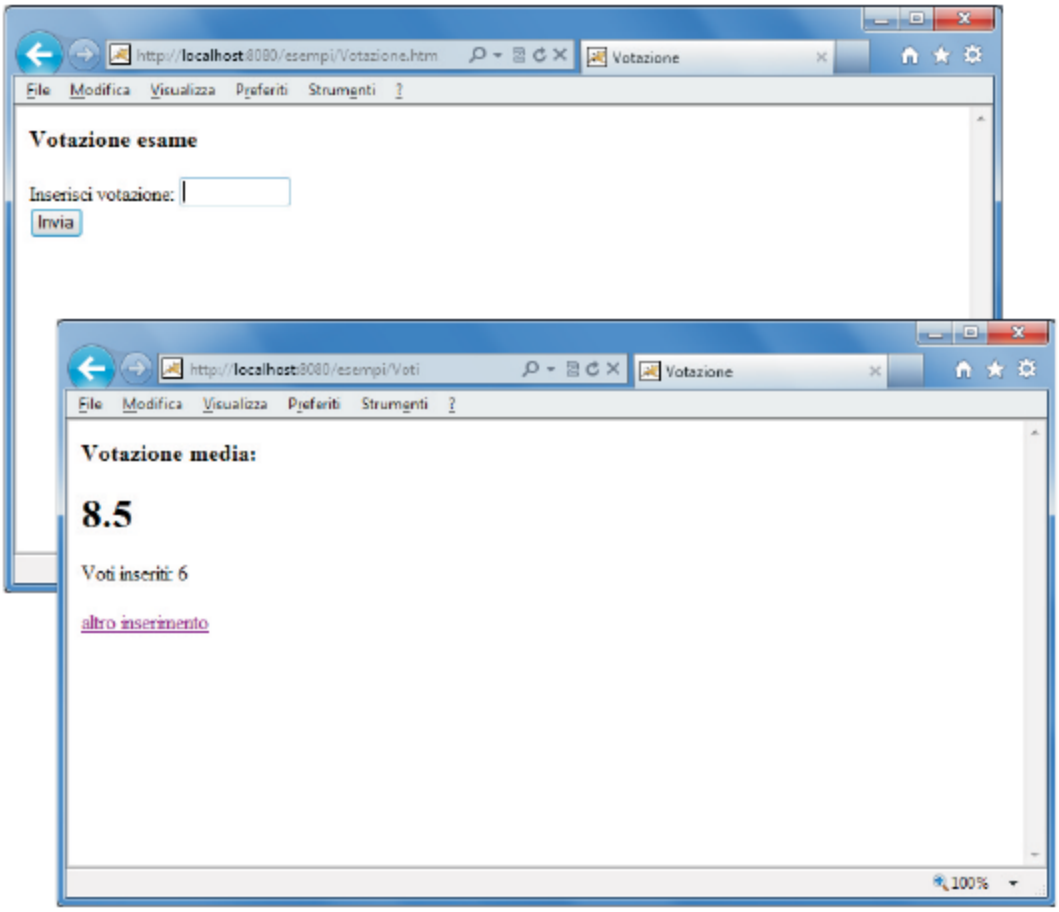
C:\Programmi\Tomcat\webapps\esempi\WEB-INF\classes.

Le righe da inserire nel file *web.xml* sono le seguenti:

```
<servlet>
  <servlet-name>VotiStudenti</servlet-name>
  <servlet-class>Voti</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>VotiStudenti</servlet-name>
  <url-pattern>/Voti</url-pattern>
</servlet-mapping>
```

La pagina Web, che chiama la servlet, deve essere attivata scrivendo nella casella del browser il seguente indirizzo:

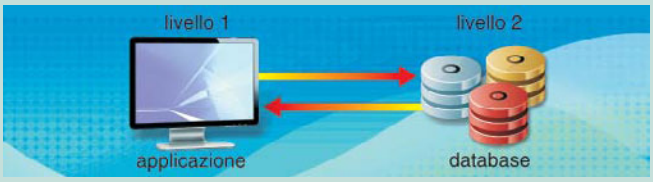
http://localhost:8080/esempi/Votazione.htm



ACCESSO AI DATABASE IN RETE

Nel capitolo precedente abbiamo visto come realizzare l'accesso ai database tramite un'applicazione Java. Nel prossimo paragrafo vedremo come realizzare una connessione a un database remoto usando il browser Web e le servlet. Seguendo questo approccio si possono costruire i siti Web formati da pagine dinamiche, le cui informazioni vengono estratte dai database in rete.
Sia le applicazioni Java che le servlet realizzano la connessione ai database in modo simile, ma con un'architettura di fondo diversa. Le applicazioni, accedono direttamente ai database e usano un'architettura a due livelli (*2-tier*), mentre le servlet per l'accesso ai database usano un'architettura a tre livelli (*3-tier*).

Un'**architettura 2-tier** è composta dai seguenti livelli:
- applicazione Java
- gestore del database.



Il primo livello corrisponde all'applicazione che ha il compito di gestire l'interfaccia e impostare i comandi SQL da passare al gestore del database.
Il secondo livello è rappresentato dal gestore del database che ha il compito di eseguire i comandi SQL e restituire i risultati all'applicazione.
L'approccio a due livelli corrisponde all'architettura C/S: il primo livello è il *client* (applicazione Java) mentre il secondo livello è il *server* (database).

Un'**architettura 3-tier** è composta dai seguenti livelli:
- browser Web
- servlet
- gestore del database.

Il primo livello corrisponde al browser Web e ha il compito di gestire l'interfaccia: visualizza i risultati delle interrogazioni al database e, tramite i form, consente l'inserimento di dati da spedire alla servlet. In questo livello, tutto viene realizzato usando il linguaggio HTML e le pagine Web, siano esse statiche oppure generate dinamicamente dalle servlet.
Il secondo livello è realizzato dalle servlet che vengono eseguite dal server Web. Il loro compito è quello di costruire i comandi SQL e passarli al gestore del database. Quando ricevono la risposta dal database, generano una pagina Web e la restituiscono all'utente.
Il terzo livello è rappresentato dal gestore del database che ha il compito di eseguire i comandi SQL e restituire i risultati alla servlet.
L'approccio a tre livelli ha il vantaggio di separare la parte dell'applicazione, che riguarda l'interfaccia, dalla parte che riguarda l'accesso al database. Un ulteriore vantaggio è rappresentato dalla possibilità di aprire la connessione al database una sola volta, quando la servlet viene inizializzata, evitando di dover aprire una connessione per ogni singolo client che si collega. Quest'ultima situazione è quella che si verifica con un approccio a due livelli.

5 Le servlet per la connessione ai database

Il codice delle servlet per la connessione ai database, oltre ai suoi costrutti tipici, deve contenere le istruzioni JDBC simili a quelle utilizzate nelle applicazioni Java del capitolo precedente.
Come tutte le applicazioni che usano JDBC, la servlet deve specificare l'uso del package **java.sql.***, che fornisce le classi e i metodi per la connessione ai database.

All'interno del metodo *init* viene caricato il driver JDBC-ODBC per la connessione al database e viene creata la connessione con la sorgente di dati.
Questa connessione resta attiva per tutta la durata della servlet e viene chiusa quando viene eseguito il metodo *destroy* della servlet.

Di seguito vengono mostrati due esempi di accesso a un database in rete: il primo è un esempio di interrogazione al database, mentre il secondo esegue un'operazione di manipolazione per inserire nuovi dati.

In questi esempi, il problema che si vuole risolvere con l'uso delle servlet è quello di registrare nuovi utenti memorizzando il cognome, il nome, la password e l'indirizzo di posta elettronica. La necessità di registrare gli utenti ricorre frequentemente all'interno dei siti Web. Lo scopo è quello di raccogliere informazioni sugli utenti per poi permettere solo agli utenti registrati l'accesso ai servizi offerti dal sito stesso.

Negli esempi seguenti, il database è implementato in *Access* ed è contenuto nel file *Anagrafiche.mdb*. Esso è associato tramite ODBC al nome *db2*, specificato come nome della sorgente di dati (DSN).

La tabella *Utenti* del database ha la seguente struttura:

Nome	Tipo	Dimensione
ID	Contatore	
Cognome	Testo	50
Nome	Testo	50
Password	Testo	50
Email	Testo	50

Il campo *ID* è la chiave primaria.

La gestione del database viene eseguita costruendo una servlet per ogni tipo di operazione: ci sarà una servlet per eseguire l'interrogazione, una per eseguire l'inserimento, una per eseguire la cancellazione e così per tutte le altre operazioni.

Il progetto seguente mostra come realizzare una pagina Web per invocare le servlet e gestire il passaggio di parametri. Questa pagina rappresenta l'interfaccia per invocare l'esecuzione delle servlet e per raccogliere l'input dell'utente.

PROGETTO 4

Realizzare una pagina Web in cui è presente un link per mostrare l'elenco degli utenti registrati e un form per la registrazione di nuovi utenti.

PAGINA WEB (*GestioneUtenti.htm*)

```
<HTML>
<HEAD>
<TITLE>Gestione anagrafiche</TITLE>
</HEAD>
<BODY>
<H1>Elenco iscritti</H1>
<A HREF="/esempi/Elenco">Elenco</A>
<H1>Registrazione nuovo utente</H1>
<FORM METHOD="POST" ACTION="/esempi/Registra">
<TABLE BORDER="0" CELLPADDING="5">
<TR>
<TD><B>Cognome</B></TD>
<TD><INPUT TYPE="text" NAME="cognome" SIZE="50"></TD>
</TR>
<TD><B>Nome</B></TD>
<TD><INPUT TYPE="text" NAME="nome" SIZE="50"></TD>
</TR>
<TR>
<TD><B>Password</B></TD>
<TD><INPUT TYPE="password" NAME="password" SIZE="50"></TD>
</TR>
<TR>
<TD><B>Email</B></TD>
<TD><INPUT TYPE="text" NAME="email" SIZE="50"></TD>
</TR>
<TR>
<TD><INPUT TYPE="submit" VALUE="Invia"></TD>
<TD><INPUT TYPE="reset" VALUE="Annulla"></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

La pagina Web deve essere salvata nella directory

C:\Programmi\Tomcat\webapps\esempi.

PROGETTO 5

Realizzare la servlet che visualizza in una tabella il nome e l'indirizzo email di tutti gli utenti registrati.

La classe *Elenco* è composta da tre attributi, utilizzati per gestire la connessione e le interrogazioni al database, e da tre metodi.
Il metodo *init* si occupa di aprire la connessione al database e di associarla all'oggetto *con*.
Il metodo *doGet* esegue un ciclo sugli utenti presenti nel database e costruisce dinamicamente la tabella, usando i tag del linguaggio HTML.
Infine si noti la presenza del metodo *destroy*, in cui sono state inserite le operazioni per la chiusura della connessione con il database.

SERVLET (*Elenco.java*)

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Elenco extends HttpServlet
{
  private Connection con = null;
  private Statement stmt = null;
  private ResultSet rs = null;

  // apre la connessione con il database
  public void init(ServletConfig config)
  throws ServletException
  {
    String url = "jdbc:odbc:db2";
    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      con = DriverManager.getConnection(url, "", "");
    }
    catch(ClassNotFoundException ex)
    {
      System.out.println("Driver non trovato.");
      return;
    }
    catch(SQLException ex)
    {
      System.out.println("Connessione fallita.");
      return;
    }
  }

  public void doGet(HttpServletRequest richiesta,
                    HttpServletResponse risposta)
  throws IOException, ServletException
  {
    risposta.setContentType("text/html");
```

```
      String query = "SELECT Cognome, Nome, Email FROM Utenti ORDER BY Cognome";
      PrintWriter out = risposta.getWriter();
      out.println("<HTML>");
      out.println("<HEAD>");
      out.println("<TITLE>Elenco utenti</TITLE>");
      out.println("</HEAD>");
      out.println("<BODY>");
      out.println("<TABLE BORDER=1>");
      out.println("<TR>");
      out.println("<TH>COGNOME</TH><TH>NOME</TH><TH>EMAIL</TH>");
      out.println("</TR>");
      try
      {
       stmt = con.createStatement();
       rs = stmt.executeQuery(query);
       while (rs.next())
       {
         String str1 = rs.getString("Cognome");
         String str2 = rs.getString("Nome");
         String str3 = rs.getString("Email");
         out.println("<TR>");
         out.println("<TD>" + str1 + "</TD>");
         out.println("<TD>" + str2 + "</TD>");
         out.println("<TD>" + str3 + "</TD>");
         out.println("</TR>");
       }
       stmt.close();
     }
     catch(SQLException ex)
     {
       out.println("Eccezione SQL.");
     }

     out.println("</TABLE>");
     out.println("<P><A HREF=/esempi/GestioneUtenti.htm>" +
                 "Ritorna alla pagina precedente</A>");
     out.println("</BODY>");
     out.println("</HTML>");
     out.close();
   }

   // chiude la connessione con il database
   public void destroy()
   {
     try
     {
       con.close();
     }
     catch(SQLException ex)
     {
       System.out.println("Eccezione SQL.");
     }
   }
 }
```

La servlet deve essere compilata e salvata nella directory
C:\Programmi\Tomcat\webapps\esempi\WEB-INF\classes.

PROGETTO 6

Realizzare una servlet per l'inserimento di un nuovo utente nel database degli utenti registrati.

L'operazione di inserimento nel database viene eseguita nel metodo *doPost*. In particolare, i parametri vengono letti, con il metodo *getParameter*, e vengono successivamente aggiunti alla istruzione SQL parametrica con il metodo *setString*. Se il comando SQL non genera errori, viene mostrata una scritta di conferma dell'avvenuto inserimento, altrimenti viene stampato un messaggio di errore.

SERVLET (*Registra.java*)

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Registra extends HttpServlet
{
  private Connection con = null;

  // apre la connessione con il database
  public void init(ServletConfig config)
  throws ServletException
  {
    String url = "jdbc:odbc:db2";
    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      con = DriverManager.getConnection(url, "", "");
    }
    catch(ClassNotFoundException ex)
    {
      System.out.println("Driver non trovato.");
      return;
    }
    catch(SQLException ex)
    {
      System.out.println("Connessione fallita.");
      return;
    }
  }

  public void doPost(HttpServletRequest richiesta,
                     HttpServletResponse risposta)
  throws IOException, ServletException
  {
    PreparedStatement stmt;
    String query = "INSERT INTO Utenti (Cognome, Nome, Password, Email)"+
                   " VALUES (?, ?, ?, ?)";
```

```
    String str1 = richiesta.getParameter("cognome");
    String str2 = richiesta.getParameter("nome");
    String str3 = richiesta.getParameter("password");
    String str4 = richiesta.getParameter("email");
    risposta.setContentType("text/html");
    PrintWriter out = risposta.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Registrazione</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    try
    {
      stmt = con.prepareStatement(query);
      stmt.setString(1, str1);
      stmt.setString(2, str2);
      stmt.setString(3, str3);
      stmt.setString(4, str4);
      stmt.executeUpdate();
      stmt.close();
      out.println("Nuovo utente registrato: " + str1 + " " + str2);
      out.println("<P><A HREF=/esempi/GestioneUtenti.htm>" +
                  "Ritorna alla pagina precedente</A>");
    }
    catch(SQLException ex)
    {
      out.println("Eccezione SQL.");
    }
    out.println("</BODY>");
    out.println("</HTML>");
    out.close();
  }

  // chiude la connessione con il database
  public void destroy()
  {
    try
    {
      con.close();
    }
    catch(SQLException ex)
    {
      System.out.println("Eccezione SQL.");
    }
  }
}
```

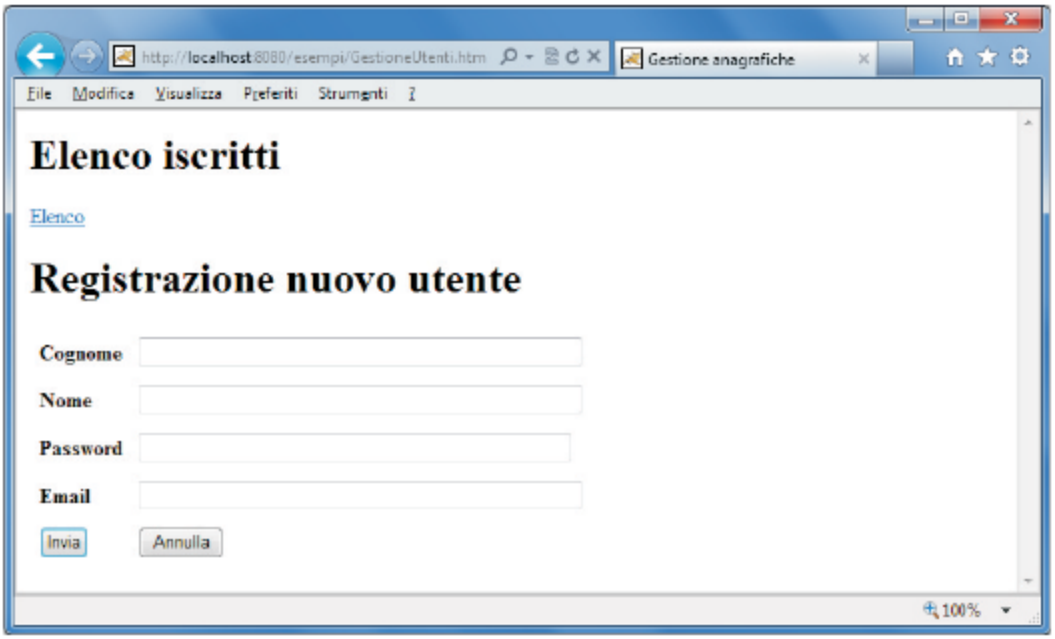
Anche questa servlet deve essere compilata e salvata nella directory
C:\Programmi\Tomcat\webapps\esempi\WEB-INF\classes.

Le righe da inserire nel file *web.xml*, con le informazioni sulle due servlet, sono le seguenti:

```
<servlet>
  <servlet-name>ElencoUtenti</servlet-name>
  <servlet-class>Elenco</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>ElencoUtenti</servlet-name>
  <url-pattern>/Elenco</url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>RegistrazioneUtenti</servlet-name>
  <servlet-class>Registra</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>RegistrazioneUtenti</servlet-name>
  <url-pattern>/Registra</url-pattern>
</servlet-mapping>
```

La pagina Web, che chiama le due servlet *Elenco* e *Registra*, deve essere attivata scrivendo nella casella del browser il seguente indirizzo:

http://localhost:8080/esempi/GestioneUtenti.htm



AUTOVERIFICA
Domande da 11 a 13 pag. 453
Problemi da 8 a 16 pag. 455

MATERIALI ON LINE

4. Gestire le sessioni con le Servlet

6 Le pagine JSP

L'acronimo **JSP** (*Java Server Page*) indica uno dei tanti modi per creare pagine dal contenuto dinamico. Tramite JSP sono definiti alcuni tag che estendono i classici tag del linguaggio HTML e solitamente contengono righe di codice in linguaggio Java.
Quando una pagina in formato JSP viene richiesta da un browser, il server Web abilitato a gestire queste pagine non spedisce direttamente la pagina, ma ne interpreta il contenuto. Le parti contenenti tag HTML vengono restituite così come sono, mentre i comandi contenuti nei tag JSP vengono eseguiti inviando al browser il risultato.

La tecnologia JSP è nata come astrazione delle servlet per rendere più facile la creazione e la manutenzione delle pagine Web generate dinamicamente. Il server Web interpreta le pagine JSP nel seguente modo: durante una prima fase compila il codice JSP e lo traduce in servlet, generando automaticamente i metodi **doGet** e **doPost**, successivamente esegue la servlet che restituisce il risultato al browser. Le richieste successive vengono trattate dalla servlet.

In una servlet il contenuto statico di una pagina HTML è incorporato nel programma all'interno dei metodi *println*, quindi risulta difficile riconoscerlo e identificarlo. Al contrario una pagina JSP incorpora il codice all'interno della pagina HTML: in questo modo il codice HTML è racchiuso nei suoi tag classici, mentre il codice JSP si trova all'interno di tag particolari.
Si osservi anche che ogni modifica alla servlet richiede la ricompilazione del codice, l'eventuale modifica del file *web.xml* e il riavvio dell'applicazione o, in alcuni casi, del server Web.

I tag JSP hanno la seguente struttura **<%...%>** e contengono tipicamente codice Java.
Le pagine JSP possono essere composte da tre elementi:
- tag HTML
- tag JSP
- parti di codice, solitamente in linguaggio Java, chiamati **scriptlet**.

I tag JSP a loro volta si possono dividere in tre categorie:
- comandi
- commenti
- azioni.

I **tag di comando** sono messaggi per il server Web che indicano che cosa fare con il resto della pagina e non producono un risultato visibile. La loro struttura è **<%@...%>**.
I due comandi principali sono *page* e *include*. Si trovano solitamente all'inizio di una pagina JSP, per esempio:

```
<%@ page import="java.util.*" %>
<%@ include file="intestazione.html" %>
```

Il comando **page** è usato per specificare quali sono le librerie che verranno utilizzate nel resto della pagina.
Il comando **include** aggiunge il contenuto di un file nella posizione indicata all'interno di una pagina JSP. Risulta utile per rendere più modulare la costruzione delle pagine dividendole in blocchi.
Con un altro comando **page** è possibile specificare il linguaggio di programmazione utilizzato:

```
<%@ page language="java" %>
```

Un ulteriore comando **page** consente di specificare il tipo di pagina inviata al client (*text/html*) e il set di caratteri utilizzato (*ISO-8859-5*):

```
<%@ page contentType="text/html; charset=ISO-8859-5" %>
```

I **tag di commento** assumono la seguente struttura:

```
<%-- riga di commento ->
```

e sono usati per aggiungere righe di testo che non verranno interpretate ma serviranno solo come documentazione.

I **tag di azione** comprendono le dichiarazioni e le espressioni. Le dichiarazioni permettono di definire variabili e metodi che possono essere utilizzati nel resto della pagina JSP. Le dichiarazioni sono contenute nei seguenti tag **<%! .... %>** e devono contenere le istruzioni Java, terminanti con il punto e virgola.

La dichiarazione di una variabile intera assume la seguente forma:

```
<%! int i=0; %>
```

Le espressioni sono utilizzate per generare un risultato sulla pagina, sia esso il valore di una variabile o il valore di ritorno di un metodo. La struttura di un *tag di espressione* è **<%= .... %>**.

Per visualizzare il valore della variabile intera *i* si usa il seguente tag:

```
<%= i %>
```

mentre per visualizzare il valore dell'area del cerchio che ha come raggio *i* si usa:

```
<%= Math.PI * Math.pow(i, 2) %>
```

Gli *scriptlet* sono dichiarati all'interno dei tag **<%...%>** e contengono righe di codice. Questo codice viene eseguito quando la pagina viene richiesta. Il seguente esempio mostra come può essere visualizzata una scritta usando i tag HTML da H1 a H6:

```
<% for (int i=1; i<=6; i++) { %>
<H<%= i %>>Testo di prova</H<%= i %>>
<% } %>
```

Gli *scriptlet* permettono di lasciare una parentesi graffa aperta sempre che ci sia la rispettiva parentesi chiusa in un successivo *scriptlet* nella pagina. Il precedente frammento di codice esegue un ciclo generando le seguenti sei righe di codice HTML:

```
<H1>Testo di prova</H1>
<H2>Testo di prova</H2>
<H3>Testo di prova</H3>
<H4>Testo di prova</H4>
<H5>Testo di prova</H5>
<H6>Testo di prova</H6>
```

7 Attivazione di una pagina JSP

Le pagine JSP devono essere salvate sul disco con l'estensione **.jsp** all'interno della directory *webapps* di Tomcat oppure in una sua sottodirectory: negli esempi successivi useremo il nome *esempiJSP* per indicare la sottodirectory.
La prima attivazione della pagina JSP dal browser provoca la sua compilazione e la generazione della servlet corrispondente, che viene automaticamente memorizzata nella directory *work* di Tomcat.
Per la compilazione il programma Tomcat usa le librerie contenute in

lib/jsp-api.jar

Un primo semplice progetto mostra come si possa inserire il codice (evidenziato in colore per comodità di lettura) all'interno di una pagina JSP, insieme ai tag HTML.

PROGETTO 7

Visualizzare la data e l'ora corrente con una pagina Web.

PAGINA JSP (*DataOra.jsp*)

```
<%@ page language="java" %>
<%@ page import="java.util.*" %>
<%@ page contentType="text/html; charset=ISO-8859-5" %>
<HTML>
<HEAD>
<TITLE>Data e ora</TITLE></HEAD>
<H1>
<%= new java.util.Date().toString() %>
</H1>
</BODY>
</HTML>
```

Il testo della pagina è salvato nella sottodirectory

C:\Programmi\Tomcat\webapps\esempiJSP

del server Web, con il nome *DataOra.jsp*.

Scrivendo poi nella casella dell'indirizzo del browser

http://localhost:8080/esempiJSP/DataOra.jsp

si ottiene una pagina Web con il titolo, la data e l'ora del sistema. Aprendo dal browser la finestra del sorgente HTML (menu *Visualizza*, scelta *HTML*), si può notare che non viene visualizzato il testo della pagina JSP, ma la pagina HTML generata dinamicamente dalla servlet sul server.

Si noti come, a differenza delle servlet, la pagina JSP non debba essere compilata. La compilazione viene gestita automaticamente da Tomcat durante la prima esecuzione. Il codice sorgente e il relativo file *.class* sono salvati automaticamente nella cartella

C:\Programmi\Tomcat\work\Catalina\localhost\esempiJSP\org\apache\jsp

Il progetto seguente mostra come sia possibile costruire una pagina JSP integrando, in più punti dello stesso testo, codice e tag HTML. Il codice è separato dal testo e dai tag HTML attraverso le coppie di delimitatori <% … %>.

PROGETTO 8

Costruire la tavola pitagorica.

La pagina JSP costruisce una tabella di 10 righe e 10 colonne; ciascuna cella contiene il prodotto del numero di riga per il numero di colonna. La procedura usa due cicli *for* annidati.

PAGINA JSP (*Tavola.jsp*)

```
<%@ page language="java" %>
<%@ page import="java.util.*" %>
<%@ page contentType="text/html; charset=ISO-8859-5" %>
<HTML>
<HEAD>
<TITLE>Tavola Pitagorica</TITLE>
</HEAD>
<CENTER>
<H2>Tavola Pitagorica </H2>
<TABLE BORDER="1">
<% for (int i=1; i <=10; i++) { %>
<TR>
<% for (int j=1; j <=10; j++) { %>
<TD ALIGN="center" WIDTH="10%">
<%= i*j %>
</TD>
<% } %>
</TR>
<% } %>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

Il testo della pagina è salvato nella sottodirectory

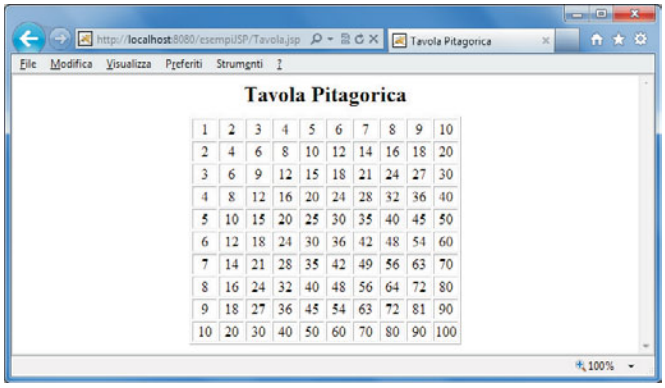
C:\Programmi\Tomcat\webapps\esempiJSP

del server Web, con il nome *Tavola.jsp*.

Scrivendo poi nella casella dell'indirizzo del browser

http://localhost:8080/esempiJSP/Tavola.jsp

si ottiene una pagina Web con i 100 numeri della tavola organizzati in una tabella.



1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

8 Passaggio di parametri alla pagina JSP

Una delle caratteristiche più importanti di tutti i linguaggi per creare pagine Web in modo dinamico è rappresentata dalla possibilità di interagire con l'utente che utilizza la pagina Web. Con le pagine JSP l'interazione avviene nel momento in cui l'utente invia una richiesta al server Web: quest'ultimo genera come risposta una pagina Web creata dinamicamente in quel momento.
L'interazione con l'utente prevede anche la possibilità di inviare alcuni dati come **parametri** della chiamata alla pagina JSP.
In modo analogo a quanto visto nei paragrafi precedenti per le servlet, il passaggio di parametri a una pagina JSP viene gestito attraverso i **form**, cioè i moduli del linguaggio HTML, che permettono la costruzione di un'interfaccia grafica, formata da caselle di testo e da pulsanti. Tramite questa interfaccia, l'utente può inserire i valori e inviarli al server Web come parametri dello script.

PROGETTO 9

Creare un modulo per raccogliere i dati di un utente che desidera iscriversi a un servizio Web (cognome, nome, password e indirizzo di e-mail).

Il modulo è formato da quattro caselle di testo e da due pulsanti di comando e viene rappresentato in HTML con il seguente codice.

PAGINA WEB (*Richiesta.htm*)

```
<HTML>
<HEAD>
<TITLE>Inserimento dati</TITLE>
</HEAD>
<BODY>
<H1>Iscrizione al servizio Web</H1>
<FORM ACTION="/esempiJSP/Iscrizione.jsp" METHOD="post">
<TABLE BORDER="0" CELLPADDING="5">
<TR>
<TD><B>Cognome</B></TD>
<TD><INPUT TYPE="text" NAME="cognome" SIZE="50"></TD>
</TR>
<TD><B>Nome</B></TD>
<TD><INPUT TYPE="text" NAME="nome" SIZE="50"></TD>
</TR>
<TR>
<TD><B>Password</B></TD>
<TD><INPUT TYPE="password" NAME="password" SIZE="50"></TD>
</TR>
<TR>
<TD><B>Email</B></TD>
<TD><INPUT TYPE="text" NAME="email" SIZE="50"></TD>
</TR>
<TR>
<TD><INPUT TYPE="submit" VALUE="Invia"></TD>
<TD><INPUT TYPE="reset" VALUE="Annulla"></TD>
```

```
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

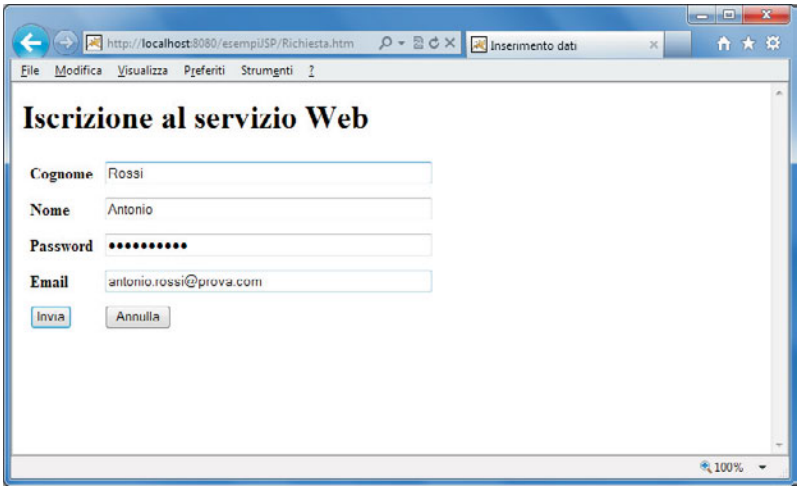
Si noti l'uso del tipo *password* per la casella di testo nell'inserimento della password, per non rendere visibili i caratteri inseriti.
La pagina Web viene salvata nella sottodirectory

C:\Programmi\Tomcat\webapps\esempiJSP

con il nome *Richiesta.htm*.

La pagina viene poi aperta nel browser scrivendo nella casella dell'indirizzo

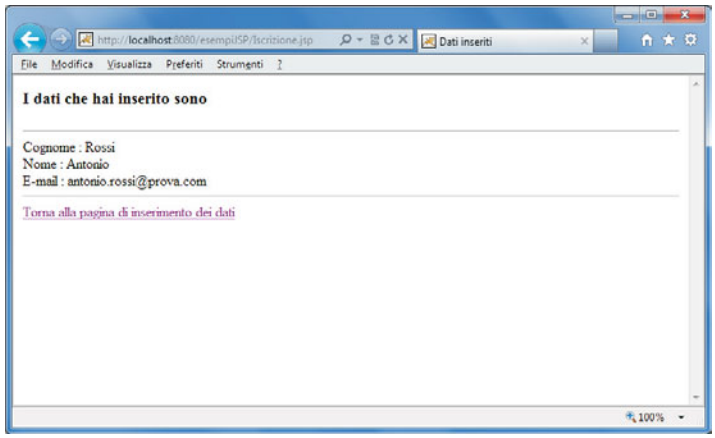
http://localhost:8080/esempiJSP/Richiesta.htm



Quando si fa clic sul pulsante *Invia*, il browser richiama la pagina *Iscrizione.jsp*, indicata nell'intestazione del modulo come valore dell'attributo ACTION. Il file *jsp* si trova nella stessa sottodirectory della pagina Web.

C:\Programmi\Tomcat\ webapps\esempiJSP

Il browser aggiunge automaticamente alla richiesta tutti i campi presenti nel modulo. Per ogni campo crea un parametro, avente come nome il nome del campo (indicato dall'attributo NAME del tag <INPUT>) e come valore ciò che l'utente ha inserito nella casella di testo. In questo modo viene realizzato il passaggio di parametri a una pagina JSP tramite l'utilizzo dei form HTML.
L'attributo METHOD del tag <FORM> ha il compito di indicare al browser quale metodo deve utilizzare per inviare i campi del modulo al server Web. In questo caso è stato usato il metodo *post*.
In risposta all'inserimento dei dati, il server, eseguendo la servlet ottenuta dalla pagina JSP *Iscrizione.jsp*, crea dinamicamente una pagina Web con il riassunto dei dati inseriti e offre la possibilità, attraverso un link, di tornare alla pagina di inserimento dei dati. La pagina *Iscrizione.jsp* dovrebbe poi avere anche il compito di registrare i dati in un database, come vedremo nel prossimo paragrafo.



PAGINA JSP (*Iscrizione.jsp*)

```
<%@ page language="java" %>
<%@ page contentType="text/html; charset=ISO-8859-5" %>
<%
// legge i valori ricevuti dal form HTML
String Stringa1 = new String
(request.getParameter("cognome"));
String Stringa2 = new String
(request.getParameter("nome"));
String Stringa3 = new String
(request.getParameter("email"));
%>
<HTML>
<HEAD>
<TITLE>Dati inseriti</TITLE>
</HEAD>
<BODY>
<H3>I dati che hai inserito sono </H3><HR>
Cognome : <%= Stringa1 %><BR>
Nome : <%= Stringa2 %><BR>
E-mail : <%= Stringa3 %><BR>
<HR>
<A HREF="/esempiJSP/Richiesta.htm">Torna alla pagina di
inserimento dei dati</A>
</BODY>
</HTML>
```

Per visualizzare i dati si può usare anche il metodo *println* per l'oggetto *out*:

```
<%
out.println("Cognome: "+Stringa1+"<br>");
out.println("Nome: "+Stringa2+"<br>");
out.println("E-mail: "+Stringa3+"<br>");
%>
```

La pagina JSP viene salvata nella sottodirectory

C:\Programmi\Tomcat\webapps\esempiJSP

con il nome *Iscrizione.jsp*.

IL PASSAGGIO DI PARAMETRI TRAMITE L'INDIRIZZO URL

Un modo diverso per passare i parametri a una pagina JSP consiste nell'**aggiungere i parametri all'indirizzo URL** della pagina JSP che si sta richiamando.
L'indirizzo assume la seguente struttura generale:

pagina.jsp?nome=valore

dove
- *nome* indica il nome del parametro che viene passato a *pagina.jsp*,
- *valore* indica il valore del parametro.
- il carattere ? separa il nome dello script dai parametri.

È possibile passare anche più di un parametro: in questo caso si devono separare i parametri, cioè le coppie nome=valore, tramite il simbolo &.

pagina.jsp?nome1=valore1&nome2=valore2

Gli eventuali spazi contenuti nei nomi dei parametri, o nei valori stringa assegnati, sono convertiti automaticamente in **%20**, che indica il valore esadecimale del carattere spazio (valore ASCII decimale = 32).
Quando viene richiesta una pagina JSP con passaggio di parametri, lo script costruisce una variabile che fa riferimento al parametro ricevuto. Il nome e il valore della variabile corrispondono al nome e al valore del parametro.

Per esempio, con riferimento alla pagina JSP presentata nel paragrafo precedente, è possibile inviare i valori dei parametri con il seguente URL:

```
http://localhost:8080/esempiJSP/Iscrizione.jsp?cognome=Rossi&nome=
Antonio&password=rosant1234&email=antonio.rossi@prova.com
```

Se il nome fosse *Marco Antonio*, lo spazio inserito con la barra spaziatrice verrebbe convertito automaticamente in %20:

```
http://localhost:8080/esempiJSP/Iscrizione.jsp?cognome=Rossi&nome=
Marco%20Antonio&password=rosant1234&email=antonio.rossi@prova.com
```

9 Accesso ai database con JSP

Le operazioni di manipolazione e, soprattutto, le interrogazioni sui database di un server Web sono svolte in modo semplice ed efficiente utilizzando i comandi SQL all'interno del codice. I comandi SQL possono essere *Insert*, *Update*, *Delete* per le operazioni di manipolazione e *Select* per le interrogazioni.

L'accesso ai dati contenuti nelle tabelle di un database si realizza nelle pagine JSP con modalità del tutto analoghe a quanto già visto per le servlet.

Per eseguire il comando *Select* di SQL occorre usare, come per le servlet, il metodo **executeQuery** di JDBC.

Per un confronto tra la servlet e la pagina JSP si riveda il file *Elenco.java* del Progetto 5 di questo capitolo.

PROGETTO 10

Visualizzare l'elenco con cognome, nome e indirizzo e-mail degli utenti registrati nel database.

All'inizio della pagina si deve aggiungere il tag di comando

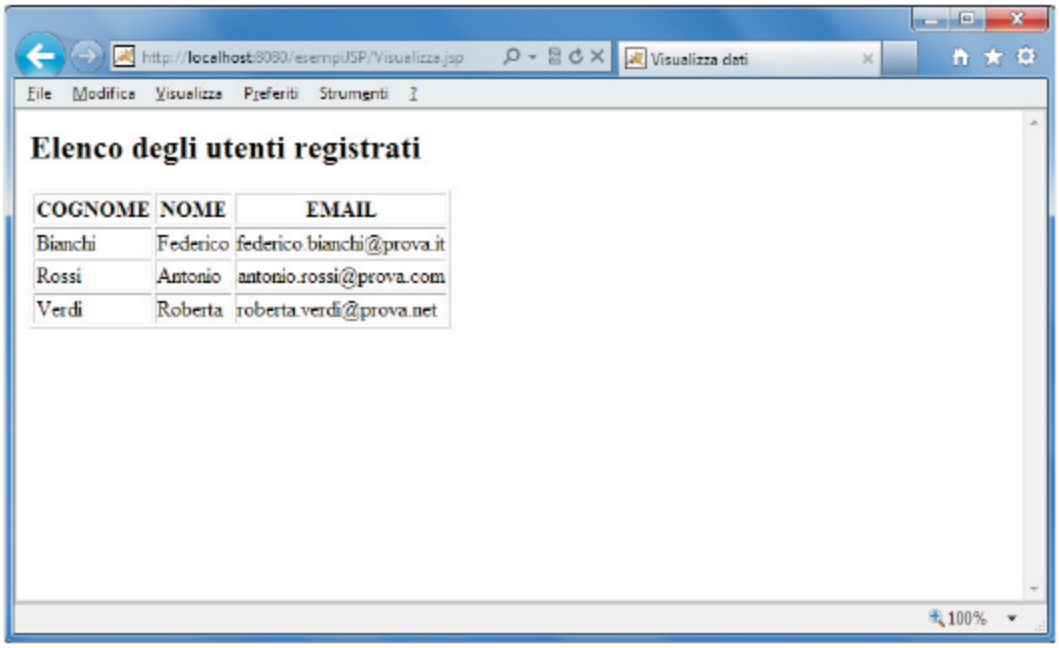
```
<%@ page import="java.sql.*" %>
```

per l'importazione del package **java.sql.*** con le classi e i metodi di connessione e gestione dei database.

PAGINA JSP (*Visualizza.jsp*)

```
<%@ page language="java" %>
<%@ page import="java.sql.*" %>
<%@ page contentType="text/html; charset=ISO-8859-5" %>
<HTML>
<HEAD>
<TITLE>Visualizza dati</TITLE>
</HEAD>
<BODY>
<H2>Elenco degli utenti registrati</H2>
<TABLE BORDER="1">
<TR>
<TH>COGNOME</TH>
<TH>NOME</TH>
<TH>EMAIL</TH>
</TR>
<%
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;

String url = "jdbc:odbc:db2";
try
{
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  conn = DriverManager.getConnection(url, "", "");
  stmt = conn.createStatement();
}
catch(ClassNotFoundException ex)
{
  out.println("Driver non trovato.");
}
catch(SQLException ex)
{
  out.println("Connessione fallita.");
}
```

```
String query = "SELECT Cognome, Nome, Email
                FROM Utenti ORDER BY Cognome";

try
{
  rs=stmt.executeQuery(query);
  while (rs.next())
  {
    String str1 = rs.getString("Cognome");
    String str2 = rs.getString("Nome");
    String str3 = rs.getString("Email");
    out.println("<TR>");
    out.println("<TD>" + str1 + "</TD>");
    out.println("<TD>" + str2 + "</TD>");
    out.println("<TD>" + str3 + "</TD>");
    out.println("</TR>");
  }

  stmt.close();
  conn.close();
}
catch(SQLException ex)
{
  out.println("Eccezione SQL.");
}
%>
</TABLE>
</BODY>
</HTML>
```



OPERAZIONI DI MANIPOLAZIONE SUL DATABASE CON JSP

Il progetto seguente mostra come si possa realizzare l'operazione di inserimento di nuove righe in una tabella del database, utilizzando il comando di manipolazione *Insert* del linguaggio SQL all'interno del codice JSP. Anche in questo esempio si utilizza il metodo **executeUpdate** di JDBC, in modo analogo a quanto visto per le servlet.
L'esempio rappresenta il completamento della pagina JSP presentata nel Paragrafo 8 con l'aggiunta della parte di registrazione nel database dei dati dell'utente acquisiti tramite il form contenuto nella pagina Web *Richiesta.htm*.

PROGETTO 11

Inserire nel database i dati di un nuovo utente, acquisiti come parametri attraverso il form di una pagina Web.

PAGINA JSP (*Iscrizione.jsp*)

```
<%@ page language="java" %>
<%@ page import="java.sql.*" %>
<%@ page contentType="text/html; charset=ISO-8859-5" %>
<%
// legge i valori ricevuti dal form HTML
String Stringa1 = new String (request.getParameter("cognome"));
String Stringa2 = new String (request.getParameter("nome"));
String Stringa3 = new String (request.getParameter("password"));
String Stringa4 = new String (request.getParameter("email"));
%>
<HTML>
<HEAD>
<TITLE>Registrazione dati</TITLE>
</HEAD>
<BODY>
<H3>I dati che hai inserito sono </H3><HR>
<%
out.println("Cognome: " + Stringa1 + "<br>");
out.println("Nome: " + Stringa2 + "<br>");
out.println("E-mail: " + Stringa4 + "<br>");
%>
<HR>
<%
Connection conn = null;
Statement stmt = null;
String url = "jdbc:odbc:db2";
try
{
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  conn = DriverManager.getConnection(url, "", "");
  stmt = conn.createStatement();
}
catch(ClassNotFoundException ex)
{
  out.println("Driver non trovato.");
}
```

```
catch(SQLException ex)
{
  out.println("Connessione fallita.");
}

// stringa SQL per l'inserimento
String query = new String (
  "INSERT INTO Utenti (Cognome, Nome, Password, Email) " +
  " VALUES ('" + Stringa1 + "','" + Stringa2
  + "','" + Stringa3 + "','" + Stringa4 +"')"
);

try
{
  stmt.executeUpdate(query);
  out.println("Nuovo utente registrato: " + Stringa1 + " " + Stringa2);
  stmt.close();
  conn.close();
}
catch(SQLException ex)
{
  out.println("Eccezione SQL.");
}
%>
<BR>
<A HREF="/esempiJSP/Richiesta.htm">
Torna alla pagina di inserimento dei dati
</A>
</BODY>
</HTML>
```

Si deve fare molta attenzione alla costruzione della stringa *query* contenente il comando *Insert* di SQL, utilizzando gli apici per delimitare i valori stringa dei campi e separando i campi con la virgola:

```
String query = new String (
  "INSERT INTO Utenti (Cognome, Nome, Password, Email) " +
  " VALUES ('"+ Stringa1 + "','" + Stringa2 +
  "','" + Stringa3 + "','" + Stringa4 + "')"
);
```

Se i campi sono di tipo numerico, i valori non devono essere racchiusi tra apici.

Seguendo la traccia del progetto precedente, si possono costruire le pagine per effettuare operazioni di modifica e cancellazione delle righe di una tabella utilizzando il metodo **executeUpdate** e i comandi *Update* e *Delete* del linguaggio SQL.

AUTOVERIFICA

Domande da 14 a 17 pag. 454
Problemi da 17 a 34 pag. 455-456
Problemi di riepilogo da 35 a 36 pag. 457

MATERIALI ON LINE

5. Creare i pacchetti per le applicazioni Web (JAR, WAR, EAR)

AUTOVERIFICA

DOMANDE

L'architettura Client/Server

1 Quali di queste affermazioni, riferite all'architettura C/S, sono vere (V) e quali false (F)?

a) Permette a diversi calcolatori collegati in rete di utilizzare e condividere le risorse V F
b) Può essere applicata anche tra calcolatori non collegati in rete V F
c) In questa architettura si distinguono due entità: il client e il server V F
d) È solo un'architettura hardware V F
e) È l'architettura di base della rete Internet V F

2 Che cosa indica il termine *localhost*?

a) Il server Web sul computer locale.
b) Il computer remoto dell'utente.
c) Un'applicazione locale sul disco C:.
d) Un browser locale.

3 Completa le frasi seguenti usando una tra le parole elencate alla fine della domanda.

a) Il servizio …….. consente di condividere un insieme di documenti chiamati pagine Web.
b) Il ……. risponde alle richieste inviando una pagina Web.
c) Il ……. viene utilizzato per richiedere le pagine Web.
d) Il protocollo ……. stabilisce come devono essere formulate le richieste e generate le risposte di pagine Web.

Internet, browser Web, server Web, HTTP, WWW, FTP

4 Quali di queste affermazioni, riferite al server Web, sono vere (V) e quali false (F)?

a) Il suo compito è quello di rispondere alle richieste dei browser V F
b) Le estensioni del server Web sono state introdotte per generare pagine statiche V F
c) Si possono introdurre contenuti dinamici anche dal lato client V F
d) Le pagine dinamiche vengono costruite dal server in risposta alle richieste del browser V F

5 Quale di queste sigle *non* corrisponde a pagine dinamiche che integrano codice e tag HTML?

a) ASP
b) JSP
c) HTTP
d) PHP

Le servlet

6 Ordina i seguenti metodi seguendo l'ordine di invocazione del server Web.

a) doGet
b) service
c) init
d) destroy
e) doPost

7 Come viene definita una classe che implementa una servlet?

a) public class <nome servlet> implements HttpServlet
b) public class <nome servlet> extends HttpServlet
c) public <nome servlet> extends HttpServlet
d) public class <nome servlet> extends Applet

8 Quali di queste affermazioni, riferite al software Tomcat, sono vere (V) e quali false (F)?

a) Tomcat è un esempio di servlet engine V F
b) Il server Tomcat è associato alla porta 8080 V F
c) Le applicazioni Web in Tomcat possono essere registrate in una directory qualsiasi V F
d) Le informazioni sulle applicazioni Web del server sono memorizzare in un file speciale denominato *web-apps* V F

9 Quali di queste affermazioni, riferite alle servlet, sono vere (V) e quali false (F)?

a) Funzionano secondo un'architettura di tipo Client/Server V F
b) Le richieste sono gestite dalla classe HttpServletResponse V F
c) Le richieste sono gestite dalla classe HttpServletRequest V F
d) Le richieste possono essere di tipo POST e di tipo GET V F
e) Il metodo *init* è eseguito ogni volta che la servlet viene invocata V F

10 Quali dei seguenti indirizzi occorre scrivere nella casella del browser per eseguire in Tomcat la servlet *prova.class* che si trova nella directory *esempi*?

a) http://localhost/esempi/prova
b) http://localhost:80/esempi/prova
c) http://localhost:8080/esempi/prova
d) http://localhost:8080/prova

Parametri e accesso al database con le servlet

11 Qual è il metodo utilizzato per leggere i parametri passati dall'utente?

a) getString
b) getName
c) getParameter
d) getParameters

12 Quale package, tra i seguenti, *non* è utilizzato in una servlet per la connessione ai database?

a) java.io.*
b) java.awt.*
c) javax.servlet.*
d) javax.servlet.http.*
e) java.sql.*

13 Qual è, tra i seguenti, il secondo livello di un'applicazione costruita seguendo un approccio 3-tier?

a) browser Web
b) servlet
c) database

AUTOVERIFICA

Le pagine JSP

14 Quali sono i marcatori che racchiudono il codice della pagina JSP?

a) <* *>
b) <? ?>
c) <% %>
d) // //

15 Quale delle seguenti istruzioni di una pagina JSP è scritta in modo corretto per assegnare alla variabile *Nome* il valore proveniente da una casella di testo avente la proprietà *Name* uguale a *Testo1*, contenuta nel form di una pagina HTML?

a) <% String Testo1 = new String (request.getParameter("Nome")); %>
b) <% String Nome = new String (Form("Testo1")); %>
c) <% String Request.Nome = new String (getParameter("Testo1")); %>
d) <% String Nome = new String (request.getParameter("Testo1")); %>

16 Quale tag di comando occorre inserire nelle pagine JSP per poter usare il package con le classi e i metodi di connessione e gestione dei database?

a) <%@ page import="java.sql.*" %>
b) <%@ page import="java.util.*" %>
c) <%@ page import="java.jsp.*" %>
d) <%@ page import="java.servlet.*" %>

17 Qual è la codifica corretta dell'URL per passare due parametri (*codice* e *provincia*) alla pagina *Cerca.jsp*?

a) Cerca.jsp&codice=19?provincia=PN
b) Cerca.jsp?codice=19&provincia=PN
c) Cerca.jsp?codice=19;provincia=PN
d) Cerca.jsp!codice=19&provincia=PN

PROBLEMI

L'architettura client/server

1 Controllare se il Web server IIS è attivo sul computer.

2 Copiare nella cartella principale del Web Server (*wwwroot*) un file di tipo HTML. Aprire poi il browser e scrivere il suo indirizzo nella forma http://... per poterlo visualizzare.

Le servlet

3 Predisporre l'ambiente per eseguire le servlet, installando il software Tomcat e creando una directory dove registrare le servlet.

4 Costruire una servlet che genera una pagina Web statica, contenente due immagini con le rispettive didascalie.

5 Costruire una servlet che genera una pagina Web statica, contenente una tabella con 4 colonne e tre righe. La riga di intestazione contiene il nome dei primi quattro mesi dell'anno.

6 Realizzare una servlet che visualizza in una pagina Web il doppio del numero precedente, a partire da un valore fissato in fase di inizializzazione della servlet.

7 Realizzare una servlet che visualizza in una pagina Web un valore incrementato del 10% rispetto al precedente, a partire da un valore fissato in fase di inizializzazione della servlet.

Parametri e accesso al database con le servlet

8 Una pagina Web permette l'inserimento della quotazione giornaliera di un titolo di Borsa. Tramite una servlet viene restituita la variazione percentuale rispetto al precedente valore inserito.

9 Tramite un form di una pagina Web viene inserita la temperatura in una certa città. Dopo ogni inserimento vengono mostrate la temperatura minima e massima fino a quel momento registrate.

10 Creare una pagina Web con lo scopo di esprimere le preferenze su un elenco di libri. Utilizzare una casella di testo per inserire il voto (compreso tra 1 e 10). Visualizzare il voto medio calcolato con una servlet.

11 Creare un database contentente la tabella *Computer* con i campi: processore, memoria, harddisk, monitor. Scrivere una servlet che mostra tutti i record contenuti nella tabella.

12 Realizzare una servlet che estrae dalla tabella del problema precedente tutti i computer con più di 2048 MB di memoria.

13 Realizzare una servlet che estrae dalla tabella *Computer* tutti i computer con un monitor da 17 pollici e non più 100 GB di harddisk.

14 Realizzare una servlet che aggiunge alla tabella *Computer* un nuovo computer.

15 Realizzare una servlet che mostra tutti i computer che hanno un processore uguale a quello indicato dall'utente come parametro.

16 Data una tabella *Voti* con nome dello studente, materia, voto, realizzare un'applicazione che fornisca a un docente un insieme di funzionalità da realizzare tramite una pagina Web e una servlet.
- Inserire il voto di un certo studente in una certa materia.
- Mostrare tutti i voti di un certo studente.

Le pagine JSP

17 Visualizzare dal browser il codice HTML di una pagina JSP generata dal server.

18 Costruire una pagina JSP che visualizza una tabella a due colonne, avente nella prima colonna i numeri interi da 1 a 10 e nella seconda colonna i quadrati dei numeri.

19 Costruire una pagina JSP che visualizza una tabella avente tre righe e due colonne: ogni riga contiene a sinistra un'immagine registrata sul disco e a destra il nome del file contenente l'immagine.

20 Scrivere la pagina JSP che riceve tramite un form HTML un numero compreso tra 1 e 5 e visualizza la rappresentazione del numero in lettere (uno, due, tre, quattro, cinque).

AUTOVERIFICA

21 Scrivere la pagina JSP che calcola l'area e la circonferenza di un cerchio del quale viene fornita la misura del raggio attraverso un form HTML.

22 Un utente fornisce con un form HTML il proprio cognome, nome e sesso (M/F); una pagina JSP risponde all'inserimento con un messaggio di saluto "Buongiorno Sig." o "Buongiorno Sig.ra", a seconda del sesso, seguito dal cognome dell'utente.

23 Costruire una pagina JSP per calcolare l'imposta da pagare, conoscendo l'imponibile e l'aliquota.

24 Calcolare l'area di un triangolo dopo aver acquisito, attraverso il form di una pagina Web la misura della base e dell'altezza.

25 Leggere con una pagina JSP le righe di una tabella di un database, visualizzando solo il primo e il secondo campo di ogni riga.

26 Data una tabella *Anagrafica* di un database, costruire la pagina JSP che visualizza il cognome e nome delle persone che abitano nella provincia di Genova.

27 La tabella *Movimenti* di un database contiene i dati relativi alle vendite, con numero di registrazione, data, codice cliente e importo: visualizzare in una pagina Web l'elenco dei movimenti di un cliente aventi importo superiore a una cifra prefissata. Il codice del cliente e la cifra prefissata sono forniti dall'utente tramite il form di una pagina HTML.

28 Data la tabella di un database contenente le informazioni sugli studenti di una scuola, con matricola, cognome, nome e classe frequentata, visualizzare l'elenco con cognome e nome degli studenti di una classe richiesta tramite la casella di testo di un form HTML.

29 La tabella *Atleti* di un database contiene le informazioni sui concorrenti a una gara sportiva internazionale. Costruire la pagina Web per visualizzare l'elenco alfabetico con cognome e nome degli atleti di una Nazione richiesta tramite una casella di testo.

30 Costruire una pagina JSP che riceve come parametri due numeri denominati come Prezzo e PercentualeSconto e calcola il prezzo scontato. Scrivere poi l'URL da inserire nella casella Indirizzo del browser per passare i valori alla pagina JSP.

31 Costruire un form all'interno di una pagina Web che consenta di registrare i dati di una persona che desidera iscriversi a un convegno (cognome, nome ed e-mail). Il nuovo iscritto viene aggiunto nella tabella *Iscritti* già esistente in un database.

32 Aggiornare il numero di telefono di un persona di cui viene fornito il codice. I dati della persona si trovano nella tabella *Anagrafica* di un database. Il numero di telefono e il codice sono forniti dall'utente attraverso un form all'interno della pagina Web.

33 Creare una pagina JSP per inserire un nuovo studente nel database di una scuola, fornendo i dati tramite il form di una pagina HTML e utilizzando il comando *Insert* del linguaggio SQL.

34 All'inizio del nuovo anno scolastico la classe degli studenti, già iscritti e provenienti dall'anno precedente, viene cambiata. La pagina JSP aggiorna gli studenti dopo aver acquisito il codice dello studente, e la nuova classe da assegnare, attraverso il form di una pagina HTML. Utilizzare il comando *Update* del linguaggio SQL.

PROBLEMI DI RIEPILOGO

35 Creare un database con i dati delle Regioni e delle città organizzandoli con due tabelle:

Regioni (Descrizione, Abitanti, Superficie)
Città (Nome, Abitanti, Superficie, Capoluogo, *NomeRegione*)

Descrizione e *Nome* sono le chiavi primarie, *NomeRegione* è la chiave esterna che indica la Regione di appartenenza della città. *Capoluogo* è una variabile booleana (Si/No) che indica se la città è capoluogo di Regione.

- creare le pagine Web per inserire i dati nelle tabelle
- creare una pagina Web per ottenere l'elenco delle città di una Regione prefissata
- creare una pagina Web per ottenere le Regioni aventi un numero di abitanti superiore a una cifra prefissata
- creare una pagina Web per ottenere l'elenco delle città capoluogo di Regione, aventi superficie superiore a un valore prefissato.

36 Costruire un database con le informazioni dei dipendenti di una grande azienda e dei dipartimenti dove lavorano:

Dipartimenti (ID, Descrizione)
Dipendenti (Matricola, Cognome, Nome, Funzione, *CodiceDipartimento*)

ID e *Matricola* sono le chiavi primarie, *CodiceDipartimento* è la chiave esterna.

- creare le pagine Web per inserire i dati nelle tabelle
- creare una pagina Web per ottenere l'elenco di tutti i dipendenti
- creare una pagina Web per ottenere l'elenco dei dipendenti di un dipartimento il cui codice viene fornito dall'utente tramite una casella di testo
- creare una pagina Web per ottenere l'elenco dei dipendenti di una funzione fornita tramite una casella di testo.

Focus notes

WEB APPLICATIONS

Java servlets and JSP pages are the basic components for building Web applications. They provide an extension for a Web server and they are used to dynamically generate Web pages.

In Web applications, a Web client interacts with a Web server by sending an HTTP request. A Web server able to manage Java servlets and JSP pages, like Tomcat, takes the request and creates an *HTTPServletRequest* object. This object is sent to the relevant servlet or JSP page that is responsible for the generation of an *HTTPServletResponse* object. This response is converted by the Web server in an HTTP response that is finally returned to the Web client.

The process for creating, deploying, and executing a Web application, consisting of a Java Servlet and using Tomcat, can be summarised as follows:
1. Create a new subclass of the class *HttpServlet*.
2. Compile the class.
3. Copy the class in the *webapps* folder.
4. Add a servlet description and references to the file *web.xml*.
5. Open the Web application URL in a Web browser.

APACHE TOMCAT

Apache Tomcat is an open source software implementation of the Java servlet and Java Server Pages technologies. It includes a Web Server and a Servlet Engine.

The official Web site can be found at *http://tomcat.apache.org*. Installing Tomcat under Windows can be done easily by using the Windows installer. In a standard installation, Tomcat creates the following main directories: *bin*, *conf*, *lib*, *logs*, *webapps* and *work*.

Web applications are deployed in the *webapps* folder. Every application has a configuration file called *web.xml* located in the *WEB-INF* subfolder. This is an XML file describing all the servlets that constitute the Web application.

Focus notes

Glossary

browser
A client application displaying Web pages.

client application
A program that uses a service from the server.

client/server paradigm
A mix of resources with at least one client and one server.

deploy
The process of getting a software application up and running.

LAN
A computer network with a small range.

port
A connection point that identifies an application on a server.

protocol
A set of common rules used to manage the communication between two entities.

resource
A hardware device or a software application.

server application
A program that runs on a powerful computer and offers a service.

WAN
A computer network covering a broad area.

Web server
A server program that receives requests from a Web browser and sends Web pages back to it.

Acronyms

ASP	Active Server Page	**JSP**	Java Server Page
C/S	Client/Server	**LAN**	Local Area Network
CPU	Central Processing Unit	**ODBC**	Open Database Connectivity
DSN	Data Source Name	**PHP**	PHP: Hypertext Preprocessor
EAR	Enterprise ARchive	**SQL**	Structured Query Language
HTML	HyperText Markup Language	**URL**	Uniform Resource Locator
HTTP	HyperText Transfer Protocol	**WAN**	Wide Area Network
IIS	Internet Information Services	**WAR**	Web application ARchive
JAR	Java ARchive	**WWW**	World Wide Web
JDBC	Java Database Connectivity	**XML**	eXtensible Markup Language

SCHEDA DI AUTOVALUTAZIONE

CONOSCENZE

- ☐ Architettura client/server
- ☐ Server Web
- ☐ Contenuti statici e dinamici per il Web
- ☐ Tecnologie per il server Web
- ☐ Servlet
- ☐ Ambiente di esecuzione delle servlet
- ☐ Compilazione ed esecuzione delle servlet
- ☐ Passaggio di parametri alla servlet
- ☐ Architetture a 2 livelli e a 3 livelli per l'accesso ai database
- ☐ Connessione al database con una servlet
- ☐ Pagine JSP
- ☐ Ambiente di esecuzione delle pagine JSP
- ☐ Passaggio di parametri alla pagina JSP
- ☐ Passaggio di parametri tramite l'indirizzo URL
- ☐ Accesso ai database con JSP
- ☐ Operazioni di manipolazione sul database con pagine JSP
- ☐ Interrogazioni al database con pagine JSP

ABILITÀ

- ☐ Installare e avviare un server Web
- ☐ Installare e avviare un ambiente di esecuzione delle servlet
- ☐ Compilare ed eseguire le servlet
- ☐ Scrivere pagine Web che mandano parametri a una servlet
- ☐ Realizzare servlet per effettuare interrogazioni al database
- ☐ Realizzare servlet per effettuare manipolazioni sul database
- ☐ Scrivere pagine JSP per generare pagine Web statiche
- ☐ Scrivere pagine Web che mandano parametri a una pagina JSP
- ☐ Scrivere l'indirizzo URL per passare i parametri a una pagina JSP
- ☐ Effettuare operazioni di inserimento, modifica o cancellazione sul database con pagine JSP
- ☐ Effettuare interrogazioni al database con pagine JSP

SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 539

parte quarta

Applicazioni Web

9 Applicazioni per l'informatica mobile

OBIETTIVI DI APPRENDIMENTO

In questo capitolo imparerai a progettare applicazioni per dispositivi mobili, quali *smartphone* e *tablet*, basati sul sistema operativo Android.

Sarai in grado di utilizzare l'ambiente di sviluppo delle applicazioni per realizzare progetti di informatica mobile.

Imparerai a collaudare i progetti con un emulatore di dispositivi e conoscerai i passi per pubblicare le applicazioni sullo *store* per Android.

1 L'informatica mobile

L'evoluzione tecnologica, per quanto riguarda l'hardware, ha prodotto strumenti di dimensioni sempre più piccole (**miniaturizzazione**), con capacità elaborative o di memorizzazione molto elevate, con costi di produzione bassi, e tempi brevi di assemblaggio.
La diffusione di **Internet** ha facilitato l'accesso alle informazioni di generi diversi e la comunicazione su scala planetaria.

L'**informatica mobile** (*mobile computing*) è il risultato dell'evoluzione tecnologica (hardware, software e reti) che ha reso disponibile l'accesso alla rete Internet da qualsiasi luogo, attraverso dispositivi di piccole dimensioni, quali telefoni cellulari e computer portatili.

I **dispositivi mobili** sono gli strumenti che integrano le capacità di elaborazione dei tradizionali computer da tavolo con le funzionalità di comunicazione dei telefoni cellulari e della rete Internet.

I dispositivi mobili sono generalmente di tipo **touch**, cioè sono dotati di uno schermo sensibile al tatto (**touchscreen**) che svolge la doppia funzione di unità di input e output. Questi dispositivi si possono distinguere in due tipologie in base alle dimensioni dello schermo.

Lo **smartphone** (o telefono intelligente) è un'evoluzione del telefono cellulare ed è caratterizzato da un *touchscreen* con dimensioni fino ai 5 pollici.
Il **tablet** può essere assimilato ad un computer portatile senza la tastiera fisica; lo schermo, sensibile al tatto, ha una dimensione superiore ai 5 pollici.

Oltre ad uno schermo sensibile al tatto, gli *smartphone* e i *tablet* sono dotati di **strumenti integrati** che caratterizzano le moderne tecnologie mobili:
- **strumenti multimediali**: le *fotocamere* sono in grado di registrare immagini e video in formato digitale, il *microfono* consente di acquisire suoni e gli *speaker* permettono la riproduzione di brani musicali;
- **interfacce di comunicazione**: la rete telefonica mobile (3G e 4G) è utilizzabile tramite le schede *SIM*, l'interfaccia *WiFi* permette la connessione con i punti di accesso alla rete Internet, le tecnologie *USB* e *Bluetooth* offrono una modalità di comunicazione standard con i sistemi esterni;
- **strumenti di geolocalizzazione**: l'antenna **GPS** (*Global Positioning System*) permette di conoscere, in breve tempo e con la precisione di qualche metro, la propria posizione.

Il *touchscreen* ha modificato la modalità di interazione tra l'utente e il dispositivo mobile. Lo schermo sensibile al tatto riconosce i movimenti del dito dell'utente e la pressione sopra di esso.

I **gesti** che possono essere eseguiti con le dita sullo schermo *touch* sono:
- **toccare**, toccando e sollevando il dito dallo schermo, per attivare i pulsanti, per selezionare delle opzioni e per digitare le lettere sulla tastiera virtuale;
- **trascinare**, toccando lo schermo e, senza sollevare il dito, muoverlo fino a raggiungere la posizione desiderata, per spostare le icone e altri elementi grafici;
- **far scorrere**, muovendo velocemente il dito sullo schermo, per eseguire lo scorrimento orizzontale o verticale delle liste o delle pagine;
- **pizzicare**, trascinando due dita sullo schermo avvicinandole o allontanandole, per ridurre o aumentare lo *zoom* delle immagini visualizzate.

Altri gesti meno frequenti sono:
- **ruotare lo schermo**, orientando il dispositivo in posizione verticale (*portrait*, modalità ritratto) oppure in orizzontale (*landscape*, modalità paesaggio);
- **toccare due volte**, toccando e sollevando il dito dallo schermo per due volte in rapida sequenza, per eseguire uno *zoom* o adattare le dimensioni delle pagine nel browser Web;
- **toccare e tenere premuto**, toccando lo schermo senza sollevare il dito, per visualizzare i menu di scelta rapida, emulando il tasto destro del mouse nei computer desktop.

2 Il sistema operativo Android e le applicazioni

I dispositivi mobili hanno bisogno del *software* per poter funzionare. In particolare è necessario un *sistema operativo* che si occupi di gestire le risorse hardware del dispositivo e, allo stesso tempo, possa offrire un insieme di applicazioni di base per l'utente.

Android è un sistema operativo per i dispositivi mobili (*smartphone* e *tablet*) sviluppato da *Google* e basato su un kernel *Linux*.

A differenza dagli altri sistemi operativi per dispositivi mobili, come **iOS** di *Apple* e **Windows Phone** di *Microsoft*, il codice sorgente di Android è **open source**. Per questo motivo, i produttori di *smartphone* e *tablet* hanno potuto creare diverse versioni personalizzate del sistema operativo, per esempio modificando l'interfaccia grafica e introducendo alcune funzionalità aggiuntive al sistema di base.
Google fornisce agli utilizzatori dei dispositivi mobili un insieme di applicazioni che si integrano con i servizi già accessibili tramite i tradizionali computer. È previsto un accesso diretto al software di ricerca di Google, un'applicazione per la posta elettronica (*Gmail*), una per le mappe e la navigazione assistita (*Maps*), una per manipolare i documenti (*Docs*), una per salvare nel **cloud**, condividere e sincronizzare i documenti (*Drive*) e una per i social network (*Google+*).

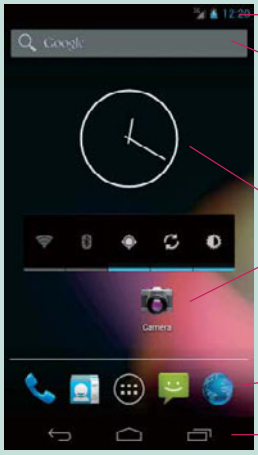
L'integrazione dei dispositivi mobili con la rete Internet ha favorito la nascita di cataloghi *on line* per la raccolta delle applicazioni. Questi luoghi virtuali, chiamati **marketplace** (o *store*), consentono la ricerca tra un vasto insieme di applicazioni, organizzate per categorie, che l'utente può scaricare e installare sul proprio dispositivo. Un esempio di market per le applicazioni Android è **Google Play**, in cui sono presenti migliaia di applicazioni sia gratuite che a pagamento.

Le *applicazioni*, dette anche **apps**, sono programmi software che estendono le funzionalità dei dispositivi mobili e possono essere scaricate dai *marketplace*.

Le applicazioni per il sistema operativo Android si possono sviluppare usando il linguaggio di programmazione **Java** e il kit **SDK** (*Software Development Kit*, kit per lo sviluppo software) fornito da *Google*.
La applicazioni Java, compilate e ottimizzate per i dispositivi mobili, sono distribuite all'interno di file con estensione **APK** (*Android Package*). Android esegue queste applicazioni all'interno di una sua specifica *Java Virtual Machine*, chiamata *Dalvik*.
Il *Software Development Kit* di Android è un insieme di librerie software, di progetti di esempio e di file di documentazione che servono al programmatore per realizzare le applicazioni per i dispositivi mobili. L'SDK include un vasto insieme di classi e di metodi, raggruppati all'interno di package funzionali. Per esempio il package *android.view* contiene le classi per realizzare e manipolare l'interfaccia grafica (pulsanti, caselle di testo), il package *android.hardware* raggruppa le funzionalità per gestire la fotocamera e gli altri sensori del dispositivo mobile, i package *android.graphics* e *android.media* sono utilizzati per l'elaborazione degli elementi multimediali, il package *android.database.sqlite* consente di gestire una base di dati privata tramite il motore *SQLite*.
Negli anni sono state rilasciate, periodicamente, versioni aggiornate dell'SDK, corrispondenti alle diverse versioni del sistema operativo. Le **versioni** di Android sono identificate da un codice numerico e da un nome, per esempio: 2.3 (*Gingerbread*), 3.0, 3.1 e 3.2 (*Honeycomb*), 4.0 (*IceCream Sandwich*), 4.1 e 4.2 (*Jelly Bean*). La molteplicità delle versioni e delle tipologie di dispositivi sono da tenere sempre presenti durante l'attività di programmazione, infatti le applicazioni di successo sono state sviluppate con l'obiettivo di comportarsi allo stesso modo sia sui *tablet* con un processore molto potente e uno schermo ampio, che su uno *smartphone* con un piccolo display o con una versione meno recente di Android.

ELEMENTI BASE DELL'INTERFACCIA ANDROID

La **schermata Home** è la videata da cui l'utente può accedere a tutte le funzionalità del dispositivo Android.



La schermata *Home* si compone di più pannelli virtuali che l'utente può far scorrere a destra e sinistra con un movimento del dito. Questi pannelli sono personalizzabili dall'utente, inserendo i collegamenti alle applicazioni oppure disponendo i **Widget**, cioè le applicazioni che sono direttamente utilizzabili dalla schermata *Home*.

I **pulsanti di navigazione** sono posizionati nella parte inferiore della schermata *Home* e consentono di gestire la navigazione tra le applicazioni e le schermate di Android.


Il pulsante **Back** permette di ritornare sulla schermata precedentemente utilizzata.


Il pulsante **Home** permette di aprire la schermata *Home*.


Il pulsante **Applicazioni recenti** apre la lista delle applicazioni che sono state utilizzate recentemente. Toccando un'applicazione la si apre, mentre facendola scorrere a destra o a sinistra la si rimuove dalla lista.

La **barra dei preferiti** è posizionata sopra i pulsanti di navigazione e contiene i collegamenti alle applicazioni più frequentemente utilizzate. Questi pulsanti restano visualizzati su ogni pannello della schermata *Home*.

Il pulsante centrale della barra dei preferiti è l'icona di **Avvio** e consente di accedere alla lista di tutte le applicazioni e di tutti i *widget* installati sul dispositivo. Dalla lista delle applicazioni, l'utente può avviare l'esecuzione di un'applicazione oppure, toccando e tenendo premuta l'icona, può trascinarla e inserire un collegamento sulla schermata *Home*.

La **barra di stato** è posizionata nella parte superiore dello schermo e ha lo scopo di visualizzare i messaggi di notifica, nella parte sinistra, e le informazioni sullo stato del dispositivo, nella parte destra. Per esempio, nella barra di stato si leggono le notifiche sugli SMS recentemente ricevuti, si controlla lo stato della batteria e l'ora del giorno. L'utente può aprire la lista di tutte le notifiche ricevute facendo scorrere, con un gesto del dito, la barra di stato verso il basso.

La **barra di ricerca** è posizionata in alto, sotto alla barra di stato ed è visibile in tutti i pannelli della schermata *Home*. Tramite la barra di ricerca, l'utente può eseguire ricerche in Internet o sul dispositivo, digitando il testo o inserendolo tramite il servizio di riconoscimento vocale.

AUTOVERIFICA

Domande da 1 a 5 pag. 496
Problemi da 1 a 2 pag. 498

MATERIALI ON LINE

1. Versioni del sistema operativo Android
2. Richiami sul linguaggio XML

3 L'ambiente di sviluppo

L'ambiente di sviluppo per la creazione di applicazioni Android si compone di tre elementi principali:

1. **Eclipse**, l'ambiente di sviluppo integrato,
2. **Android SDK** (*Software Development Kit*), l'insieme delle librerie specifiche per lo sviluppo di applicazioni Android,
3. **plug-in ADT** (*Android Developer Tools*), un plugin per *Eclipse* che facilita le operazioni di progettazione, di test e di rilascio delle *apps*.

I tre elementi precedenti possono essere installati separatamente, oppure si può utilizzare un pacchetto fornito da *Google* e chiamato **ADT Bundle** che li include tutti.
Il pacchetto *ADT Bundle*, disponibile per *Windows*, *MacOS* e *Linux*, può essere scaricato dall'indirizzo:

http://developer.android.com/sdk/

In ambiente Windows, *ADT Bundle* è un file *zip* chiamato *adt-bundle-windows-x86.zip*.
L'installazione viene eseguita scompattando l'archivio in una cartella del disco, che solitamente ha lo stesso nome del file *adt-bundle-windows-x86*. All'interno di questa cartella sono presenti due sottocartelle *eclipse* e *sdk*.

Per aprire l'ambiente di sviluppo si deve eseguire il programma **eclipse.exe** presente nella cartella *adt-bundle-windows-x86/eclipse*. All'avvio viene chiesto al programmatore di indicare la cartella in cui verranno memorizzati i progetti (*workplace*) e successivamente viene visualizzata una pagina di benvenuto.

Le funzioni di base dell'ambiente di sviluppo *Eclipse* sono già state illustrate nell'inserto dopo il Capitolo 3.

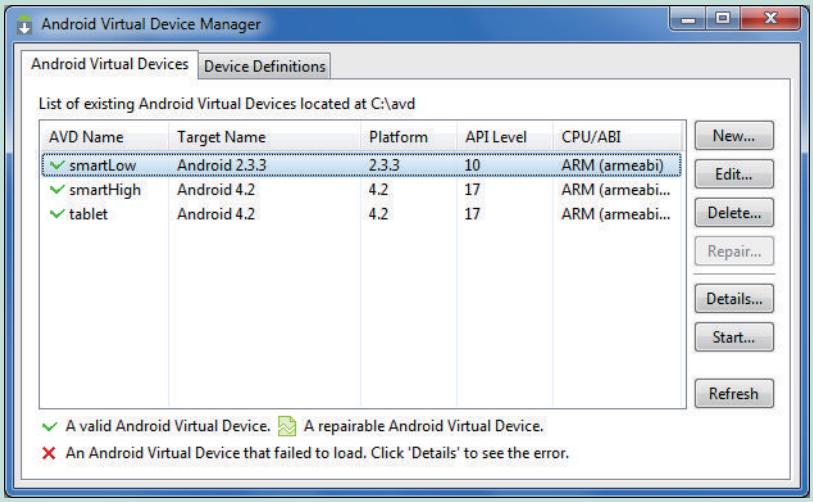
Il plugin ADT aggiunge ulteriori funzionalità ad *Eclipse*, tra cui:
- un **editor grafico**, per disegnare l'interfaccia grafica e visualizzarla in anteprima su diversi dispositivi mobili;
- la **documentazione** integrata, per visualizzare la descrizione delle classi e dei metodi dell'SDK quando si posiziona il mouse sopra una voce;
- uno strumento di **debug** avanzato, per visualizzare i messaggi di *log*, monitorare i *thread* e la memoria del dispositivo;
- gli **emulatori**, per simulare e testare le applicazione su dispositivi virtuali.

EMULATORI DI DISPOSITIVI

Un **emulatore**, chiamato anche **AVD** (*Android Virtual Device*), è un dispositivo (*device*) virtuale che permette di progettare e testare le applicazioni simulando il comportamento dei dispositivi reali.

L'ambiente di sviluppo per Android fornisce uno strumento, chiamato **AVD Manager**, per creare e configurare i dispositivi virtuali. La configurazione degli *AVD* prevede che vengano impostate le caratteristiche hardware dello *smartphone* o del *tablet* da simulare.

Per creare gli emulatori, dal menu di *Eclipse*, occorre selezionare **Window** e poi **Android Virtual Device Manager**.



Viene mostrata una schermata con i *device* virtuali presenti nel sistema: per aggiungere un *device* fare clic su **New...**

Nella finestra che si apre, occorre specificare alcune caratteristiche dell'AVD. Le opzioni principali sono:
- **Name**, il nome del *device* virtuale.
- **Device**, le caratteristiche del display del dispositivo (dimensione e risoluzione).
- **Target**, la versione di Android da simulare.

Facendo clic su **OK**, si salvano le impostazioni e si crea un nuovo *device* virtuale. In linea teorica qualsiasi tipo di terminale può essere emulato.
Si noti che il pacchetto *ADT Bundle* contiene solo la versione più recente di Android. Per utilizzare le versioni precedenti si deve eseguire l'installazione facendo clic sul menu **Window** e poi su **Android SDK Manager**.

Per collaudare le applicazioni, è opportuno creare più emulatori AVD con caratteristiche molto differenti tra loro, per esempio:
- *smartphone* con display a 3,2" a bassa risoluzione (tipica dei modelli meno recenti)
- *smartphone* con display a 4,65" ad alta risoluzione (tipica dei modelli più recenti)
- *tablet* con display a 7".

Inoltre è utile avere a disposizione dei *device* con varie versioni di Android, per esempio:
- *Android 2.x*, tipica degli smartphone meno recenti
- *Android 4.x*, tipica dei tablet e degli smartphone più recenti.

Per esempio, si possono creare tre dispositivi con le seguenti caratteristiche:

Nome	Descrizione	Versione di Android	Risoluzione	Dimensione display
smartLow	Smartphone di fascia bassa	2.3	320x480	3,2"
smartHigh	Smartphone di fascia alta	4.2	1280x720	4,65"
Tablet	Tablet	4.2	1280x800	7"

Per avviare un AVD si deve fare clic sul pulsante **Start**.
Nella finestra che compare è utile selezionare la casella *Scale display to real size* (in modo da visualizzare il dispositivo con le dimensioni reali) e successivamente fare clic sul pulsante **Launch**.

Il primo avvio, di solito, richiede qualche minuto.

Le figure mostrano rispettivamente l'emulatore del dispositivo virtuale *smartLow* e *smartHigh*. In entrambe le figure, la parte sinistra è occupata dall'interfaccia del dispositivo virtuale, mentre la parte destra contiene i pulsanti per simulare alcuni tasti del dispositivo reale.

4 Realizzare un'applicazione per Android

Il processo per realizzare un'applicazione per Android si può scomporre nei seguenti tre passi:

1. Sviluppo
2. Test e debug
3. Pubblicazione.

Per ognuna di queste fasi, l'ambiente di sviluppo *Eclipse* fornisce al programmatore gli strumenti di lavoro per ridurre gli errori e aumentare la produttività.
La fase di **sviluppo** prevede la creazione di un progetto, la scrittura del codice sorgente Java e l'impostazione dei file XML contenenti le risorse utilizzate da Android.
La fase di **test** e **debug** consiste nell'eseguire l'applicazione su uno o più dispositivi virtuali, verificando gli eventuali messaggi di *log*, con lo scopo di correggere i malfunzionamenti.
La fase di **pubblicazione** ha il compito di generare il pacchetto da distribuire agli utenti finali, solitamente l'*app* viene resa disponibile all'interno di un *marketplace*.

PROGETTO 1

Creare un'app per visualizzare l'ora corrente.

L'applicazione ha il seguente comportamento: quando l'utente la esegue, si apre una maschera che mostra, nel centro dello schermo, l'ora corrente letta dal dispositivo.

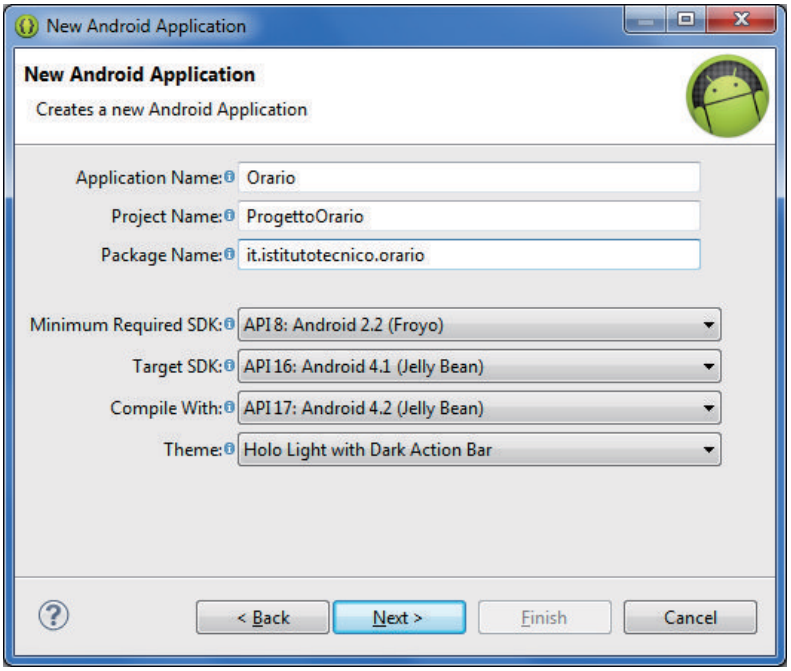
Per realizzare l'applicazione seguiremo le tre fasi elencate precedentemente.

1. *Sviluppo*

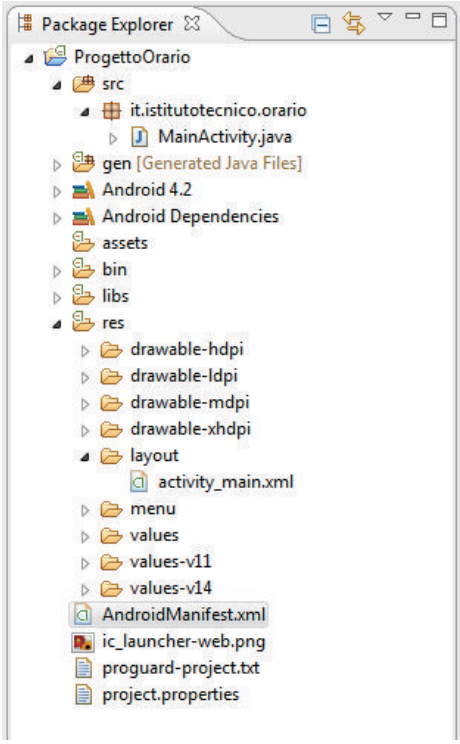
Per creare un nuovo progetto Android, dalla finestra di *Eclipse*, facciamo clic su **File**, **New**, **Project** e, nella finestra di dialogo che si apre, selezioniamo l'opzione **Android Application Project**, poi facciamo clic sul pulsante **Next**.
Viene aperta una finestra di dialogo nella quale occorre inserire alcune caratteristiche del progetto, in particolare:

- il nome dell'applicazione, visibile all'utente finale;
- il nome del progetto, usato per riconoscere l'applicazione in *Eclipse*;
- il nome del *package*: serve per identificare univocamente l'applicazione su tutti i dispositivi Android;
- le versioni dell'SDK: la versione minima richiesta per eseguire il programma e quella usata per la compilazione;
- lo stile grafico (*theme*): specifica le caratteristiche grafiche standard.

Inseriamo *Orario* come nome dell'applicazione e *ProgettoOrario* come nome del progetto. Si noti che, come nome del *package*, solitamente si usa il nome di dominio, scritto in ordine inverso, dell'organizzazione o della società che sviluppa l'applicazione. Per esempio il nome *it.istitutotecnico.orario* indica univocamente l'applicazione *orario* sviluppata da *istitutotecnico*, che è un'organizzazione localizzata in Italia (prefisso *it*).



Facciamo clic su **Next** per proseguire. Nella schermata successiva viene data la possibilità di impostare un'icona per l'applicazione. Lasciamo le impostazioni predefinite e continuiamo con **Next**. Poi, nella schermata *Create Activity*, selezioniamo la voce *BlankActivity* e infine **Next**. L'ultima schermata consente di impostare il nome della classe Java e il nome del file XML che descrivono rispettivamente i comportamenti e l'interfaccia grafica della maschera principale dell'applicazione (*Activity*). Lasciamo anche qui le impostazioni predefinite e facciamo clic su **Finish**.



In *Eclipse*, nel riquadro **Package Explorer** a sinistra, viene visualizzata la struttura del progetto appena creato, formata da un insieme di cartelle e di file generati automaticamente.

I file principali del progetto sono:

- **AndroidManifest.xml**: descrive le caratteristiche principali dell'applicazione ed elenca le componenti;
- **activity_main.xml**, nella cartella **res/layout**: descrive l'interfaccia grafica dell'applicazione.
- **MainActivity.java**, nella cartella **src/**: contiene il codice sorgente che descrive il comportamento dell'applicazione.

Facendo doppio clic sul file *AndroidManifest.xml*, nel riquadro centrale di *Eclipse* si apre una finestra di modifica che, in modo grafico, aiuta lo sviluppatore nella compilazione delle proprietà dell'applicazione. In questa finestra si può, per esempio, impostare la versione dell'app espressa con due campi: un numero (*Version code*), usato internamente e aggiornato dallo sviluppatore ad ogni nuova pubblicazione, e un nome (*Version name*) visibile all'utente finale.
Il contenuto completo del file è visibile selezionando il pannello *AndroidManifest.xml*, AndroidManifest.xml ed è riportato di seguito.

Manifest (*AndroidManifest.xml*)

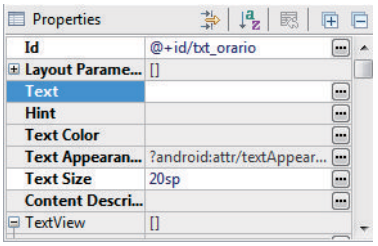
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="it.istitutotecnico.orario"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="16" />
```

```
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="it.istitutotecnico.orario.MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Si noti che nel tag *manifest* sono riportate le informazioni sulla versione dell'applicazione, nel tag *uses-sdk* le versioni dell'SDK e nel tag *activity* il nome della classe contenente il codice sorgente.

Facendo doppio clic sul file *activity_main.xml*, nel riquadro centrale di *Eclipse* si apre un'area di modifica che, in modo grafico, aiuta lo sviluppatore nel disegno dell'interfaccia grafica dell'applicazione (**Layout**). La modalità di costruzione dell'interfaccia grafica in Android è simile a quanto è stato descritto nell'inserto dopo il Capitolo 5 per la costruzione delle interfacce grafiche con *NetBeans*.

Selezionando il pannello *Graphical Layout* Graphical Layout si apre l'area in cui si può costruire in modo visuale l'interfaccia utente. La finestra **Palette** contiene tutte le componenti standard dell'interfaccia utente (come caselle di testo, pulsanti, etichette, ecc.) che possono essere trascinate sul pannello centrale per comporre la schermata visualizzata dall'utente finale. La finestra **Properties** mostra un insieme di caratteristiche degli oggetti grafici che possono essere modificate, come la posizione, la dimensione e il tipo di font.



La schermata, automaticamente creata con il progetto, contiene un'etichetta di testo (**TextView**), che useremo per inserire l'orario corrente da visualizzare nell'applicazione. Facciamo clic sull'etichetta e modifichiamo le seguenti proprietà, nella finestra *Properties*:

- nella proprietà **Id** inseriamo il valore **@+id/txt_orario**, per identificare la componente grafica,
- nella proprietà **Text** cancelliamo il contenuto, in quanto sarà aggiornato successivamente con l'ora corrente,
- nella proprietà **Text Size** impostiamo la dimensione del font con il valore **20sp**.

La dimensione è espressa in **sp** (*scale-independent pixels*), che indica una misura scalabile secondo la preferenza del font dell'utente.
La descrizione dell'interfaccia grafica viene memorizzata in un file XML, il cui contenuto è visibile selezionando il pannello *activity_main.xml* activity_main.xml ed è riportato di seguito.

Layout (*res/layout/activity_main.xml*)

```
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity" >

  <TextView
     android:id="@+id/txt_orario"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_centerHorizontal="true"
     android:layout_centerVertical="true"
     android:textSize="20sp" />

  </RelativeLayout>
```

Facendo doppio clic sul file *MainActivity.java*, nel riquadro centrale di *Eclipse* si apre il codice sorgente Java relativo alla maschera principale dell'applicazione. Il metodo **onCreate** viene eseguito quando l'utente avvia l'applicazione, quindi deve contenere le istruzioni di inizializzazione dell'interfaccia grafica.
L'applicazione che stiamo realizzando deve leggere l'ora attuale del dispositivo e la deve inserire nell'etichetta *txt_orario*, definita nel file di *layout*.
L'ora corrente viene letta utilizzando la classe **Date** e viene inserita in una stringa usando la formattazione definita con la classe **DateFormat** nel seguente modo:

```
Date date = new Date();
DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
String strOrario = dateFormat.format(date);
```

L'etichetta è stata identificata nel file del layout con il nome *txt_orario* e può essere utilizzata nel codice Java cercando il riferimento con il metodo **findViewById**:

```
TextView txtOrario = (TextView) findViewById(R.id.txt_orario);
```

Avendo il riferimento alla componente grafica *txtOrario*, si usa il metodo **setText** per inserire e visualizzare l'orario nell'etichetta.

```
txtOrario.setText(strOrario);
```

Il codice completo della classe *MainActivity*, in cui le precedenti istruzioni sono state collocate all'interno del metodo *onCreate*, è riportato di seguito.

Activity (*MainActivity.java*)

```
package it.istitutotecnico.orario;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
```

```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.TextView;

public class MainActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_orario);

    Date date = new Date();
    DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
    String strOrario = dateFormat.format(date);

    TextView txtOrario = (TextView) findViewById(R.id.txt_orario);
    txtOrario.setText(strOrario);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.activity_main, menu);
    return true;
  }
}
```

Si noti che, all'inizio del codice, sono stati inseriti i comandi di *import* per tutte le classi utilizzate. Nel caso non fossero state riportate tutte le classi, *Eclipse* segnala l'errore con una sottolineatura rossa sulla classe non riconosciuta.

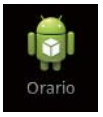
Per completare la fase di sviluppo, si deve salvare tutto il lavoro facendo clic sul menu **File** e poi su **Save All**, oppure facendo clic sulla relativa icona nella Barra degli strumenti.

2. *Test e Debug*

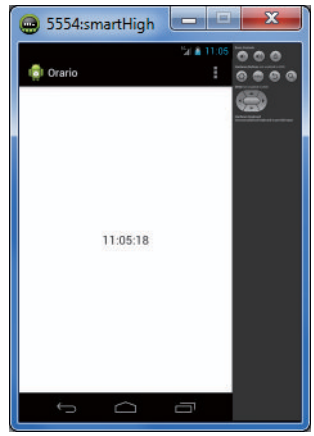
Per testare l'applicazione, eseguiamo l'emulatore facendo clic sul menu **Run** e poi sulla voce **Run**, oppure usando la combinazione di tasti **Ctrl + F11**. Se sono presenti più emulatori, si può impostare quello preferito dal menu *Run* facendo clic su **Run Configurations**. All'interno del pannello **Target**, selezioniamo *smartHigh* come ADV preferito, e infine facciamo clic su **Apply** per confermare la scelta.

Quando viene avviata l'esecuzione sull'emulatore, *Eclipse* opera nel seguente modo:
- genera il pacchetto compilato *ProgettoOrario.apk*,
- installa il pacchetto nel dispositivo virtuale,
- avvia l'applicazione.

Sull'emulatore, la nuova applicazione viene aggiunta alla lista delle applicazioni e si identifica con l'icona e il nome che sono stati impostati durante la creazione guidata del progetto, all'inizio della fase di sviluppo.



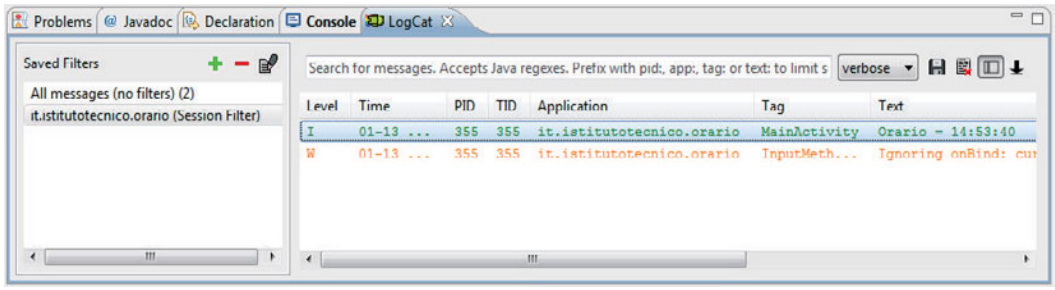
Durante la fase di test, è opportuno effettuare il collaudo dell'applicazione su diversi emulatori in modo da garantire il corretto funzionamento sul maggior numero possibile di dispositivi.



Android dispone di un sistema per la raccolta e la visualizzazione di tutti i messaggi di errore chiamato **Logcat**.
Logcat è una finestra di *Eclipse* che può essere aperta facendo clic sul menu **Window** e poi su **Show View**. Nella lista che si apre si deve scegliere la voce *Logcat* oppure, se non è presente, si deve aprire la voce **Other...** e cercare il *Logcat* nel gruppo delle finestre personalizzate di *Android*.
Oltre ai messaggi relativi agli errori di *run-time*, lo sviluppatore può inserire nel codice Java i punti di controllo, prevedendo la scrittura dei propri messaggi di *log* nella finestra *Logcat*.
I metodi da usare per stampare questi messaggi sono contenuti nella classe **Log**. Per esempio, per stampare un messaggio di *log* che mostra l'orario calcolato, si deve usare il comando:

```
Log.i("MainActivity", "Orario = " + strOrario);
```

I due parametri del metodo **Log.i** vengono visualizzati rispettivamente nella colonna **Tag** e nella colonna **Text** della finestra di *Logcat*.



La classe *Log*, a seconda del tipo di messaggio che si vuole stampare, suggerisce l'utilizzo di uno tra i seguenti cinque metodi:
- **Log.v**, per i messaggi *verbose*, cioè di stile descrittivo;
- **Log.d**, per i messaggi specifici di *debug*;
- **Log.i**, per i messaggi informativi (*information*);
- **Log.w**, per i messaggi di avvertimento (*warning*), quando ci sono malfunzionamenti;
- **Log.e**, per i messaggi di errore (*error*).

3. *Pubblicazione*

Dopo l'esecuzione dei test su diversi emulatori e dopo la correzione degli eventuali malfunzionamenti riscontrati, possiamo predisporre il **pacchetto APK** per distribuire l'applicazione sui dispositivi Android.

La distribuzione della applicazioni sui *marketplace online* richiede che il pacchetto venga firmato con un certificato digitale. Questa modalità di pubblicazione sarà illustrata nei paragrafi successivi.

Il pacchetto APK può essere confezionato anche senza una firma digitale e il file generato può essere installato sugli *smartphone* e *tablet*. Il pacchetto così creato prende il nome di **Unsigned Application Package** e viene creato durante la fase di sviluppo dal sistema di compilazione automatica.
Se il *workspace* di *Eclipse* è posizionato nella cartella *C:\android*, il file *ProgettoOrario.apk* è memorizzato nella sottocartella **bin** all'interno della cartella di progetto:

C:\android\ProgettoOrario\bin\

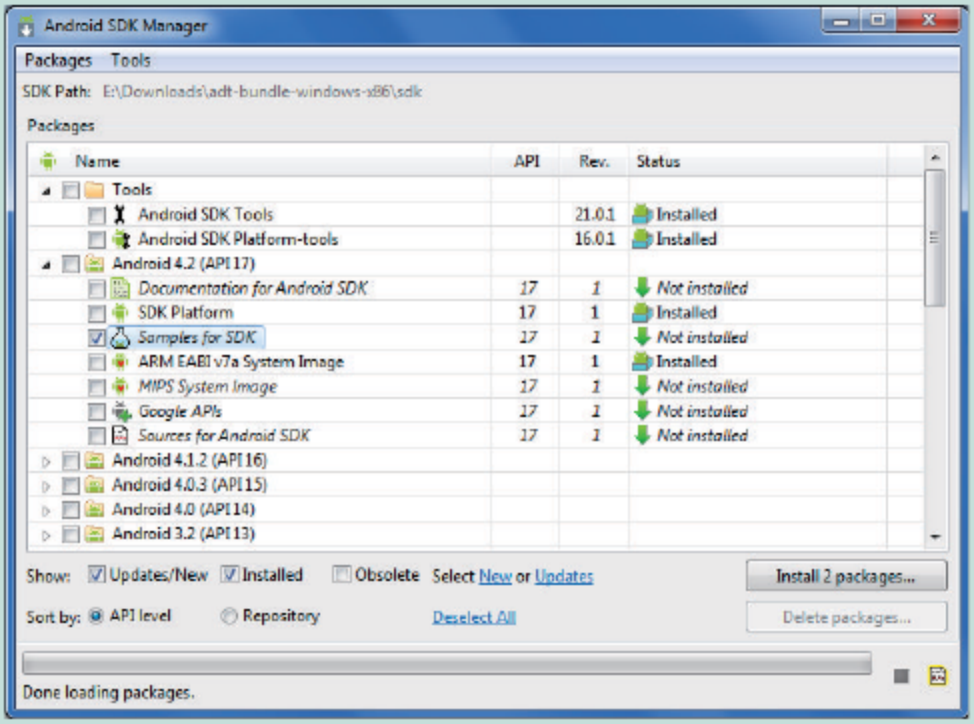
In alternativa, per creare direttamente un file APK non firmato, si deve fare clic con il tasto destro sul nome del progetto nel riquadro *Package Explorer*. Nel menu a tendina, si deve scegliere la voce **Android Tools** e fare clic su **Export Unsigned Application Package...**. Nella finestra successiva occorre inserire il nome del file e fare clic sul pulsante **Salva**.

Il file APK, corrispondente all'applicazione Android, può essere copiato sulla scheda di memoria del dispositivo mobile usando un collegamento USB oppure può essere inviato al dispositivo come allegato ad un messaggio di posta elettronica.

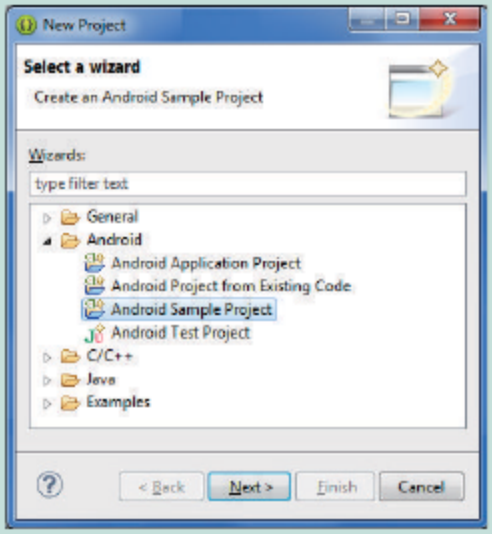
Quando apriamo il messaggio di e-mail sul dispositivo mobile, possiamo installare l'applicazione facendo clic sul pulsante dell'allegato *ProgettoOrario.apk*. Al termine dell'installazione, l'icona dell'applicazione viene aggiunta alla lista delle applicazioni.

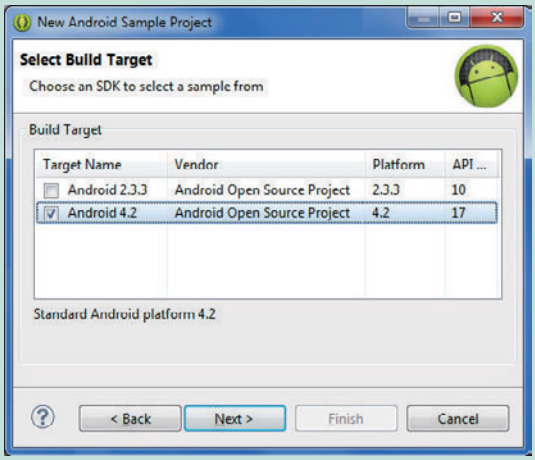
LE APPLICAZIONI DI ESEMPIO IN SDK

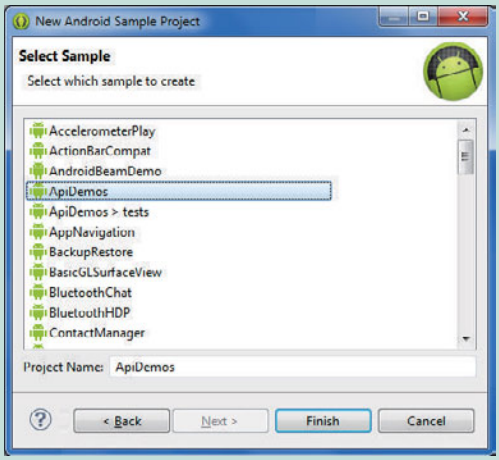
Android SDK viene distribuito con un insieme di applicazioni di esempio che risultano utili per imparare il corretto sviluppo delle *apps*. Nel seguito viene spiegato come aprire un'applicazione di esempio all'interno dell'ambiente di sviluppo.
Per prima cosa verifichiamo che gli esempi siano stati correttamente installati con l'SDK. Selezionando, dal menu **Window** di *Eclipse*, la voce **Android SDK Manager** si apre la finestra con l'elenco dei pacchetti installati. Assicuriamoci che la riga **Samples for SDK**, all'interno dell'ultima versione di Android, sia stata installata. In caso contrario eseguiamo l'installazione, selezionando la riga e facendo clic sul pulsante **Install**.



Per aprire un'applicazione di esempio in *Eclipse*, facciamo clic sul menu **File**, poi su **New** e infine sulla voce **Project**. Nella finestra di dialogo che si apre scegliamo, all'interno del gruppo **Android**, la voce **Android Sample Project**.



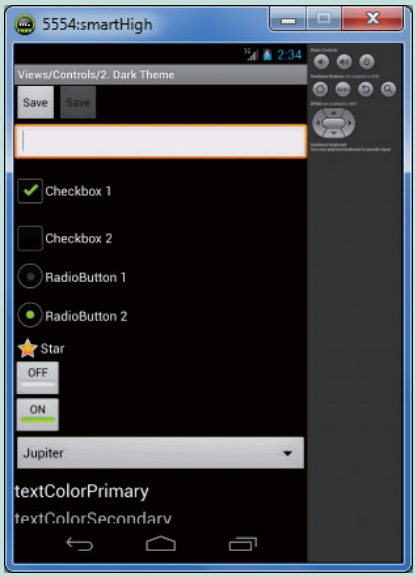




Dopo aver fatto clic sul pulsante **Next**, scegliamo la versione **Target** dell'SDK a cui gli esempi fanno riferimento. Facendo nuovamente clic su **Next**, viene visualizzato l'elenco delle applicazioni di esempio. Tra queste selezioniamo *ApiDemos* e facciamo clic sul pulsante **Finish**.

L'applicazione di esempio viene creata come una nuova cartella all'interno del *Workspace* di *Eclipse*. Nel riquadro *Package Explorer* si può navigare all'interno delle cartelle che compongono il progetto e si può analizzare, ed eventualmente modificare, il contenuto dei file sorgente.

Per testare l'applicazione, eseguiamo l'emulatore facendo clic sul pulsante *Run*. Se sono presenti più emulatori, si può impostare quello preferito dal menu *Run* facendo clic su **Run Configurations**. All'interno del pannello **Target**, selezioniamo *smartHigh* come AVD preferito, e infine facciamo clic su **Apply** per confermare la scelta.
Si noti che, selezionando l'opzione *Always prompt to pick device*, il dispositivo AVD può essere scelto dal programmatore ogni volta che viene avviata l'esecuzione di test.



L'applicazione *ApiDemos*, avviata nell'emulatore, mostra qual è il comportamento delle varie componenti grafiche utilizzabili per la progettazione delle interfacce grafiche in Android. Alcune di queste componenti grafiche saranno presentante nei paragrafi successivi.

AUTOVERIFICA

Domande da 6 a 12 pag. 496-497
Problemi da 3 a 14 pag. 498-499

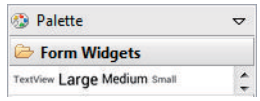
MATERIALI ON LINE

3. Aggiornare ed estendere l'Android SDK

5 Etichette, caselle di testo e pulsanti di comando

L'**etichetta** viene creata utilizzando la classe **TextView** (*android.widget.TextView*) ed è solitamente utilizzata per inserire messaggi di testo, o descrizioni di caselle di testo, nelle videate dell'applicazione. Per aggiungere un'etichetta basta trascinare l'oggetto *TextView*, presente nel gruppo *Form Widget* della finestra *Palette* di *Eclipse*, sulla finestra *Graphical Layout*.

Nello stesso gruppo della *Palette*, oltre all'oggetto base *TextView*, sono presenti altri tre oggetti che definiscono le etichette con una diversa dimensione del font: grande (**Large**), media (**Medium**) e piccola (**Small**).

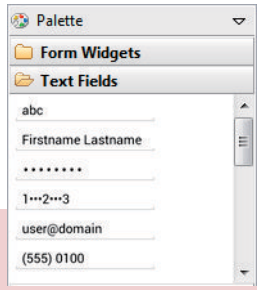


Il testo da visualizzare può essere modificato direttamente nella finestra *Properties*, impostando l'attributo **Text**. Come tutte le altre componenti grafiche, l'etichetta viene registrata in un file XML all'interno della cartella **res/layout/**, con il seguente tag:

```
<TextView android:id           ="@+id/txtDesc"
          android:layout_width ="wrap_content"
          android:layout_height="wrap_content"
          android:text         ="Descrizione" />
```

La proprietà **id** stabilisce un nome univoco (*txtDesc*) per l'etichetta e deve essere indicato con la codifica *@+id/txtDesc*. Questo nome potrà essere utilizzato successivamente all'interno del codice Java come riferimento all'oggetto grafico.

La casella di testo **EditText** (*android.widget.EditText*) è uno degli oggetti grafici più utilizzati per permettere all'utente di inserire i dati. Per aggiungere una casella di testo si deve trascinare uno degli oggetti presenti nel gruppo *Text Fields* della finestra *Palette* di *Eclipse*.



Il corrispondente codice XML è il seguente:

```
<EditText android:id="@+id/editNominativo "
          android:layout_width="match_parent"
          android:layout_height="wrap_content" />
```

Le proprietà **layout_width** e **layout_height** stabiliscono le dimensioni in larghezza e in altezza dell'oggetto grafico. Il valore **wrap_content** indica che le dimensioni devono adattarsi al contenuto della casella di testo, mentre il valore **match_parent** stabilisce di espandere le dimensioni fino a raggiungere quelle del contenitore in cui la casella di testo è inserita.

Nello stesso gruppo della *Palette*, oltre all'oggetto base *EditText,* sono presenti altre caselle di testo che gestiscono e facilitano l'inserimento di valori alfanumerici come password, indirizzi email, numeri di telefono e dati anagrafici.

I **pulsanti di comando** (o *bottoni*) vengono creati utilizzando la classe **Button** (*android.widget.Button*) e sono utilizzati per attivare l'esecuzione di particolari azioni. Per aggiungere un pulsante in una schermata dell'applicazione occorre trascinare l'oggetto *Button* presente nel gruppo *Form Widget* della finestra *Palette* di *Eclipse*.

Il corrispondente codice XML è il seguente:

```
<Button android:id              ="@+id/btnCalcola"
        android:layout_width    ="wrap_content"
        android:layout_height   ="wrap_content"
        android:text            ="Calcola"
        android:onClick         ="calcolaTotale" />
```

Le proprietà **onClick** è usata per indicare il gestore dell'evento per il pulsante. In particolare il valore è il nome del metodo che viene eseguito in risposta all'evento clic generato dal pulsante. Nel codice Java si deve inserire un metodo pubblico, contenente le azioni da eseguire in risposta all'evento, con la seguente struttura:

```
public void calcolaTotale (View view) {
    // azioni da eseguire
}
```

La proprietà *Text* delle etichette e dei pulsanti di comando viene utilizzata per specificare il testo mostrato nell'interfaccia grafica. Un corretto stile di programmazione in Android prevede che nella proprietà *Text* non venga indicato direttamente il testo, ma il riferimento alla risorsa che contiene il testo da visualizzare.

Per esempio, la proprietà *Text* della componente *Button*, dichiarata precedentemente con la notazione:

```
android:text = "Calcola"
```

dovrebbe essere sostituita con la seguente:

```
android:text = "@string/str_calcola"
```

La notazione **@string/str_calcola** indica il riferimento ad una risorsa di tipo testo (*@string*) che si chiama *str_calcola*.

Il file **strings.xml**, contenuto nella cartella **res/values/**, raggruppa tutte le risorse di tipo testo che possono essere utilizzate dall'applicazione.

Il valore della stringa *str_calcola* viene dichiarato nel file *strings.xml* utilizzando il seguente tag:

```
<string name="str_calcola">Calcola</string>
```

Un esempio di file *strings.xml* che, oltre alla stringa precedente, contiene le stringhe *app_name* e *str_descrizione*, è il seguente:

```
<resources>
    <string name="app_name">Calcolatrice</string>
    <string name="str_calcola">Calcola</string>
    <string name="str_descrizione">Descrizione</string>
</resources>
```

Si osservi che questa modalità di rappresentazione delle risorse consente di mantenere separato il layout della pagina, salvato nella cartella *res/layout/*, dal contenuto memorizzato nella cartella *res/values/*.

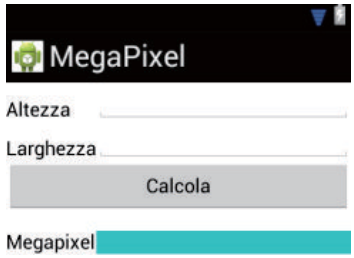
PROGETTO 2

Calcolare il numero di Megapixel di una fotocamera, date le dimensioni di una immagine prodotta.

Si vuole creare un'applicazione che permetta all'utente di inserire le dimensioni di un'immagine, specificando i valori di altezza e larghezza espressi in pixel. Facendo clic su un pulsante di comando, l'applicazione calcola e visualizza il numero di Megapixel della fotocamera che ha scattato l'immagine.

Si ricorda che il **Megapixel** (*Mp*) è un'unità di misura corrispondente a un milione di pixel ed è utilizzata per definire la risoluzione di un'immagine prodotta da una fotocamera.

L'interfaccia grafica è composta da due caselle di testo (*editAltezza* e *editLarghezza*) per leggere i dati dall'utente, da un pulsante di comando (*btnCalcola*) per avviare il calcolo e da un'etichetta (*txtRisultato*) per visualizzare il risultato del calcolo.



La seguente tabella riassume tutte le classi grafiche che vengono aggiunte al file di layout *res/layout/activity_main.xml* e le due principali proprietà, *Id* e *Text*. Nella colonna *Nome oggetto* viene indicato il nome che verrà utilizzato successivamente nel codice Java.

Classe	Id	Text	Nome oggetto
TextView	@+id/txtAltezza	@string/str_altezza	
TextView	@+id/txtLarghezza	@string/str_larghezza	
EditText	@+id/editAltezza		editAltezza
EditText	@+id/editLarghezza		editLarghezza
Button	@+id/btnCalcola	@string/str_calcola	
TextView	@+id/txtMegapixel	@string/str_megapixel	
TextView	@+id/txtRisultato		txtRisultato

In aggiunta alle precedenti, impostiamo nell'etichetta *txtRisultato* le seguenti proprietà:
- **Gravity**: *center_horizontal*, per allineare il testo al centro;
- **Background**: *#00CCCC*, per evidenziare lo sfondo con il colore azzurro.

Nelle caselle di testo *editAltezza* e *editLarghezza* indichiamo come proprietà **Input Type** il valore *number*, per consentire all'utente di inserire solo cifre numeriche.

Per il pulsante *btnCalcola* specifichiamo, nella proprietà **On Click**, il metodo *calcolaRisultato* come gestore dell'evento.

Risorse (*res/values/strings.xml*)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">MegaPixel</string>
    <string name="menu_settings">Settings</string>
    <string name="str_altezza">Altezza</string>
    <string name="str_larghezza">Larghezza</string>
    <string name="str_calcola">Calcola</string>
    <string name="str_megapixel">Megapixel</string>
</resources>
```

Layout (*res/layout/activity_main.xml*)

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <EditText
        android:id="@+id/editLarghezza"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/editAltezza"
        android:layout_toRightOf="@+id/txtLarghezza"
        android:inputType="number" />

    <EditText
        android:id="@+id/editAltezza"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/txtLarghezza"
        android:inputType="number" />

    <Button
        android:id="@+id/btnCalcola"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/editLarghezza"
        android:onClick="calcolaRisultato"
        android:text="@string/str_calcola" />
```

```
    <TextView
        android:id="@+id/txtRisultato"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/txtMegapixel"
        android:layout_alignLeft="@+id/editLarghezza"
        android:layout_alignRight="@+id/editLarghezza"
        android:layout_alignTop="@+id/txtMegapixel"
        android:background="#00CCCC"
        android:gravity="center_horizontal"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/txtMegapixel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/btnCalcola"
        android:layout_marginTop="17dp"
        android:text="@string/str_megapixel"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/txtAltezza"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/editAltezza"
        android:layout_alignParentLeft="true"
        android:text="@string/str_altezza"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/txtLarghezza"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/btnCalcola"
        android:layout_alignParentLeft="true"
        android:text="@string/str_larghezza"
        android:textAppearance="?android:attr/textAppearanceMedium" />

</RelativeLayout>
```

Nel codice Java, i riferimenti agli elementi grafici vengono dichiarati come attributi privati della classe *MainActivity*.

```
private EditText editAltezza, editLarghezza;
private TextView txtRisultato;
```

All'interno del metodo *onCreate*, viene eseguito il metodo **setContentView** per caricare l'interfaccia grafica descritta nel file di layout *activity_main.xml*. Il riferimento al layout è indicato con il nome del file preceduto dalla notazione **R.layout**.

```
setContentView(R.layout.activity_main);
```

Successivamente vengono recuperati i collegamenti con le componenti grafiche dichiarate nel layout.

```
editAltezza    = (EditText) findViewById(R.id.editAltezza);
editLarghezza  = (EditText) findViewById(R.id.editLarghezza);
txtRisultato   = (TextView) findViewById(R.id.txtRisultato);
```

Il metodo *findViewById* cerca l'oggetto grafico il cui nome, indicato nel parametro, è preceduto dalla notazione **R.id**.

Nel metodo *calcolaRisultato*, i valori inseriti dall'utente vengono letti utilizzando i metodi **getText().toString()** sull'oggetto grafico e vengono convertiti in valori numerici.

```
altezza   = Integer.parseInt(editAltezza.getText().toString());
larghezza = Integer.parseInt(editLarghezza.getText().toString());
```

Il risultato viene visualizzato nell'etichetta *txtRisultato* utilizzando il metodo **setText**.

```
txtRisultato.setText("" + mp + " Mp");
```

Il codice completo della classe Java che implementa l'applicazione è mostrato di seguito.

Activity (*MainActivity.java*)

```
package it.istitutotecnico.megapixel;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity {

  private EditText editAltezza, editLarghezza;
  private TextView txtRisultato;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // crea il collegamento con le componenti grafiche
    editAltezza   = (EditText) findViewById(R.id.editAltezza);
    editLarghezza = (EditText) findViewById(R.id.editLarghezza);
    txtRisultato  = (TextView) findViewById(R.id.txtRisultato);
  }
```

```
  public void calcolaRisultato(View view) {
     int altezza, larghezza, mp;

     altezza   = Integer.parseInt(editAltezza.getText().toString());
     larghezza = Integer.parseInt(editLarghezza.getText().toString());

     Log.i("MainActivity",
           "Altezza= " + altezza + " Larghezza= " + larghezza);

     mp = (altezza * larghezza) / 1000000;

     txtRisultato.setText("" + mp + " Mp");
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
     getMenuInflater().inflate(R.menu.activity_main, menu);
     return true;
  }
}
```

PROGETTO 3

Modificare il progetto precedente per il calcolo dei Megapixel, aggiungendo un controllo che avvisa l'utente quando mancano alcuni dati.

Se l'utente fa clic sul pulsante *Calcola* prima di aver inserito i valori dell'altezza e della larghezza dell'immagine, viene visualizzato un messaggio di testo che resta visibile per qualche secondo sul dispositivo mobile.

In Android, la classe **Toast** (*android.widget.Toast*) implementa l'oggetto grafico che visualizza un messaggio di testo, resta visibile per un certo periodo di tempo, e poi automaticamente scompare.

La creazione dell'oggetto viene eseguita con il metodo **Toast.makeText**, il cui primo parametro è il contesto, il secondo è il messaggio da visualizzare e il terzo è la durata. I valori standard per la durata sono indicati dai due attributi statici *Toast.LENGTH_LONG* e *Toast.LENGTH_SHORT*.

```
toast = Toast.makeText(getApplicationContext(),
         "Altezza sbagliata", Toast.LENGTH_LONG);
```

Per visualizzare il messaggio si deve usare il metodo **show**.

```
toast.show();
```

Il codice del metodo *calcolaRisultato* deve essere modificato come segue.

Gestione degli eventi

```
public void calcolaRisultato(View view) {
  int altezza, larghezza, mp;
  Toast toast;

  try {
    altezza = Integer.parseInt(editAltezza.getText().toString());
  }
  catch(NumberFormatException exc) {
    Log.e("MainActivity", "Errore conversione altezza");
    toast = Toast.makeText(getApplicationContext(),
              "Altezza sbagliata", Toast.LENGTH_LONG);
    toast.show();
    return;
  }

  try {
    larghezza = Integer.parseInt(editLarghezza.getText().toString());
  }
  catch(NumberFormatException exc) {
    Log.e("MainActivity", "Errore conversione larghezza");
    toast = Toast.makeText(getApplicationContext(),
              "Larghezza sbagliata", Toast.LENGTH_LONG);
    toast.show();
    return;
  }

  Log.i("MainActivity",
        "Altezza= " + altezza + " Larghezza= " + larghezza);

  mp = (altezza * larghezza) / 1000000;

  txtRisultato.setText("" + mp + " Mp");
}
```

Prova di esecuzione
Se il valore nella casella *editAltezza* non può essere correttamente trasformato in un numero, appare un messaggio che comunica il tipo di anomalia. In caso contrario si procede normalmente.

LOCALIZZAZIONE DEL SOFTWARE

La **localizzazione** (in inglese *localisation*) è il processo che consente di adattare l'applicazione software all'utilizzo in altre nazioni e, nella maggior parte delle situazioni, consiste nel fornire le traduzioni del testo nella lingua richiesta. In casi più particolari, la localizzazione potrebbe richiedere ulteriori adattamenti per adeguare l'applicazione alla cultura del paese in cui si vuole distribuire.
Per gestire la localizzazione, Android consente di definire più risorse all'interno della stessa applicazione. In fase di esecuzione sarà scelta la risorsa che meglio si adatta al dispositivo e all'utente.
Il file *res/values/strings.xml* rappresenta la **risorsa di default** per i messaggi di testo. Contiene i messaggi espressi nella lingua che ci si aspetta essere quella maggiormente utilizzata dagli utenti dell'applicazione.
Le risorse alternative, per i messaggi di testo in altre lingue, devono essere memorizzati all'interno di cartelle specifiche, il cui nome identifica il tipo di localizzazione.
Per esempio,
- *res/values*-**en**/*strings.xml* contiene le stringhe tradotte in lingua inglese,
- *res/values*-**fr**/*strings.xml* contiene le stringhe tradotte in lingua francese.

Se la risorsa di default (*res/value/strings.xml*) contiene i seguenti messaggi:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="str_nome">Nome</string>
    <string name="str_cognome">Cognome</string>
</resources>
```

le risorse alternative in lingua inglese (*res/value-en/strings.xml*) sono:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="str_nome">First name</string>
    <string name="str_cognome">Last name</string>
</resources>
```

6 Activity

All'interno dell'inserto dopo il Capitolo 5 abbiamo imparato che le finestre delle applicazioni Java corrispondono ad oggetti di classe *JFrame*, detti anche **contenitori**.
In Android, il contenitore corrispondente al *JFrame* è rappresentato dalla classe **Activity**.
Ogni finestra di un'applicazione Android deve essere dichiarata come una sottoclasse di *Activity*, nel seguente modo:

```
public class FinestraPrincipale extends Activity {

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  }
}
```

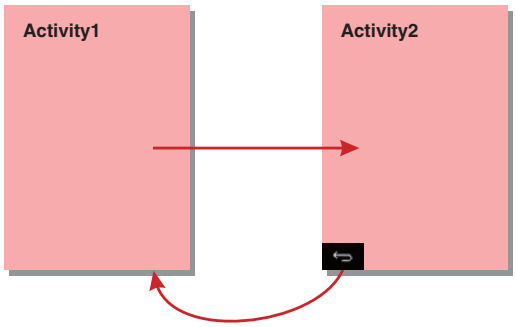
Il metodo **onCreate**, ereditato dalla sopraclasse *Activity*, deve essere ridefinito (*Override*) con l'aggiunta di tutte le istruzioni necessarie per la visualizzazione e la gestione della finestra grafica. La prima istruzione del metodo *onCreate* richiama il costruttore della sopraclasse. Successivamente viene aggiunta l'istruzione per caricare l'interfaccia grafica, definita in un file XML. Per esempio, se l'interfaccia grafica è salvata nel file *prima_videata.xml*, l'istruzione per il suo caricamento è la seguente:

```
setContentView(R.layout.prima_videata);
```

Per poter utilizzare un'*Activity*, oltre a dichiarare la sua classe, si deve inserire il nome della classe nel file *AndroidManifest.xml* dell'applicazione, utilizzando il seguente tag:

```
<activity android:label="@string/app_name"
          android:name =".FinestraPrincipale">
</activity>
```

La realizzazione di un'applicazione composta da due videate comporta la creazione di due sottoclassi di *Activity* e l'inserimento nel *Manifest* di due riferimenti tramite il tag *<activity>*.



Per passare dalla videata *Activity1* alla videata *Activity2*, si devono eseguire le seguenti istruzioni:

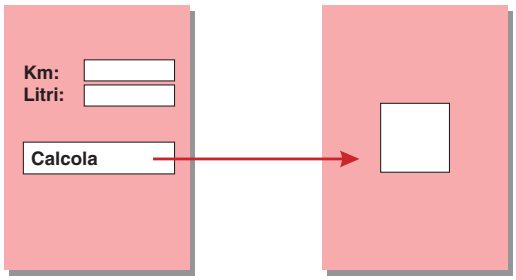
```
Intent intent = new Intent(Activity1.this, Activity2.class);
startActivity(intent);
```

Il metodo **startActivity** viene richiamato con il parametro **intent**, che stabilisce l'oggetto da cui scaturisce l'azione (*Activity1.this*) e la classe dell'*Activity* che deve essere aperta (*Activity2.class*)

PROGETTO 4

Creare un'app per calcolare il consumo medio di carburante di un veicolo.

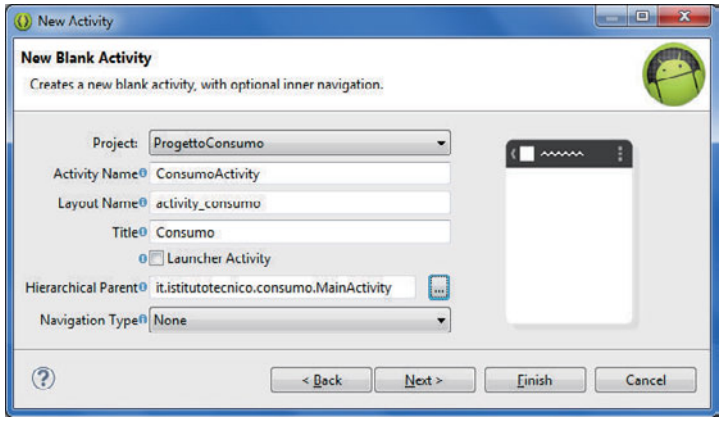
L'applicazione è composta da due *Activity*: la prima contiene due caselle di testo per la raccolta dei dati e un pulsante per l'avvio del calcolo, la seconda visualizza il risultato del calcolo.



Il consumo viene calcolato dividendo i km percorsi per i litri di carburante consumati ed è espresso in km/l. Le componenti usate per la progettazione dell'interfaccia grafica sono indicate nella seguente tabella:

Classe	Id	Text	Nome oggetto
TextView	@+id/txtKM	@string/str_km	
TextView	@+id/txtLitri	@string/str_litri	
EditText	@+id/editKM		editKM
EditText	@+id/editLitri		editLitri
Button	@+id/btnConsumo	@string/str_consumo	
TextView	@+id/txtConsumo		txtConsumo

Dopo aver creato il progetto con l'*Activity* principale, per creare la seconda *Activity* si deve fare clic sul pulsante **New** nella barra degli strumenti, scegliere **Android Activity** e fare clic su **Next**. Nella successiva finestra, selezionare **BlankActivity** e poi di nuovo **Next**. Impostiamo *ConsumoActivity* e *activity_consumo* rispettivamente come nome della classe e del layout della nuova *Activity*.



Facendo clic su **Finish** vengono generati automaticamente i file *ConsumoActivity.java* e *activity_consumo.xml*. Inoltre il file *AndroidManifest.xml* viene aggiornato con le informazioni sulla nuova *Activity*.
Quando l'utente fa clic sul pulsante di calcolo, il gestore dell'evento, definito nel metodo *calcolaConsumo* della prima *Activity*, effettua il calcolo e lo visualizza nella seconda *Activity*.

```
Intent intent = new Intent(this, ConsumoActivity.class);
intent.putExtra(PARAM_CONSUMO, consumo);
startActivity(intent);
```

Il metodo **putExtra** viene usato per aggiungere il parametro *consumo* all'oggetto *intent*. La costante *PARAM_CONSUMO* viene usata per assegnare un nome univoco al parametro e viene dichiarata con la seguente istruzione:

```
public final static String PARAM_CONSUMO = "param_consumo";
```

La lettura del parametro numerico viene eseguita con il metodo **getDoubleExtra** dalla seconda *Activity*, indicando come nome del parametro la costante pubblica *PARAM_CONSUMO* dichiarata nella classe *MainActivity*.

```
Intent intent = getIntent();
double consumo = intent.getDoubleExtra(MainActivity.PARAM_CONSUMO, 0);
```

Il codice sorgente di tutti i file che compongono il progetto è riportato di seguito.

Manifest (*AndroidManifest.xml*)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="it.istitutotecnico.consumo"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
     android:minSdkVersion="8"
     android:targetSdkVersion="16" />

  <application
     android:allowBackup="true"
     android:icon="@drawable/ic_launcher"
     android:label="@string/app_name"
     android:theme="@style/AppTheme" >
     <activity
        android:name="it.istitutotecnico.consumo.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
           <action android:name="android.intent.action.MAIN" />
           <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
     </activity>
     <activity
        android:name="it.istitutotecnico.consumo.ConsumoActivity"
        android:label="@string/title_activity_consumo"
        android:parentActivityName="it.istitutotecnico.consumo.MainActivity" >
        <meta-data
           android:name="android.support.PARENT_ACTIVITY"
           android:value="it.istitutotecnico.consumo.MainActivity" />
     </activity>
  </application>
</manifest>
```

Risorse (*res/values/strings.xml*)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Consumo</string>
  <string name="menu_settings">Settings</string>
  <string name="title_activity_consumo">KM / L</string>
  <string name="str_km">KM percorsi</string>
  <string name="str_litri">Litri consumati</string>
  <string name="str_consumo">Calcola il consumo</string>
</resources>
```

Layout (*res/layout/activity_main.xml*)

```
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity" >

  <TextView
     android:id="@+id/txtKM"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_alignParentLeft="true"
     android:layout_alignParentTop="true"
     android:text="@string/str_km"
     android:textAppearance="?android:attr/textAppearanceMedium" />

  <EditText
     android:id="@+id/editKM"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_alignParentLeft="true"
     android:layout_below="@+id/txtKM"
     android:ems="10" />

  <TextView
     android:id="@+id/txtLitri"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_alignParentLeft="true"
     android:layout_below="@+id/editKM"
     android:layout_marginTop="43dp"
     android:text="@string/str_litri"
     android:textAppearance="?android:attr/textAppearanceMedium" />

  <EditText
     android:id="@+id/editLitri"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_alignParentLeft="true"
     android:layout_below="@+id/txtLitri"
     android:ems="10" />

  <Button
     android:id="@+id/btnConsumo"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_alignParentLeft="true"
     android:layout_alignRight="@+id/editLitri"
     android:layout_centerVertical="true"
     android:onClick="calcolaConsumo"
     android:text="@string/str_consumo" />

  </RelativeLayout>
```

Layout (*res/layout/activity_consumo.xml*)

```
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".ConsumoActivity" >

  <TextView
     android:id="@+id/txtConsumo"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_centerHorizontal="true"
     android:layout_centerVertical="true"
     android:text="-"
     android:textSize="100sp" />

</RelativeLayout>
```

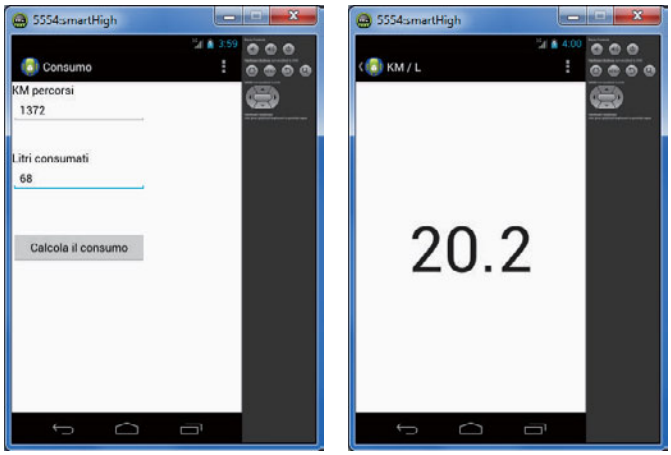
Activity (*MainActivity.java*)

```
package it.istitutotecnico.consumo;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends Activity {

  public final static String PARAM_CONSUMO = "param_consumo";
  private EditText editKM, editLitri;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);

     // crea il collegamento con le componenti grafiche
     editKM    = (EditText) findViewById(R.id.editKM);
     editLitri = (EditText) findViewById(R.id.editLitri);
  }

  public void calcolaConsumo(View view) {
     int km, litri;
     double consumo;

     km    = Integer.parseInt(editKM.getText().toString());
     litri = Integer.parseInt(editLitri.getText().toString());
```

```
      // arrotonda il consumo alla prima cifra decimale
      consumo = (1.0 * km) / litri;
      consumo = Math.round(10.0 * consumo) / 10.0;

      // apre la seconda Activity
      Intent intent = new Intent(this, ConsumoActivity.class);
      intent.putExtra(PARAM_CONSUMO, consumo);
      startActivity(intent);
   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.activity_main, menu);
      return true;
   }
}
```

Activity (*ConsumoActivity.java*)

```
package it.istitutotecnico.consumo;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.widget.TextView;

public class ConsumoActivity extends Activity {

   private TextView txtConsumo;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_consumo);

      // legge i parametri ricevuti
      Intent intent = getIntent();
      double consumo = intent.getDoubleExtra(MainActivity.PARAM_CONSUMO, 0);

      // imposta il valore nella TextView
      txtConsumo = (TextView) findViewById(R.id.txtConsumo);
      txtConsumo.setText("" + consumo);
   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.activity_consumo, menu);
      return true;
   }
}
```



AVVIARE UNA TELEFONATA

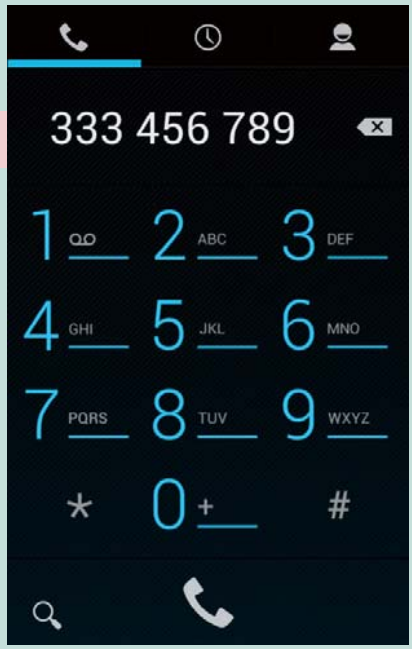
Il meccanismo di comunicazione tramite gli *Intent* è utilizzato ampiamente per scambiare le informazioni tra videate diverse della stessa applicazione o tra applicazioni diverse. Molte applicazioni predefinite in Android, per esempio il gestore delle chiamate telefoniche, sono predisposti per essere invocati tramite l'utilizzo di *Intent*.

Per comunicare con l'applicazione *Telefono* si deve creare un oggetto di classe *Intent*, passando al costruttore l'azione da eseguire (**Intent.ACTION_DIAL**) e il numero di telefono da visualizzare, indicato all'interno di una stringa di testo con il prefisso *tel:*.

```
Intent telefonata = new Intent(Intent.ACTION_DIAL,
                               Uri.parse("tel:333 456789"));
```

Dopo aver costruito l'oggetto *telefonata*, si usa il metodo **startActivity** per richiamare l'applicazione *Telefono*.

```
startActivity(telefonata);
```



Per verificare il comportamento delle precedenti istruzioni si deve realizzare un'applicazione e la si deve eseguire su un dispositivo reale, perché l'emulatore non dispone dell'applicazione predefinita *Telefono*.

Quando il metodo *startActivity* viene eseguito, l'applicazione *Telefono* viene aperta e il numero di telefono viene visualizzato al suo interno, ma la telefonata non inizia automaticamente: l'utente può modificare il numero e dare inizio alla chiamata con l'opportuno pulsante.

7 Distribuzione delle applicazioni

La disponibilità di un grande numero di applicazioni *on line*, di cui molte gratuite, è uno tra i motivi di successo dell'informatica mobile negli anni recenti. Le applicazioni per i dispositivi mobili sono distribuite dalle *software house*, o direttamente dai programmatori, nei cosiddetti **store** (*negozi virtuali*) di applicazioni. Esistono diversi *store* per Android gestiti da importanti società nel mondo dell'informatica, per esempio Amazon (*Amazon Apps*) e Samsung (*Samsung Apps*).

Google Play è lo *store* ufficiale di *Google* e permette di scaricare applicazioni, libri e musica sia gratuitamente che a pagamento. L'accesso a *Google Play* dai dispositivi mobili avviene tramite **Play Store**, un'applicazione preinstallata su tutti i dispositivi Android.
Google Play è disponibile anche sul Web all'indirizzo *http://play.google.com*. In questo caso l'elenco di applicazioni è visualizzabile con un browser Web e, collegandosi con un account *Google*, si ha anche accesso alle applicazioni installate sui propri dispositivi mobili.

Il passo finale nella realizzazione di un'applicazione è la sua **pubblicazione** in uno *store* in modo da poter essere distribuita nel mercato globale. Prima della pubblicazione, il programmatore deve eseguire i collaudi su diversi emulatori in modo da garantire un alto livello di qualità del software prodotto.

Nel seguito vengono illustrati i passi necessari per pubblicare un'applicazione su *Google Play*.

Per poter caricare applicazioni su *Google Play* è richiesta la **registrazione** come sviluppatore, che comporta il pagamento di una quota di iscrizione. Non ci sono ulteriori commissioni per la distribuzione delle applicazioni gratuite, mentre per le applicazioni a pagamento *Google* richiede, come rimborso del servizio offerto, una provvigione sul prezzo di vendita, solitamente del 30%.

La registrazione come sviluppatore può essere eseguita comunicando i propri dati personali a *Google* tramite il sito riservato agli sviluppatori (**Developer Console**) raggiungibile all'indirizzo:

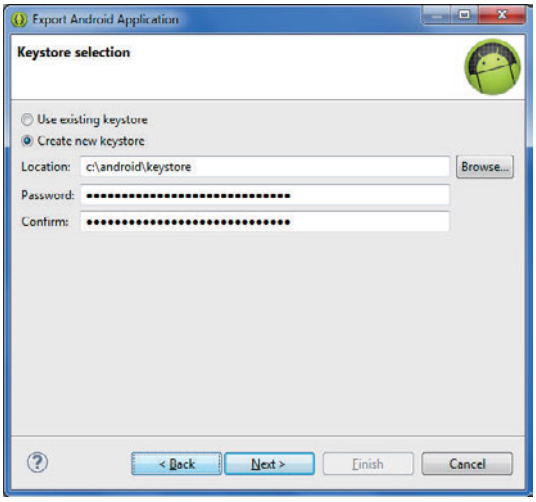
https://play.google.com/apps/publish

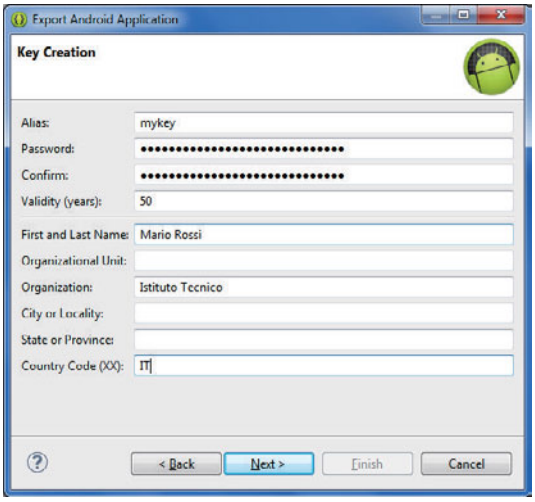
Dopo aver concluso la procedura di registrazione, viene mostrato il seguente pulsante che permette di caricare nuove applicazioni.

Pubblicare un'applicazione su *Google Play* significa caricare il file **APK** (*Android Package*), **firmato** con un certificato digitale, la cui chiave privata è conosciuta solo dal programmatore. In questo modo *Google* può garantire che la pubblicazione, e soprattutto gli aggiornamenti successivi, facciano riferimento allo stesso programmatore.

In *Eclipse* esiste una procedura guidata per firmare un'applicazione: dal menu **File**, fare clic su **Export**… e successivamente selezionare l'opzione **Export Android Application**, quindi fare clic sul pulsante **Next**.

Dopo aver indicato il nome del progetto da esportare, dobbiamo specificare il nome dell'archivio (**keystore**) contenente la chiave da usare per la firma e la password.





Per creare una nuova chiave indichiamo il percorso del file del *keystore*, nella casella *Location*, e una password. Facendo clic sul pulsante **Next** si apre una maschera in cui vengono richieste altre informazioni per la creazione della chiave: il nome della chiave (*alias*), la password, gli anni di validità e i dati identificativi del creatore.

Dopo aver fatto clic su **Next**, indichiamo il nome del file APK e facciamo clic su **Finish**.

L'applicazione, costituita dal file *.apk* appena creato, è pronta per essere caricata nello *store*.

Oltre al caricamento del file *APK firmato*, Google Play richiede l'inserimento di una **descrizione** e di uno o più **screenshot** (videate) che permettano agli utenti di visualizzare un'anteprima del prodotto software. È opportuno impostare un'**icona** che identifichi l'applicazione e diverse immagini che possono essere utilizzate per invogliare gli utenti a scaricare, ed eventualmente acquistare, l'applicazione. È inoltre consigliata la traduzione della descrizione in altre lingue, o come minimo in lingua inglese, in modo da poter offrire l'applicazione ad un pubblico di utenti più vasto.

Dopo aver pubblicato un'applicazione, la *Developer Console* consente allo sviluppatore di monitorare le **statistiche** dettagliate sull'andamento delle installazioni, per esempio il numero di installazioni e disinstallazioni suddivisi per giorno, per area geografica e per tipo di dispositivo.
Sempre nella **Developer Console** sono riportati tutti gli errori generati dall'applicazione e che sono stati automaticamente inviati a *Google* dai dispositivi mobili. Queste informazioni risultano molto utili allo sviluppatore per eseguire il *debug* e correggere errori e anomalie, in modo da migliorare il software e rilasciare una nuova versione aggiornata.

AUTOVERIFICA

Domande da 13 a 19 pag. 497-498
Problemi da 15 a 23 pag. 499

MATERIALI ON LINE

4. Le componenti grafiche di Android
5. Il layout degli oggetti
6. La geolocalizzazione

AUTOVERIFICA

DOMANDE

Android e l'informatica mobile

1 Quale delle seguenti affermazioni, riferite ai dispositivi mobili, sono vere (V) e quali false (F)?
a) Hanno una capacità di elaborazione paragonabile ai computer V F
b) Sono solitamente collegati alla rete Internet V F
c) Sono dispositivi di grandi dimensioni V F
d) Sono spesso dotati di uno schermo sensibile al tatto V F

2 Quale tra i seguenti *non* è uno strumento integrato nei moderni *smartphone* ?
a) Antenna GPS
b) Fotocamera
c) Interfaccia WiFi
d) Mouse

3 Associa ad ogni gesto della colonna di sinistra il corrispondente effetto scegliendo dalla colonna di destra:

a) toccare	1) spostare le icone
b) trascinare	2) eseguire uno scorrimento verticale
c) far scorrere	3) aumentare o ridurre lo zoom
d) pizzicare	4) attivare i pulsanti

4 Quale delle seguenti affermazioni, riferite ad *Android*, sono vere (V) e quali false (F)?
a) È un sistema operativo solo per telefoni cellulari V F
b) È un sistema operativo sviluppato da Google V F
c) È un insieme dei servizi Internet utilizzabili con uno smartphone V F
d) È un software open source V F

5 Completa le frasi seguenti utilizzando uno tra i termini elencati alla fine della domanda.
a) I ..... sono le applicazioni direttamente utilizzabili dalla schermata Home.
b) Il pulsante ..... permette di ritornare sulla schermata precedente.
c) Le notifiche sono visualizzate nella barra di .....
d) La schermata ..... si compone di più pannelli virtuali.

avvio, back, ricerca, stato, navigazione, home, preferiti, android, widget

Sviluppare applicazioni per dispositivi Android

6 Quale delle seguenti affermazioni, riferite al pacchetto ADT Bundle, sono vere (V) e quali false (F)?
a) È un pacchetto fornito da Google V F
b) Contiene solo l'Android SDK V F
c) Contiene un ambiente di sviluppo integrato V F
d) Include gli strumenti per simulare i dispositivi mobili V F

7 Quale tra i seguenti è il dispositivo virtuale usato nell'ambiente di sviluppo Android ?
a) Emulatore AVD
b) Emulatore APP
c) Emulatore SDK
d) ADT Bundle

8 Quale tra le seguenti definizioni descrive la fase di pubblicazione di un'applicazione Android?
a) Consiste nel creare un progetto e nello scrivere il codice sorgente,
b) Consiste nell'eseguire l'applicazione su uno o più dispositivi virtuali,
c) Consiste nel generare il pacchetto da distribuire agli utenti finali,
d) Consiste nel ridurre la possibilità di compiere errori,

9 Quale tra i seguenti dati *non* deve essere fornito durante la creazione di un progetto Android?
a) Il nome dell'applicazione.
b) Il nome del progetto.
c) Il nome delle componenti grafiche.
d) Il nome del package.
e) La versioni dell'SDK.

10 Completa le frasi seguenti utilizzando uno tra i termini elencati alla fine della domanda.
a) Il file ..... descrive le caratteristiche principali dell'applicazione
b) La cartella ..... contiene la descrizione dell'interfaccia grafica
c) Il Version ..... si riferisce ad un codice di versione usato internamente
d) L'area in cui costruire l'interfaccia grafica in modo visuale si chiama .....
e) La finestra ..... mostra un insieme di caratteristiche degli oggetti grafici

Code, Name, Palette, Graphical Layout, AndroidManifest.xml, activity_main.xml, res/layout, res/src/, res/value/, Properties

11 Quale delle seguenti affermazioni, riferite alla fase di test e debug, sono vere (V) e quali false (F)?
a) Il test può essere eseguito su un dispositivo virtuale V F
b) *Logcat* è un sistema per la raccolta e visualizzazione dei messaggi di errore V F
c) I messaggi di avvertimento possono essere stampati con il metodo *Log.a* V F
d) Le applicazioni vanno collaudate su un solo emulatore V F

12 Quale tra i seguenti nomi identifica un'applicazione Android non firmata ?
a) Signed APK.
b) Unsigned APK.
c) Application package.
d) Installation package.
e) Unsigned application.

L'interfaccia grafica

13 Associa ad ogni classe della colonna a sinistra la corrispondente componente grafica scegliendo dalla colonna a destra:

a) TextView	1) casella di testo
b) EditText	2) finestra
c) Button	3) etichetta
d) Activity	4) pulsante di comando

14 Quale tra le seguenti notazioni rappresenta l'*id* di una casella di testo chiamata *importo* ?
a) @id/importo
b) @string/importo
c) @+id/importo
d) @+string/importo

AUTOVERIFICA

15 In quale dei seguenti file vengono inserite tutte le risorse di tipo testo che possono essere utilizzate dall'applicazione?
a) res/layout/strings.xml
b) res/images/strings.xml
c) res/values/manifest.xml
d) res/values/strings.xml

16 Quale delle seguenti affermazioni, riferite all'interfaccia grafica in Android, sono vere (V) e quali false (F)?
a) La classe *Toast* visualizza un'immagine a pieno schermo V F
b) Il prefisso *R.layout* è usato per riferirsi ai file dell'interfaccia grafica V F
c) Il gestore degli eventi per i pulsanti di comando è indicato dalla proprietà *onClick* V F
d) Il valore *wrap_content* è usato per impostare le dimensioni degli oggetti grafici V F

17 Quale tra i seguenti termini indica l'adattamento di un'applicazione all'utilizzo in diverse nazioni?
a) geolocalizzazione
b) localizzazione
c) traduzione
d) versioning

18 Quale tra i seguenti metodi viene eseguito per passare da una *Activity* all'altra?
a) startIntent
b) putExtra
c) getIntExtra
d) startActivity

19 Quale delle seguenti affermazioni, riferite alla distribuzione delle applicazioni, sono vere (V) e quali false (F)?
a) Le applicazioni per dispositivi mobili sono distribuite in negozi virtuali online V F
b) La pubblicazione delle app consiste nel rendere disponibili le applicazioni online V F
c) La pubblicazione sullo *store* di Google prevede che il pacchetto APK sia firmato V F
d) Non si può usare *Eclipse* per firmare le applicazioni V F

PROBLEMI

Android e l'informatica mobile

1 Raccogliere e confrontare le caratteristiche di tre smartphone, elencando gli strumenti multimediali integrati, le interfacce di comunicazione e la presenza di strumenti di geolocalizzazione.

2 Seguendo lo schema del problema precedente, mettere a confronto tre modelli di tablet.

Sviluppare applicazioni per dispositivi Android

3 Collegarsi al sito di Google per prelevare e installare il pacchetto *ADT Bundle*.

4 Utilizzando *Android SDK Manager*, installare la versione 2.3 di Android.

5 Creare un emulatore con il display da 4'' e una versione di Android superiore a 4.

6 Creare un emulatore con il display a bassa risoluzione e una versione di Android inferiore a 4.

7 Creare un emulatore per simulare un tablet con un display da 10''.

8 Creare un'applicazione che visualizza il proprio nome al centro dello schermo.

9 Eseguire il test dell'applicazione precedente su tre emulatori diversi.

10 Installare l'applicazione precedente su un dispositivo reale.

11 Aggiornare l'applicazione creata precedentemente visualizzando, oltre al nome, anche il proprio cognome in un'altra etichetta. Aggiornare la versione (*Version Code* e *Version Name*) nel file *AndroidManifest.xml* e pubblicare il nuovo pacchetto.

12 Creare un'applicazione che visualizza l'ora corrente all'interno di tre etichette, indicanti separatamente le ore, i minuti e i secondi.

13 Predisporre un nuovo progetto Android in Eclipse in cui l'icona dell'applicazione sia scelta tra l'elenco delle clipart disponibili.

14 Aprire in Eclipse un'applicazione di esempio presente nell'ultima versione di Android SDK.

L'interfaccia grafica

15 Creare un'applicazione per l'inserimento dei dati anagrafici (nome e cognome) e due pulsanti di comando. Un pulsante aggiunge il dato anagrafico ad un array, mentre l'altro pulsante cancella il contenuto delle caselle di testo.

16 Creare un'applicazione per calcolare la conversione di un importo da euro a dollari. Se l'importo non è inserito correttamente, l'applicazione segnala un errore.

17 Creare un'applicazione per autenticare gli utenti, con le caselle per il nome utente e la password. Se l'utente è autorizzato viene mostrato un messaggio di conferma, altrimenti di accesso negato.

18 Estendere l'applicazione precedente, inserendo le traduzioni per la lingua inglese e francese.

19 Creare un'applicazione composta da tre pulsanti a cui è associato il numero di telefono di tre amici. Il tocco su un pulsante (evento clic) deve attivare la chiamata al numero relativo.

20 Creare un'applicazione composta da una schermata principale e da due videate secondarie. Inserire due pulsanti nella schermata principale per aprire le videate secondarie.

21 Creare un certificato digitale valido 25 anni per firmare un'applicazione.

22 Generare il pacchetto APK firmato per una delle applicazioni sviluppate precedentemente.

23 Predisporre la descrizione e gli screenshot per una delle applicazioni sviluppate precedentemente.

Focus notes

ANDROID

Android is an operating system developed by Google. It is designed for mobile devices such as *smartphones* and *tablet computers*.

Applications for Android are developed in Java language using the Android SDK. The SDK provides a set of libraries and developer tools necessary to build, test and debug apps. Developers can easily build new applications using *Eclipse* and the *ADT plugin*.

Applications are available for download through the Google Play on-line store. Users can browse, download and install thousands of apps, some of them are free and others have to be paid. In addition to browsing and searching for new apps in the Google Play, users can add their own comments and ratings.

THE MANIFEST FILE

Each Android application has an XML file called *AndroidManifest.xml*. The manifest defines the components of the application and several parameters. It contains a description for each *Activity* that composes the user interface. The manifest can also specify the application icon, the version code and name, permissions and SDK requirements.

The manifest tag includes several attributes:
- *versionCode*: defines the current application version as an integer that increases with each version;
- *versionName*: specifies a public name, informing users about the application's version;
- *uses-sdk*, defines the minimum and maximum SDK version required by the application.

A manifest tag can contain only one application tag that specifies the name and the icon of the application.

For every *Activity* in the application, an activity tag is needed, otherwise the application will not be able to open it and an exception will be raised. The main attribute of an activity tag is the *name* that refers to the name of the relevant Java class.

Focus notes

Glossary

APK
The file format used by Android applications.

activity
A single screen of an Android application.

cloud computing
The access and the use of shared resources via the Internet.

developer console
A Google Web tool for developers, used to manage the application publishing process.

emulator
A virtual device that can simulate several physical devices.

keystore
A container of private keys and public certificates.

layout file
An XML file that describes the structure of the user interface for an activity.

localisation
The process of translating an application into different languages.

manifest file
An XML file that contains information about the application.

mobile computing
Processing activity and interaction with a computer that can be transported during its usage.

smartphone
A mobile phone with more processing power.

tablet
A mobile computer with a wide touch screen.

Acronyms

3G	3rd Generation
4G	4th Generation
ADT	Android Developer Tools
APK	Android Package
AVD	Android Virtual Device
GPS	Global Positioning System
iOS	iPhone Operating System
MP	MegaPixel
SDK	Software Development Kit
SIM	Subscriber Identity Module
SP	Scale-independent Pixels
USB	Universal Serial Bus
WiFi	Wireless Fidelity (IEEE 802.11b wireless networking)

SCHEDA DI AUTOVALUTAZIONE

CONOSCENZE

- ☐ Informatica mobile
- ☐ Dispositivi con touchscreen
- ☐ Sistema operativo Android
- ☐ Apps
- ☐ Ambiente di sviluppo
- ☐ Emulatori di dispositivi
- ☐ Fasi di sviluppo di un'applicazione
- ☐ Il file manifest
- ☐ Le componenti grafiche dell'interfaccia
- ☐ Localizzazione del software
- ☐ Le activity
- ☐ Distribuzione dell'applicazione

ABILITÀ

- ☐ Riconoscere i gesti con un touchscreen
- ☐ Installare il kit di sviluppo di Android
- ☐ Riconoscere gli elementi di base dell'interfaccia Android
- ☐ Creare gli emulatori di dispositivi
- ☐ Creare una semplice app
- ☐ Scrivere il contenuto di un file manifest
- ☐ Progettare l'interfaccia dell'applicazione
- ☐ Eseguire il test e il debug
- ☐ Usare le applicazioni di esempio del SDK
- ☐ Creare risorse alternative per lingue diverse
- ☐ Utilizzare le activity nell'applicazione
- ☐ Scrivere il codice per avviare una telefonata
- ☐ Individuare i passi per pubblicare l'applicazione in uno store

SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 539

Sicurezza

1 La sicurezza dei sistemi informatici

La sicurezza nei sistemi informatici riguarda sia gli aspetti hardware che gli aspetti software. Considerando le apparecchiature, i principali problemi sono connessi ai guasti che si possono verificare durante il normale funzionamento.

Le applicazioni software non sono soggette a guasti ma solitamente a malfunzionamenti, mentre a livello di sicurezza le preoccupazioni maggiori sono rivolte alla gestione degli accessi ai dati.

• Calamità naturali

Una delle cause che può incidere negativamente, e talvolta in modo irreparabile, sulla sicurezza del sistema informatico di un'azienda nel suo complesso è rappresentata dalle calamità naturali.

Le calamità prese in considerazione sono quelle connesse a terremoti, incendi e alluvioni.

Le aziende hanno sentito il bisogno di dotarsi di procedure particolari che consentano di far ripartire l'insieme delle attività in pochi giorni anche in caso di distruzione totale di un'unità lavorativa composta spesso di migliaia di persone.

Si tratta di costruire norme che definiscano tutte le apparecchiature e tutte le informazioni indispensabili per ripartire, di creare depositi sicuri in cui mettere una copia di tutti i documenti necessari alla ripartenza, e infine di definire quali attività e quali figure aziendali, o esterne all'azienda, devono intervenire secondo una scaletta articolata e prestabilita.

La salvaguardia rispetto a incendi, alluvioni e in qualche misura terremoti, può essere risolta mediante tecniche costruttive particolari, come edifici bunker o comunque che rispettano le norme antisismiche, con un sistema di rilevazione automatica dei fumi collegata ad impianti per la caduta di acqua (in assenza di problemi elettrici) o di polveri sul punto dove sono stati rilevati i fumi.

• Attentati terroristici

Solitamente il blocco del sistema informatico può produrre la paralisi dell'azienda. Il sistema informatico è diventato un elemento fondamentale per l'operatività dell'azienda e per questo può diventare il centro di interesse per chi vuole danneggiare l'attività aziendale.

Il sistema informatico può essere al centro di attentati terroristici che possono causare il blocco totale della struttura colpendo un punto, di ricatti da parte della malavita organizzata o più semplicemente da mitomani.

Come risposta le aziende tendono a collocare le principali apparecchiature informatiche in luoghi sempre più protetti al centro delle aziende o in veri e propri bunker, al riparo da chi voglia aggredire con la forza.

• Guasti e interruzioni hardware

Nelle situazioni dove non è consentita l'interruzione dell'attività di un sistema di elaborazione (*downtime*), come per esempio in una banca o in un supermercato, la sicurezza viene garantita nella parte hardware attraverso la duplicazione di parti o dell'intero sistema.

Queste tecniche vengono indicate con il termine **fault tolerance** (tolleranza del guasto) e riguardano principalmente le memorie di massa nei server delle reti e nei sistemi di medie e grandi dimensioni.
Il *fault tolerance* viene realizzato a livelli diversi:

- il metodo più semplice si chiama **mirroring** e consente di avere nelle unità di memoria di massa due copie identiche dello stesso disco (oppure solo di alcuni archivi particolarmente importanti e preselezionati); quando un'operazione di I/O riscontra un errore, l'elaborazione non viene interrotta potendo essa utilizzare la copia alternativa.
- un secondo livello di tolleranza del guasto viene realizzato con la tecnica del **duplexing**, che consiste nella duplicazione dell'unità di controllo dei dischi (*controller*) oltre che dei dischi. L'utente può continuare l'attività di elaborazione anche nel caso di un guasto al controller o al disco, e diminuiscono i rischi di interruzione.
- il terzo livello riguarda la **duplicazione dell'intero sistema**, del server nel caso di reti locali, e del mainframe nel caso di un sistema di grandi dimensioni.
- ci sono a disposizione altre tecniche meno costose rispetto alla duplicazione parziale del sistema, che vengono indicate con la sigla **RAID** (*Redundant Array of Inexpensive Disk*). Questa tecnologia consiste nel distribuire i dati su un insieme di dischi, in modo che sia possibile ricostruire per via matematica tutti i dati eventualmente persi da uno dei dischi.

• Errori del personale

L'enorme sviluppo dell'elettronica moderna consente di accumulare grandi quantità di dati in spazi molto piccoli e secondo modalità che non sono quasi mai simili alle operazioni svolte secondo la gestione manuale, per cui anche errori banali possono produrre disastri.

Le norme per prevenire tali eventualità di solito consistono in:

- una netta separazione tra l'ambiente di esercizio e l'ambiente di sviluppo, per evitare che il programmatore durante le sue prove inquini o peggio distrugga i dati che servono per le procedure aziendali;
- copia periodica di tutti i dati aziendali su supporti di memoria da conservare in luogo sicuro per un successivo ripristino se richiesto: normalmente vengono conservati più salvataggi di giornate differenti al fine di una maggiore protezione;
- intervento di livelli professionali differenti in caso di ripetizione dell'errore in modo da evitare qualsiasi rischio;
- definizione di regole organizzative e figure professionali preposte a verificare tutte le anomalie nei dati di input o di output.

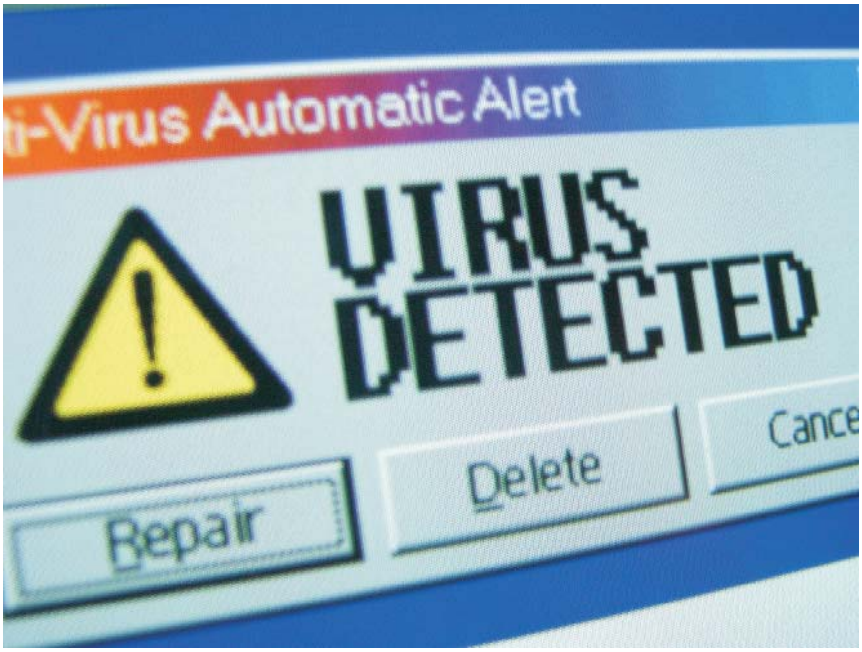
Per ridurre al minimo gli errori del personale operativo si devono prevedere le attività di formazione, tramite corsi di aggiornamento, in modo da sviluppare e consolidare le competenze informatiche delle persone che interagiscono con il sistema informativo aziendale.

• Virus

Tra i problemi di sicurezza che coinvolgono i sistemi software, applicazioni e dati, c'è quello dei virus informatici.

I virus sono programmi chiamati così perché sono stati costruiti con lo scopo di:

- infettare un programma senza alterarne in apparenza il funzionamento;
- rendersi visibili in qualche modo;
- autoriprodursi nell'ambiente in cui sono inseriti creando un'epidemia che può essere dannosa (distruzione di dati e programmi) o solo contagiosa estendendosi a tutti i programmi e a tutti i computer.



Possono essere prodotti per gioco oppure talvolta con finalità ricattatorie.
Non esiste un modo attraverso il quale distinguere un programma da un virus e quindi un virus diventa tale solo dopo la sua scoperta e la sua denominazione.
La protezione contro i virus può essere realizzata attraverso il controllo sistematico di tutti i supporti che vengono usati. Esistono poi organizzazioni e software house che si sono specializzate nelle identificazioni dei virus e degli effetti che producono, e nella costruzione di programmi vaccino per eliminarli, chiamati **antivirus**.
In alcuni casi la rimozione del virus può richiedere la cancellazione dei programmi danneggiati: pertanto diventa fondamentale il lavoro di filtro sugli accessi, sui dati e sul software, con scopi di prevenzione.
Gli strumenti antivirus devono essere costantemente aggiornati con le definizioni dei nuovi virus scoperti, ma questo non è sufficiente per azzerare le probabilità di essere colpiti da un virus. È quindi opportuno seguire una politica di copie di sicurezza periodiche, in modo tale da poter ripristinare i dati in caso di corruzione o cancellazione da parte di virus.

• Sicurezza e integrità dei dati

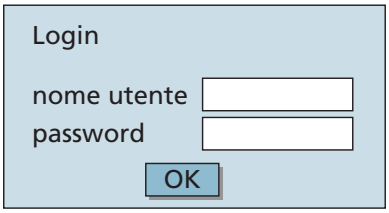
La gestione di un sistema di elaborazione ha un aspetto cruciale rappresentato dalla necessità di garantire la **consistenza** dei dati in esso memorizzati, sia nel caso di un computer personale, sia nel caso di sistemi informatici aziendali: i dati devono essere significativi ed essere effettivamente utilizzabili nelle applicazioni. I dati devono quindi essere protetti per impedire perdite accidentali dovute a improvvise interruzioni dell'attività del sistema, guasti hardware o interventi dannosi da parte di utenti o programmi: la protezione deve riguardare anche gli interventi dolosi sui dati dovuti ad accessi non autorizzati con operazioni di lettura, modifica o cancellazione.
In sostanza **sicurezza** significa impedire che il sistema e i suoi dati vengano danneggiati da interventi accidentali e non autorizzati; **integrità** significa garantire che le operazioni effettuate sul sistema e sui dati da parte di utenti autorizzati non provochino una perdita di consistenza ai dati.
Gli archivi elettronici hanno più o meno gli stessi problemi degli archivi cartacei, per esempio un'unità a dischi si può rompere provocando la perdita di tutti i dati in essa contenuti, oppure un errore degli operatori può provocare la cancellazione non voluta di uno o più archivi: pertanto si rende necessaria la copia periodica degli archivi e la conservazione delle copie in posti sicuri.

L'attività di copia di archivi di grandi dimensioni ha durate significative e di solito viene eseguita giornalmente: questa attività si chiama **backup** (salvataggio). L'attività in senso inverso si chiama **restore** (ripristino) e serve per ricaricare su disco tutti o parte degli archivi salvati precedentemente.

L'accesso non autorizzato ai dati rappresenta un altro aspetto di sicurezza legato agli archivi. In un ambiente che comprende tutti i dati aziendali occorre avere una responsabilità centralizzata in grado di distribuire le autorizzazioni e le modalità di accesso.

• Profilazione degli utenti

In generale ad ogni utente viene associato un **profilo** il cui accesso è controllato mediante l'**identificativo dell'utente** (*user ID* o *username*) e la **parola d'ordine** (*password*): il primo rappresenta il nome, che può anche essere pubblico, con il quale l'utente accede alla procedura di riconoscimento (**login**), mentre la seconda rappresenta la parola chiave, di solito scelta dall'utente stesso, che è privata e riservata.
Per questo motivo la password dovrebbe essere non banale (nomi di persona, date di nascita, ecc.) e dovrebbe essere frequentemente cambiata. La password deve essere lunga almeno 8 caratteri e deve essere una combinazione di cifre e di lettere.



Al profilo sono poi associati i **diritti di accesso**, cioè le azioni, i permessi e i divieti relativi alle attività svolte dall'utente: quali programmi può utilizzare, quali archivi può consultare, quali permessi su un singolo archivio (solo lettura oppure lettura e scrittura), la possibilità di installare nuovi programmi. Questi controlli vengono realizzati a diversi livelli: gestione delle basi di dati, sistemi operativi, reti di utenti. Essi sono gestiti dall'**Amministratore del sistema** (*system administrator*), che nel personal computer coincide con l'utente stesso, mentre per sistemi più complessi o per le reti è una specifica figura professionale del settore informatico.

• Pirati informatici

Nel linguaggio corrente il termine **hacker** viene usato come sinonimo di *criminale* o *pirata informatico*, anche se il termine corretto dovrebbe essere *cracker.* Esso sta ad indicare persone che si collegano, con sistemi di elaborazione, sfruttando i meccanismi di rete senza avere l'autorizzazione all'accesso: questi inserimenti indesiderati con messaggi o programmi possono anche portare alla paralisi del sistema.

• Crimini informatici e sicurezza

L'Italia si è allineata alle disposizioni comunitarie in materia di provvedimenti legislativi relativi alla sicurezza informatica. La **legge n. 547 del 14 dicembre 1993**, "*Modificazioni e integrazioni alle norme del codice penale e del codice di procedura penale in tema di criminalità informatica*", stabilisce le pene per **reati informatici** come il danneggiamento di dati e programmi. Ecco alcuni passi significativi della legge. Innanzitutto all'articolo 392 del Codice Penale, dopo il secondo comma è aggiunto il seguente:

> *"si ha altresì violenza sulle cose allorché un programma informatico viene alterato, modificato o cancellato in tutto o in parte ovvero viene impedito o turbato il funzionamento di un sistema informatico o telematico".*

La pena all'art. 420 (*attentato a impianti di pubblica utilità*), si applica anche a chi commette un fatto diretto a danneggiare o distruggere sistemi informatici o telematici di pubblica utilità. Rilevanza penale si ha anche per quanto riguarda la falsità dei documenti informatici.

Art. 615-ter. (*accesso abusivo a un sistema informatico o telematico*).
"Chiunque abusivamente si introduce in un sistema informatico o telematico protetto da misure di sicurezza ovvero vi si mantiene contro la volontà espressa o tacita di chi ha diritto di escluderlo, è punito con la reclusione fino a tre anni".

Art. 615-quinquies
"Chiunque diffonde, comunica o consegna un programma informatico da lui stesso o da altri redatto, avente per scopo o per effetto il danneggiamento di un sistema informatico o telematico, dei dati o dei programmi in esso contenuti o ad esso pertinenti ... è punito con la reclusione sino a due anni e con la multa sino a euro 10.329"

2 La sicurezza delle reti

Il problema della sicurezza dei computer connessi alla rete è diventato molto importante con la crescita di Internet: si parla spesso di virus, attacchi informatici, sistemi violati e truffe informatiche.
Il problema risiede nel fatto che fisicamente i computer, per definizione, sono tutti collegati alla stessa rete. Questo è il grande vantaggio di Internet, che espone però a rischi che possono coinvolgere un numero considerevole di computer.
I protocolli e i servizi di Internet non sono la causa principale dei problemi di sicurezza: le tecnologie alla base di Internet si evolvono continuamente e rapidamente per risolvere i problemi che si presentano nel tempo, producendo nuovi servizi e software più sicuri.
La parte più importante di Internet non sono i computer e le connessioni che compongono la rete delle reti (parte hardware), ma i programmi che la gestiscono e ne permettono l'uso (parte software). I programmi sono sviluppati da programmatori, che sono soggetti a errori umani. L'errore presente in un programma (indicato nel gergo informatico con il termine **bug**) può essere sfruttato per far compiere al programma stesso operazioni non previste.
Per esempio molti virus, che si sono diffusi negli anni recenti, hanno utilizzato la presenza di errori o imprecisioni presenti nel server Web o nei programmi di posta elettronica.
Le vulnerabilità software possono essere più o meno gravi, essere più frequenti in certi programmi invece che in altri, ma la responsabilità del computer è sempre a carico dell'utente. Quindi è necessario che gli utenti prendano tutte le precauzioni possibili per garantire la sicurezza del proprio computer, in particolar modo se connesso a Internet.
Alcuni aspetti di sicurezza possono essere gestiti a livello di rete aziendale, attraverso diverse tecniche:

- Il **firewall** (letteralmente, *muro taglia fuoco*) è un dispositivo o un insieme di dispositivi che hanno l'obiettivo di creare una separazione tra Internet, intesa come rete pubblica, e la rete aziendale, ritenuta vulnerabile, al fine di preservarla da intrusioni fraudolente o non autorizzate. Il firewall può essere un computer specializzato in queste funzioni di controllo della rete, oppure un programma software installato sul server della rete.
- Il **tunneling** può essere definito come la via per trasferire dati tra due reti simili attraverso una rete intermedia. Il tunneling incapsula un tipo di pacchetto di un generico protocollo in un pacchetto di un altro protocollo trasportabile in Internet. Prima che l'incapsulamento abbia luogo, i pacchetti sono crittografati, in modo che i dati non siano leggibili da chiunque tenga costantemente monitorata la rete con scopi fraudolenti. Questi pacchetti incapsulati viaggiano attraverso Internet fino a raggiungere la loro destinazione. All'arrivo i pacchetti riprendono l'aspetto originario, perché sono i due punti di entrata e di uscita del tunnel che conoscono e utilizzano il protocollo di incapsulamento.

- La **VPN** (*Virtual Private Network*) è un'estensione di una Intranet privata attraverso una rete pubblica, come Internet, che stabilisce una connessione privata sicura, essenzialmente attraverso un tunnel privato. La VPN racchiude utenti remoti, sedi aziendali distaccate e reti di vendita, in una rete aziendale estesa.

Altri aspetti di sicurezza riguardano il controllo **antivirus** e l'aggiornamento delle definizioni dei virus, oltre all'applicazione di filtri sul server di posta per proteggere la rete dallo *spamming*. Il termine **spam** indica l'invio di posta indesiderata a un grande numero di caselle di posta.

Spesso questi messaggi, oltre ad intasare inutilmente le caselle email, contengono allegati portatori di virus.
È ovvio comunque che la sicurezza viene garantita anche dal comportamento degli utenti e da un efficace metodo di salvataggio periodico dei dati più importanti (*backup*).

Una tecnica che viene utilizzata da truffatori, ma che non può essere classificata come virus, è il *phishing*. Il **phishing** è una frode informatica che consiste nel creare una copia identica di un sito Internet (per esempio la *home page* di una banca) e indurre l'utente ad inserire i propri dati personali. Il truffatore può successivamente utilizzare i dati ottenuti per i propri scopi.

La truffa tramite *phishing* utilizza di solito la posta elettronica per cui spesso si parla di *phishing mail*.

Le **phishing mail** hanno lo stesso aspetto dei normali messaggi di email. Il destinatario ha l'impressione di aver ricevuto una regolare comunicazione ufficiale da una banca, una società o dalle Poste italiane. Di solito il messaggio contiene un link che porta l'utente ad una pagina del tutto identica alla *home page* del sito istituzionale, ma in realtà residente su un server esterno gestito dai truffatori. Indicando erroneamente i dati del proprio account, come indirizzo mail, username e password, oppure dati e riferimenti bancari (numero conto corrente, codice segreto del Bancomat, numero di carta di credito) avviene il cosiddetto *phishing* dei suoi dati. Gli autori della truffa possono quindi abusare di questi dati, comportando danni economici, ma anche conseguenze penali.

Vediamo di seguito alcune regole pratiche per prevenire le frodi da *phishing*:

1. Normalmente le comunicazioni tramite email hanno:
   - un oggetto significativo,
   - non sono generiche,
   - contengono i nostri dati personali
   - non richiedono mai la password a utenti o clienti.

   Una mail che non rispetta tutti questi requisiti è da considerarsi sospetta.
2. Se si ha ancora qualche dubbio e si vuole comunque visitare il sito istituzionale, non fare clic sul link presente nella mail (che potrebbe essere contraffatto), ma digitarlo manualmente nella barra degli indirizzi.
3. Spesso i messaggi di phishing sono generati con traduttori automatici dei testi e quindi contengono molti errori grammaticali o utilizzano termini della lingua italiana in modo non appropriato.
4. L'indirizzo del sito malevolo è diverso da quello originale per cui, prima di inserire qualsiasi dato, controllare sempre che nella casella *Indirizzo* del browser (URL) sia indicato l'indirizzo esatto della Banca o dell'Ente. In caso di dubbio contattare la propria Filiale o il servizio ICT dell'Ente.
5. Solitamente banche, aziende e poste utilizzano sempre protocolli crittografati, per cui è opportuno controllare che il protocollo utilizzato sia *https* e non il semplice *http*.

3 La sicurezza nei luoghi di lavoro

Il termine *ergonomia* indica l'insieme delle tecniche, degli strumenti e degli accorgimenti che consentono di creare un ambiente di lavoro sicuro e confortevole.

Le norme per la tutela della salute e della sicurezza nei luoghi di lavoro sono contenute nel **Testo Unico della sicurezza** (Decreto Legislativo n. 81 del 2008). Esso si applica a tutti i settori di attività, privati e pubblici, e a tutte le tipologie di rischio; riguarda inoltre tutte le categorie di lavoratori.

Nel Testo Unico sono precisati gli obblighi dei datori di lavoro e i comportamenti dei lavoratori per garantire la salute e la prevenzione ed evitare situazioni di rischio e di pericolo; inoltre sono fissati i requisiti degli ambienti e delle attrezzature di lavoro.

Sono importanti alcune definizioni di base:

- **Prevenzione**: il complesso delle disposizioni o misure necessarie anche secondo la particolarità del lavoro, l'esperienza e la tecnica, per evitare o diminuire i rischi professionali nel rispetto della salute della popolazione e dell'integrità dell'ambiente esterno.
- **Salute**: stato di completo benessere fisico, mentale e sociale, non consistente solo in un'assenza di malattia o d'infermità.
- **Pericolo**: proprietà o qualità intrinseca di un determinato fattore avente il potenziale di causare danni.
- **Rischio**: probabilità di raggiungimento del livello potenziale di danno nelle condizioni di impiego o di esposizione ad un determinato fattore o agente oppure alla loro combinazione.

Figure per la sicurezza e la salute nei luoghi di lavoro

Il Testo Unico individua le principali figure che devono occuparsi di sicurezza e salute nei luoghi di lavoro.
Esse sono:

- **Datore di lavoro**, il soggetto titolare del rapporto di lavoro con il lavoratore o, comunque, il soggetto che ha i poteri decisionali e di spesa. Nelle Pubbliche Amministrazioni per datore di lavoro si intende il Dirigente al quale spettano i poteri di gestione. Il Datore di lavoro può essere anche un lavoratore autonomo oppure il titolare di un'impresa familiare.
- **Responsabile del Servizio di Prevenzione e Protezione dai rischi** (RSPP), persona designata dal datore di lavoro per coordinare il servizio di prevenzione e protezione dai rischi; può essere interno o esterno alla azienda; nelle piccole imprese può coincidere con il Datore di lavoro.
- **Rappresentante dei Lavoratori per la Sicurezza** (RLS), persona eletta o designata per rappresentare i lavoratori per quanto concerne gli aspetti della salute e della sicurezza durante il lavoro
- **Medico Competente**, medico che collabora con il Datore di lavoro ai fini della valutazione dei rischi ed è nominato dallo stesso per effettuare la sorveglianza sanitaria.
- **Addetti al servizio di prevenzione e protezione**, persone addette alle emergenze (prevenzione incendi, evacuazione e pronto soccorso).

Posizione corretta nel lavoro al computer

I rischi per la salute delle persone che lavorano per un tempo prolungato davanti al computer sono principalmente legati a:

- mal di schiena dovuto a posture errate durante il lavoro;
- danni ai polsi causati dall'uso prolungato della tastiera;
- bruciore agli occhi e disturbi visivi causati da una cattiva illuminazione dell'ambiente o dall'uso di schermi non adatti;
- dolori articolari e muscolari, in particolare al collo e alle spalle.

La prima regola per evitare questi fastidi è sicuramente organizzare in modo ottimale la propria postazione di lavoro, all'interno di un ambiente sicuro e confortevole:

- il monitor deve essere posto ad altezza leggermente inferiore rispetto agli occhi;
- lo schermo deve essere antiriflesso in modo da non riflettere la luce emessa dalle lampade che illuminano l'ambiente;
- la luce delle finestre non deve essere indirizzata sullo schermo;
- la tastiera deve avere una posizione in modo da non affaticare le braccia, la sedia ergonomica è inclinabile e regolabile in altezza ed ha lo schienale alto;
- il mouse deve avere una forma per adattarsi alla mano in modo da non affaticare il polso, con eventuale uso di un tappetino;
- la schiena deve essere diritta e le gambe ad angolo retto con eventuale uso di uno sgabello poggiapiedi;
- gli avambracci devono essere orizzontali sulla scrivania.

Altre semplici regole riguardano le modalità di lavoro e l'organizzazione del tempo:

- stirarsi regolarmente;
- variare i compiti da svolgere in modo da essere costretti a muoversi, evitando di sedere tutto il giorno nella stessa posizione;

- eseguire brevi attività che richiedono di camminare (per esempio, collocare la stampante lontano dal computer, in modo da doversi alzare per prendere i fogli, oppure archiviare la documentazione in un faldone);
- fare frequentemente brevi pause, per esempio una pausa di 15 minuti ogni due ore di attività;
- allontanare periodicamente lo sguardo dal monitor verso una distanza lontana per un rilassamento della vista.

Di seguito sono riportate le indicazioni contenute nel *Testo Unico della sicurezza* (**Allegato XXXIV**).

REQUISITI MINIMI PER IL LAVORO CON LE ATTREZZATURE INFORMATICHE

1. Attrezzature

a) Osservazione generale.
L'utilizzazione in sé dell'attrezzatura non deve essere fonte di rischio per i lavoratori.

b) Schermo.
La risoluzione dello schermo deve essere tale da garantire una buona definizione, una forma chiara, una grandezza sufficiente dei caratteri e, inoltre, uno spazio adeguato tra essi.
L'immagine sullo schermo deve essere stabile; esente da farfallamento, tremolio o da altre forme di instabilità.
La brillanza e/o il contrasto di luminanza tra i caratteri e lo sfondo dello schermo devono essere facilmente regolabili da parte dell'utilizzatore del videoterminale e facilmente adattabili alle condizioni ambientali.
Lo schermo deve essere orientabile ed inclinabile liberamente per adeguarsi facilmente alle esigenze dell'utilizzatore.
È possibile utilizzare un sostegno separato per lo schermo o un piano regolabile.
Sullo schermo non devono essere presenti riflessi e riverberi che possano causare disturbi all'utilizzatore durante lo svolgimento della propria attività.

Lo schermo deve essere posizionato di fronte all'operatore in maniera che, anche agendo su eventuali meccanismi di regolazione, lo spigolo superiore dello schermo sia posto un po' più in basso dell'orizzontale che passa per gli occhi dell'operatore e ad una distanza degli occhi pari a circa 50-70 cm, per i posti di lavoro in cui va assunta preferenzialmente la posizione seduta.

c) Tastiera e dispositivi di puntamento.

La tastiera deve essere separata dallo schermo e facilmente regolabile e dotata di meccanismo di variazione della pendenza onde consentire al lavoratore di assumere una posizione confortevole e tale da non provocare l'affaticamento delle braccia e delle mani.
Lo spazio sul piano di lavoro deve consentire un appoggio degli avambracci davanti alla tastiera nel corso della digitazione, tenendo conto delle caratteristiche antropometriche dell'operatore.
La tastiera deve avere una superficie opaca onde evitare i riflessi. La disposizione della tastiera e le caratteristiche dei tasti devono agevolarne l'uso. I simboli dei tasti devono presentare sufficiente contrasto ed essere leggibili dalla normale posizione di lavoro.
Il mouse o qualsiasi dispositivo di puntamento in dotazione alla postazione di lavoro deve essere posto sullo stesso piano della tastiera, in posizione facilmente raggiungibile e disporre di uno spazio adeguato per il suo uso.

d) Piano di lavoro.

Il piano di lavoro deve avere una superficie a basso indice di riflessione, essere stabile, di dimensioni sufficienti a permettere una disposizione flessibile dello schermo, della tastiera, dei documenti e del materiale accessorio.
L'altezza del piano di lavoro fissa o regolabile deve essere indicativamente compresa fra 70 e 80 cm. Lo spazio a disposizione deve permettere l'alloggiamento e il movimento degli arti inferiori, nonché l'ingresso del sedile e dei braccioli se presenti.
La profondità del piano di lavoro deve essere tale da assicurare una adeguata distanza visiva dallo schermo.
Il supporto per i documenti deve essere stabile e regolabile e deve essere collocato in modo tale da ridurre al minimo i movimenti della testa e degli occhi.

e) Sedile di lavoro.

Il sedile di lavoro deve essere stabile e permettere all'utilizzatore libertà nei movimenti, nonché una posizione comoda. Il sedile deve avere altezza regolabile in maniera indipendente dallo schienale e dimensioni della seduta adeguate alle caratteristiche antropometriche dell'utilizzatore.
Lo schienale deve fornire un adeguato supporto alla regione dorso-lombare dell'utente. Pertanto deve essere adeguato alle caratteristiche antropometriche dell'utilizzatore e deve avere altezza e inclinazione regolabile. Nell'ambito di tali regolazioni l'utilizzatore dovrà poter fissare lo schienale nella posizione selezionata.
Lo schienale e la seduta devono avere bordi smussati. I materiali devono essere pulibili e presentare un livello di permeabilità tali da non compromettere il comfort dell'utente.
Il sedile deve essere dotato di un meccanismo girevole per facilitare i cambi di posizione e deve poter essere spostato agevolmente secondo le necessità dell'utilizzatore. Un poggiapiedi sarà messo a disposizione di coloro che lo desiderino per far assumere una postura adeguata agli arti inferiori. Il poggiapiedi non deve spostarsi involontariamente durante il suo uso.

f) Computer portatili.
L'impiego prolungato dei computer portatili necessita della fornitura di una tastiera e di un mouse o altro dispositivo di puntamento esterni nonché di un idoneo supporto che consenta il corretto posizionamento dello schermo.

2. Ambiente

a) Spazio
Il posto di lavoro deve essere ben dimensionato e allestito in modo che vi sia spazio sufficiente per permettere cambiamenti di posizione e movimenti operativi.

b) Illuminazione
L'illuminazione generale e specifica (lampade da tavolo) deve garantire un illuminamento sufficiente e un contrasto appropriato tra lo schermo e l'ambiente circostante, tenuto conto delle caratteristiche del lavoro e delle esigenze visive dell'utilizzatore.
Riflessi sullo schermo, eccessivi contrasti di luminanza e abbagliamenti dell'operatore devono essere evitati disponendo la postazione di lavoro in funzione dell'ubicazione delle fonti di luce naturale e artificiale.
Si dovrà tener conto dell'esistenza di finestre, pareti trasparenti o traslucide, pareti e attrezzature di colore chiaro che possono determinare fenomeni di abbagliamento diretto e/o indiretto e/o riflessi sullo schermo. Le finestre devono essere munite di un opportuno dispositivo di copertura regolabile per attenuare la luce diurna che illumina il posto di lavoro.

c) Rumore
Il rumore emesso dalle attrezzature presenti nel posto di lavoro non deve perturbare l'attenzione e la comunicazione verbale.

d) Radiazioni
Tutte le radiazioni, eccezion fatta per la parte visibile dello spettro elettromagnetico, devono essere ridotte a livelli trascurabili dal punto di vista della tutela della sicurezza e della salute dei lavoratori.

e) Parametri microclimatici
Le condizioni microclimatiche non devono essere causa di discomfort per i lavoratori.
Le attrezzature in dotazione al posto di lavoro non devono produrre un eccesso di calore che possa essere fonte di discomfort per i lavoratori.

3. Interfaccia elaboratore/uomo

All'atto dell'elaborazione, della scelta, dell'acquisto del software, o allorché questo venga modificato, come anche nel definire le mansioni che implicano l'utilizzazione di unità videoterminali, il datore di lavoro terrà conto dei seguenti fattori:

a) il software deve essere adeguato alla mansione da svolgere;
b) il software deve essere di facile uso adeguato al livello di conoscenza e di esperienza dell'utilizzatore. Inoltre nessun dispositivo di controllo quantitativo o qualitativo può essere utilizzato all'insaputa dei lavoratori;
c) il software deve essere strutturato in modo tale da fornire ai lavoratori indicazioni comprensibili sul corretto svolgimento dell'attività;
d) i sistemi devono fornire l'informazione di un formato e ad un ritmo adeguato agli operatori;
e) i principi dell'ergonomia devono essere applicati in particolare all'elaborazione dell'informazione da parte dell'uomo.

Precauzioni nell'uso delle apparecchiature informatiche

I fornitori di apparecchiature informatiche devono rispettare alcuni importanti standard internazionali e nazionali che riguardano la sicurezza elettrica e meccanica.
Tuttavia l'utente deve adottare alcune precauzioni per ridurre i rischi di incendi, di lesioni fisiche o danni alle apparecchiature:

- usare cavi, spine, prese e prolunghe a norma;
- non lasciare cavi disposti in posizioni dove c'è un passaggio di persone, per evitare di inciamparvi o di passarci sopra;
- non sovraccaricare le prese di corrente elettrica, evitando l'uso di prese multiple e di adattatori di spine;
- evitare di esporre le apparecchiature all'acqua;
- non aprire i coperchi del computer se non si è sicuri del tipo di intervento da effettuare e, in ogni caso, fare qualsiasi manutenzione sulle parti interne del computer solo dopo averlo scollegato dalla corrente elettrica;
- non coprire le fessure e le aperture per la ventilazione per assicurare una protezione dal surriscaldamento.

APPENDICE

Linguaggio Java

Conversioni tra sistemi di numerazione

Decimale	Esadecimale	Ottale	Binario
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

Parole riservate

abstract	default	if	private	throw
boolean	do	implements	protected	throws
break	double	import	public	transient
byte	else	instanceof	return	try
case	extends	int	short	void
catch	final	interface	static	volatile
char	finally	long	super	while
class	float	native	switch	
const	for	new	synchronized	
continue	goto	package	this	

Tipi primitivi

Tipo	Numero di bit	Numero di byte	Valore più piccolo	Valore più grande
boolean	1	1	false	true
char	16	2	'\u0000' [0]	'\uFFFF' [2^{16}-1]
byte	8	1	-128 [-2^{7}]	+127 [2^{7}-1]
short	16	2	-32.768 [-2^{15}]	+32,767 [2^{15}-1]
int	32	4	-2.147.483.648 [-2^{31}]	+2.147.483.647 [2^{31}-1]
long	64	8	-9.223.372.036.854.775.808 [-2^{63}]	+9.223.372.036.854.775.807 [2^{63}-1]
float	32	4	±1,40129846432481707e-45	±3,40282346638528860e+38 (da 6 a 7 cifre significative)
double	64	8	±4,94065645841246544e-324	±1,79769313486231570e+308 (da 14 a 15 cifre significative)

Sequenze di escape

sequenza	descrizione	terminologia inglese
\\	barra contraria	*backslash*
\?	punto di domanda	*question mark*
\'	apice singolo	*single quote*
\0	fine stringa	*end of string*
\b	una battuta indietro	*backspace*
\"	doppio apice	*double quote*
\n	a capo	*new line*
\r	a capo sulla stessa riga	*carriage return*
\t	tabulazione	*tab*

Colori predefiniti

- Color.black
- Color.blue
- Color.cyan
- Color.darkGray
- Color.gray
- Color.green
- Color.lightGray
- Color.magenta
- Color.orange
- Color.pink
- Color.red
- Color.white
- Color.yellow

Operatori aritmetici

Operatore	Uso	Descrizione
+	op1 + op2	Aggiunge *op1* e *op2*
-	op1 - op2	Sottrae *op2* da *op1*
*	op1 * op2	Moltiplica *op1* per *op2*
/	op1 / op2	Divide *op1* per *op2*
%	op1 % op2	Calcola il resto della divisione di *op1* per *op2*
++	op++	Incrementa op di 1; viene valutato al valore di *op* prima che venga incrementato
++	++op	Incrementa op di 1; viene valutato al valore di *op* dopo che è stato incrementato
--	op--	Decrementa op di 1; viene valutato al valore di *op* prima che venga decrementato
--	--op	Decrementa op di 1; viene valutato al valore di *op* dopo che è stato decrementato
+	+op	Promuove *op* a *int* se era un *byte*, uno *short* o un *char*
-	-op	Nega aritmeticamente *op*

Operatori condizionali

Operatore	Uso	Restituisce *true* se
>	op1 > op2	*op1* è maggiore di *op2*
>=	op1 >= op2	*op1* è maggiore di o uguale a *op2*
<	op1 < op2	*op1* è minore di *op2*
<=	op1 <= op2	*op1* è minore di o uguale a *op2*
==	op1 == op2	*op1* e *op2* sono uguali
!=	op1 != op2	*op1* e *op2* non sono uguali

Operatori relazionali

Operatore	Uso	Ritorna *true* se
&&	op1 && op2	*op1* e *op2* sono entrambi *true*
||	op1 || op2	o *op1* o *op2* è *true*
!	! op	*op* è *false*
&	op1 & op2	*op1* e *op2* sono entrambi *true*
|	op1 | op2	o *op1* o *op2* è *true*
^	op1 ^ op2	Se *op1* e *op2* sono diversi , cioè se uno o l'altro operatore è *true* ma non entrambi

Operatori di shift

Operatore	Uso	Operazione
>>	op1 >> op2	Fa scorrere i bit di *op1* a destra per una distanza di *op2*
<<	op1 << op2	Fa scorrere i bit di *op1* a sinistra per una distanza di *op2*
>>>	op1 >>> op2	Fa scorrere i bit di *op1* a destra per una distanza di *op2* (senza considerare il segno)

Operatori logici

Operatore	Uso	Operazione
&	op1 & op2	*and* tra bit
|	op1 | op2	*or* tra bit
^	op1 ^ op2	*xor* tra bit
~	~op2	Complemento

Operatori di assegnamento

Operatore	Uso	Equivale a
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2
^=	op1 ^= op2	op1 = op1 ^ op2
<<=	op1 <<= op2	op1 = op1 << op2
>>=	Op1 >>= op2	op1 = op1 >> op2
>>>=	Op1 >>>= op2	op1 = op1 >>> op2

Altri operatori

Operatore	Uso	Descrizione
?:	op1 ? op2 : op3	Se *op1* è *true*, ritorna *op2*. Altrimenti ritorna *op3*.
[]	*type* []	Dichiara un array di dimensione sconosciuta, che contiene elementi di tipo *type*.
[]	*type*[op1]	Crea un array con *op1* elementi. Deve essere utilizzato con l'operatore *new*.
[]	op1[op2]	Accede all'elemento di indice *op2* dell'array *op1*. Gli indici sono compresi tra 0 e la lunghezza dell'array meno uno.
.	op1.op2	Riferimento a *op2*, membro dell'oggetto *op1*.
()	op1(*params*)	Invoca il metodo *op1* con i parametri specificati. La lista dei parametri può essere vuota o contenere elementi separati dalla virgola.
(type)	(type) op1	Esegue il casting di *op1* verso il tipo *type*. Se il tipo è incompatibile viene generata un'eccezione.
new	**new** op1	Crea un nuovo oggetto o un array.
instanceof	op1 **instanceof** op2	Ritorna *true* se *op1* è un'istanza di *op2.*

Precedenze tra operatori

La seguente tabella mostra l'ordine di precedenza assegnato agli operatori. Gli operatori che sono nelle zone alte della tabella hanno una precedenza più alta. Gli operatori con alta precedenza sono valutati prima degli operatori con bassa precedenza. Sulla stessa riga sono presenti gli operatori che hanno la stessa precedenza.

Operatori postfissi	[] . (*params*) *expr*++ *expr*--
Operatori unari	++*expr* --*expr* +*expr* -*expr* ~ !
Creazione e casting	**new** (*type*)*expr*
Prodotti	* / %
Somme	+ -
Operatori di Shift	<< >> >>>
Operatori condizionali	< > <= >= **instanceof**
Operatori di uguaglianza	== !=
AND tra bit	&
OR esclusivo tra bit	^
OR tra bit	|
AND (operatore relazionale)	&&
OR (operatore relazionale)	||
Condizione	? :
Operatori di assegnamento	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Strutture di controllo

Iterazione

```
while (condizione)
{
  // Istruzioni
}
```

```
do
{
  // Istruzioni
}
while (condizione);
```

```
for (inizializzazione; terminazione; incremento)
{
  // Istruzioni
}
```

Selezione

```
if (condizione)
{
  // Istruzione
}
```

```
if (condizione)
{
  // Istruzione
}
else
{
  // Istruzione
}
```

```
if (condizione)
{
  // Istruzioni
}
else if (condizione)
{
  // Istruzioni
}
else if (condizione)
{
  // Istruzioni
}
else
{
  // Istruzioni
}
```

```
switch (espressione intera)
{
  case espressione intera:
    // Istruzioni
    break;
    ...
  default:
    // Istruzioni
    break;
}
```

Istruzioni di salto

Istruzione	Descrizione
break	Termina i seguenti costrutti: *switch*, *for*, *while* e *do-while*.
continue	Termina l'iterazione e valuta la condizione booleana che controlla il ciclo.
return *valore*	Termina il metodo ed eventualmente restituisce un valore al chiamante.

Struttura per la gestione delle eccezioni

```
try
{
  // Istruzioni da controllare
}
catch (tipoeccezione nome)
{
  // Istruzioni
}
catch (tipoeccezione nome)
{
  // Istruzioni
}
finally
{
  // Istruzioni
}
```

Eccezioni

Nome	Package	Descrizione
ArithmeticException	java.lang	Eccezione matematica, la più comune è la divisione per zero.
ClassCastException	java.lang	Indica che si è tentato di eseguire un casting di un oggetto verso una classe di cui non è istanza.
ClassNotFoundException	java.lang	L'applicazione sta caricando una classe ma non trova il file *.class*.
EOFException	java.io	È stata raggiunta la fine del file.
Exception	java.lang	Eccezione generica.
FileNotFoundException	java.io	Il file specificato non è presente.
IndexOutOfBoundsException	java.lang	Sono stati superati i limiti di un array.
InterruptedException	java.lang	Generata quando un processo viene interrotto da un altro.
IOException	java.io	Eccezione relativa alle operazioni di input/output.
MalformedURLException	java.net	Un URL non è stato definito correttamente.
NullPointerException	java.lang	Si cerca di usare un riferimento *null* dove sarebbe richiesto l'uso di un oggetto.
NumberFormatException	java.lang	Generata quando la conversione da un valore stringa ad un formato numerico non ha successo.
SQLException	java.sql	Eccezione relativa alle operazioni in SQL.

Android

Classi del package *android*

Manifest
R.anim
R.animator
R.array
R.attr
R.bool
R.color
R.dimen
R.drawable
R.fraction
R.id
R.integer
R.interpolator
R.layout
R.menu
R.mipmap
R.plurals
R.raw
R.string
R.style
R.styleable
R.xml

Classi del package *android.widget*

AbsListView
AbsoluteLayout
AbsSeekBar
AbsSpinner
AdapterView
AlphabetIndexer
AnalogClock
ArrayAdapter
AutoCompleteTextView
BaseAdapter
BaseExpandableListAdapter
Button
CalendarView
CheckBox
CheckedTextView
Chronometer
CompoundButton
CursorAdapter
CursorTreeAdapter
DatePicker
DialerFilter
DigitalClock
EdgeEffect
EditText
ExpandableListView
Filter
FrameLayout
Gallery
GridLayout
GridView
HeaderViewListAdapter
HorizontalScrollView
ImageButton
ImageSwitcher
ImageView
LinearLayout
ListPopupWindow
ListView
MediaController
MultiAutoCompleteTextView
NumberPicker
OverScroller
PopupMenu
PopupWindow
ProgressBar
QuickContactBadge
RadioButton
RadioGroup
RatingBar
RelativeLayout
RemoteViews
ResourceCursorAdapter
Scroller
ScrollView
SearchView
SeekBar
ShareActionProvider
SimpleAdapter
SimpleCursorAdapter
SimpleExpandableListAdapter
SlidingDrawer
Space
Spinner
StackView
Switch
TabHost
TableLayout
TableRow
TabWidget
TextClock
TextSwitcher
TextView
TimePicker
Toast
ToggleButton
TwoLineListItem
VideoView
ViewAnimator
ViewFlipper
ViewSwitcher
ZoomButton
ZoomControls

Linguaggio HTML

Struttura di una pagina HTML

```
<html>
<head>
<title>. . . . . . . . </title>
</head>
<body>
. . . . . . . . . .
</body>
</html>
```

Intestazioni

```
<h1> . . . </h1>
<h2> . . . </h2>
<h3> . . . </h3>
<h4> . . . </h4>
<h5> . . . </h5>
<h6> . . . </h6>
```

Formattazione del testo

```
<p> paragrafo </p>
<br />                (a capo)
<hr />                (linea orizzontale)
<b> grassetto </b>
<i> corsivo </i>
```

Link

```
<a href="http://. . . . . . . . . ">. . . . . . </a>
```

Tabelle

```
<table border="1">
<tr>
<th> . . . . . . </th>
<th> . . . . . . </th>
</tr>
<tr>
<td> . . . . . . </td>
<td> . . . . . . </td>
</tr>
</table>
```

Form

```
<form action="http://. . . . . . . . . . . . . . . . " method="post/get">
<input type="text" name=". . . ." value=". . . " size=". . .">
<input type="password">
<input type="checkbox" checked="checked">
<input type="radio" checked="checked">
<input type="submit">
<input type="reset">
<input type="hidden">

<select>
<option> . . . . .
<option selected> . . . . .
<option> . . . . .
</select>

<textarea name=". . . . . " rows=". . . ." cols=". . . .">
</textarea>

</form>
```

Commenti

```
<!--  . . . . . . . . . . . . . . . . . . . . --->
```

Linguaggio SQL

Comando / Clausola	Sintassi
ALTER TABLE (aggiunta colonna)	ALTER TABLE *NomeTabella* ADD *NomeColonna TipoDato*
ALTER TABLE (elimina colonna)	ALTER TABLE *NomeTabella* DROP COLUMN *NomeColonna*
AS (alias per colonna)	SELECT *NomeColonna* AS *NuovoNome* FROM *NomeTabella*
AS (alias per tabella)	SELECT *NomeColonna* FROM *NomeTabella* AS *NuovoNome*
BETWEEN	SELECT *NomeColonna1*, *NomeColonna2*, … FROM *NomeTabella* WHERE *NomeColonna* BETWEEN *valore1* AND *valore2*
CREATE DATABASE	CREATE DATABASE *NomeDatabase*
CREATE INDEX	CREATE INDEX *NomeIndice* ON *NomeTabella* (*NomeColonna*)
CREATE TABLE	CREATE TABLE *NomeTabella* (*NomeColonna1 TipoDato*, *NomeColonna2 TipoDato*, ...)
CREATE UNIQUE INDEX	CREATE UNIQUE INDEX *NomeIndice* ON *NomeTabella* (*NomeColonna*)
CREATE VIEW	CREATE VIEW NomeVista AS SELECT *NomeColonna1*, *NomeColonna2*, … FROM *NomeTabella* WHERE *condizione*
DELETE FROM	DELETE FROM *NomeTabella* (cancella l'intera tabella) DELETE FROM *NomeTabella* WHERE *condizione*
DROP DATABASE	DROP DATABASE *NomeDatabase*
DROP INDEX	DROP INDEX *NomeTabella.NomeIndice*
DROP TABLE	DROP TABLE *NomeTabella*

Comando / Clausola	Sintassi
GROUP BY	SELECT *NomeColonna1*,SUM(*NomeColonna2*) FROM *NomeTabella* GROUP BY *NomeColonna1*
HAVING	SELECT *NomeColonna1*,SUM(*NomeColonna2*) FROM *NomeTabella* GROUP BY *NomeColonna1* HAVING SUM(*NomeColonna2*) *condizione*
IN	SELECT *NomeColonna1*, *NomeColonna2*, … FROM *NomeTabella* WHERE *NomeColonna* IN (*valore1*,*valore2*,..)
INSERT INTO	INSERT INTO *NomeTabella* VALUES (*valore1*,*valore2*,..) INSERT INTO *NomeTabella* (*NomeColonna1*, *NomeColonna2*,...) VALUES (*valore1*,*valore2*,..)
LIKE	SELECT *NomeColonna1*, *NomeColonna2*, … FROM *NomeTabella* WHERE *NomeColonna* LIKE *ValoreCampione*
ORDER BY	SELECT *NomeColonna1*, *NomeColonna2*, … FROM *NomeTabella* ORDER BY *NomeColonna* [ASC | DESC]
SELECT	SELECT *NomeColonna1*, *NomeColonna2*, … FROM NomeTabella
SELECT *	SELECT* FROM *NomeTabella*
SELECT DISTINCT	SELECT DISTINCT *NomeColonna1*, *NomeColonna2*, … FROM *NomeTabella*
SELECT INTO	SELECT *NomeColonna1*, *NomeColonna2*, … INTO *NuovaTabella* FROM *NomeTabella*
UPDATE	UPDATE *NomeTabella* SET *NomeColonna* =*NuovoValore* WHERE *NomeColonna* = *Valore*
WHERE	SELECT *NomeColonna1*, *NomeColonna2*, … FROM *NomeTabella* WHERE *condizione*

Glossario

A

Accesso (in generale), le operazioni di lettura, scrittura e aggiornamento delle informazioni registrate in una memoria.

ADSL (Asymmetrical Digital Subscriber Line) linea telefonica pubblica ad alta velocità che opera attraverso il tradizionale doppino telefonico. Si chiama asimmetrica perché utilizza velocità diverse di trasferimento dei dati, cioè i dati provenienti da Internet sono più veloci di quelli inoltrati verso Internet.

Aggiornamento modifica dei dati o di parte di essi contenuti in un archivio.

Algebra di Boole algebra delle proposizioni che possono essere vere o false, sviluppata dal matematico inglese George Boole; costituisce il principio di funzionamento logico del computer.

Algoritmo soluzione di un problema, sequenza di istruzioni assegnate ad un esecutore di cui si conoscono le capacità.

Allocazione il processo con il quale le zone di memoria centrale o le aree del disco vengono assegnate per la registrazione di dati o istruzioni.

Ambiente insieme delle risorse e dei vincoli per l'utente che lavora con un sistema di elaborazione.

Ampiezza di banda gamma delle frequenze di trasmissione utilizzate sulle reti di comunicazione.

Analisi studio di un problema con individuazione degli elementi caratterizzanti e dei risultati che si devono ottenere sulla base delle esigenze espresse dal committente; progettazione della soluzione automatizzata del problema.

Analogico la rappresentazione di quantità numeriche attraverso variabili fisiche, quali la traslazione, la rotazione, il voltaggio oppure la resistenza; in contrapposizione con digitale.

Àncora destinazione di un collegamento in un ipertesto.

AND operatore logico che rappresenta la congiunzione di due proposizioni; il risultato è una proposizione vera soltanto se entrambe le proposizioni sono vere.

ANSI (American National Standards Institute) Ente degli Stati Uniti che è responsabile degli standard nazionali.

Applet programma scritto in linguaggio Java e inserito all'interno della rete Internet per applicazioni interattive e multimediali.

Applicazione (in generale) un'attività, un lavoro, o la soluzione automatizzata di un problema, risolte attraverso l'uso di apparecchiature informatiche.

Architettura la modalità di organizzazione delle risorse hardware e software all'interno di un sistema di elaborazione.

Archivio insieme di dati organizzati legati tra loro da un nesso logico e che possono essere registrati su un supporto di memoria.

Area sensibile area di un'immagine definita graficamente che contiene un collegamento ipertestuale. Un'immagine contenente aree sensibili viene definita mappa immagine.

ASCII (American Standard Code for Information Interchange) codice binario a 7 bit (nella versione ASCII esteso, 8 bit) usato per rappresentare lettere, numeri, segni di punteggiatura e caratteri speciali all'interno dei computer.

Asincrono controllo delle attività del computer nelle quali una specifica operazione viene lanciata in esecuzione non appena viene ricevuto il segnale che l'operazione precedente è stata completata e quindi secondo tempi non predefiniti.

ASP (Active Server Page) linguaggio per la creazione di script da eseguire su un server Web.

Associazione (relationship) legame che stabilisce un'interazione tra le entità in un modello di dati.

Automazione la ricerca, la progettazione, lo sviluppo, l'applicazione e i metodi che riguardano i processi e le macchine che sono in grado di elaborare dati, di eseguire attività o di muoversi senza l'intervento dell'uomo.

B

Backbone (dorsale) rete fisica di comunicazione ad alta velocità per connettere reti regionali.

Background un tipo di esecuzione dove l'utente chiede alla shell del sistema operativo di eseguire un comando in modo non interattivo, consentendo nel frattempo di inserire altri comandi dalla tastiera.

Backup la copia di sicurezza di file, directory o interi supporti su altre unità di memoria di massa.

Banca Dati concentrazione e organizzazione di una grande quantità di dati, riguardanti uno specifico argomento o settore, presso Aziende di servizi che sono in grado di mettere a disposizione le informazioni su computer (host) per gli utenti che si collegano ad essi con apparecchiature di telecomunicazione.

Banda larga canale di comunicazione ad ampia capacità che permette un accesso rapido ed agile ai sistemi di informazione e di comunicazione.

Batch tecnica di elaborazione da parte di un sistema operativo che organizza i lavori in sequenza e li esegue a lotti.

Binario una proprietà caratterizzata da selezione, scelta, condizione oppure valore nella quale sono presenti solo due alternative possibili.

BIOS (Basic Input-Output System) il software già presente nel computer che controlla le operazioni di I/O (input-output) al livello più vicino all'hardware; per esempio in un Personal Computer il BIOS è il codice memorizzato nella ROM che consente al computer di interfacciarsi con la tastiera, il video, i dischi e di caricare il sistema operativo al momento del bootstrap.

Bit abbreviazione di binary digit, cioè cifra binaria, la cifra 0 oppure la cifra 1. Unità di misura dell'informazione.

Bitmap formato standard per le immagini (BMP).

Blocco (block) unità fisica di registrazione su un supporto di memoria di massa.

Blog contrazione di web log (letteralmente, diario di bordo sul Web): spazio creato su server Internet per raccogliere e condividere qualsiasi informazione interessante per altre persone o gruppi.

Bluetooth nome comune per intendere lo standard IEEE 802.15 per la comunicazione senza cavi, (wire-less) che utilizza nello spettro 2400MHz-2483MHz una tecnica di modulazione a diffusione di spettro detta FHSS, tra computer, cellulari e palmari.

Bookmark (segnalibro) indirizzo Web di frequente consultazione memorizzato dall'utente come sito di particolare interesse.

Bootstrap (o **boot**) l'operazione che il computer compie all'accensione eseguendo un piccolo programma che consente al sistema operativo di caricare se stesso in memoria centrale.

BPI (Bits per inch) misura della densità di registrazione su un supporto di memoria di massa.

Browser programma che consente di visualizzare pagine in formato grafico provenienti da computer collegati nella rete Internet.

Buffer un'area della memoria RAM riservata per registrazioni temporanee di dati che sono stati acquisiti da una periferica o che sono pronti per essere inviati a un'unità disco o ad altra periferica.

Bug (letteralmente significa cimice) errore nel software che impedisce il funzionamento corretto di un programma.

Bus la parte di un chip, di una scheda o di un'interfaccia che consente il passaggio di dati.

Byte l'insieme bit che codifica un carattere, di solito 8 bit.

C

Cache una memoria di tipo RAM utilizzata per trasferire dati tra due unità di un sistema che operano a velocità diverse, per esempio il disco e la memoria centrale, oppure la memoria centrale e la CPU.

CAD (Computer Aided Design) i prodotti software e hardware che formano sistemi dedicati alla progettazione assistita dal computer.

Campo parte di un record contenente un dato; in generale spazio riservato a contenere un dato (per esempio, campo di stampa è lo spazio all'interno di una riga di stampa riservato ad un'informazione).

Canale parte di un sistema di comunicazione che connette una sorgente ad una o più destinazioni.

Capacità la quantità di informazioni che può essere memorizzata in una memoria centrale o in un supporto di memoria di massa; è espressa in Kbyte, Mbyte, Gbyte o Tbyte.

Carattere jolly carattere che sta al posto di altri caratteri nella rappresentazione dei modelli di stringa nelle ricerche o nei criteri delle query in un database (per esempio i caratteri * e ?).

Cartella area della memoria di massa contenente file e altre cartelle (termine usato in Windows per indicare la directory).

Cartridge nastro magnetico utilizzato come memoria di massa o come supporto per il backup.

Case-sensitive proprietà dei linguaggi di programmazione e di comando che distinguono i caratteri minuscoli dai maiuscoli.

Cavallo di Troia virus all'apparenza innocuo che riesce a superare le protezioni antivirus perché è contenuto in un programma che viene autorizzato dall'utente che lo sta installando.

CD-R (Compact Disc-Recordable) disco ottico di tipo WORM che può essere scritto con speciali apparecchiature di registrazione e può poi essere letto con un normale lettore di CD.

CDROM (Compact Disc - Read-Only Memory) un disco, simile ai CD musicali, che contiene dati e dal quale si può solo leggere.

CD-RW (CD Recordable Rewritable) disco ottico sul quale si può leggere e scrivere più volte come su un normale dischetto.

Certificato digitale codice assegnato ad una persona che vuole fare transazioni sulla rete Internet, in modo che ne venga riconosciuta in modo inequivocabile l'identità.

Chat comunicazione sincrona tra diverse persone che utilizzano il servizio IRC (Internet Relay Chat) di una rete, particolarmente utilizzato in Internet.

Chiave (key) l'insieme di uno o più attributi che consentono di distinguere le istanze di una stessa entità. Campo di un record che lo identifica univocamente.

Cilindro insieme di tutte le tracce di un disco che si trovano alla stessa distanza dal centro.

Client un programma o un computer messo in comunicazione con un altro che fornisce servizi o risorse per l'elaborazione dei dati.

Client/Server modello centralizzato di amministrazione delle risorse, di controllo degli accessi e di autenticazione degli utenti che hanno accesso ad una rete di calcolatori.

ClipArt raccolta di immagini, fotografie e oggetti multimediali (suoni, video) che si possono inserire all'interno di documenti o di pagine Web.

Clock dispositivo elettronico per la sincronizzazione delle attività di un processore.

Cluster un gruppo di settori di un disco che può essere indirizzato come un'unità logica dal file system.

Coda una struttura di elementi nella quale il primo entrato è il primo ad essere elaborato (struttura FIFO, First In First Out).

Codice sinonimo di software e in generale di programma che un computer è in grado di comprendere ed eseguire.

Codifica la lista delle operazioni richieste a un computer, espresse usando un codice o uno pseudo-codice; il termine viene usato anche per indicare la traduzione di un algoritmo in un programma per il computer.

Collegamento ipertestuale collegamento che consente di passare da un testo o da una mappa immagine a una pagina o ad altro tipo di file in un ipertesto o sul World Wide Web.

Comando il segnale, o l'insieme di segnali, che avvia l'esecuzione di un'attività del computer; il termine viene usato anche come sinonimo di istruzione.

Commutazione modalità di instradamento delle comunicazioni sulle reti.

Compilatore programma di traduzione del testo sorgente di un programma, scritto con un linguaggio di programmazione, in un programma oggetto.

Compressione algoritmo di trasformazione del contenuto di un file in modo da minimizzare lo spazio di occupazione su un disco: quando deve essere utilizzato, il file viene poi sottoposto al procedimento di decompressione; la compressione risulta utile per fornire il software usando pochi dischi e per diminuire i tempi di trasferimento dei file da un computer a un altro usando le linee telefoniche.

Comunità virtuale siti di reti sociali, forum, chat, giochi informatici in rete, pubblicazione di fotografie o di clip video, per interagire e collaborare in modo attivo con altri utenti e condividere documenti.

Concentratore dispositivo della telematica che concentra il traffico proveniente da molti canali a bassa velocità verso canali ad alta velocità.

Configurazione organizzazione dei componenti di un sistema di elaborazione.

Controller apparecchiatura elettronica avente la funzione di controllare il funzionamento di una periferica consentendo il suo collegamento con un computer.

Cookie informazione proveniente da un server Web e memorizzata sul disco dell'utente, per poter essere riutilizzata dallo stesso server in momenti successivi.

Coprocessore Processore aggiuntivo che estende il set di istruzioni della CPU aumentandone le prestazioni (per esempio, coprocessore matematico).

CPU (Central Processing Unit) il microprocessore che governa il processo di elaborazione in un computer.

Crittoanalisi tecnica che si occupa di trovare metodi per decifrare documenti cifrati senza conoscere a priori la chiave di cifratura.

Crittografia tecnica per trovare algoritmi che permettono una trasmissione, segreta, integra, autentica e non ripudiabile, di dati in una rete pubblica.

D

Data Base insieme di archivi di dati organizzati in modo integrato per facilitare la gestione e il ritrovamento dei dati in essi contenuti, evitando le duplicazioni dei dati.

Debugging (letteralmente spulciare) eliminare gli errori logici (bug), mettere a punto un programma o una procedura.

Default (letteralmente mancanza) un valore assegnato automaticamente o una condizione che sussiste a meno (in mancanza) di una dichiarazione diversa esplicitamente espressa dall'utente.

Device dispositivo periferico collegato all'unità centrale di un sistema di elaborazione.

Diagnostica individuazione delle cause di situazione di errori o guasti in prodotti hardware o software con indicazione di eventuali messaggi di errore sul video del terminale.

Dialer programma che attiva la chiamata a un server Internet tramite le linee telefoniche.

Digital divide rischio che alcuni individui, gruppi o collettività possano rimanere esclusi dalla società dell'informazione.

Digitale l'utilizzo di numeri per rappresentare le quantità che caratterizzano un problema o un calcolo.

Directory organizzazione e raggruppamento di file e altre sottodirectory su una memoria di massa (disco) gestita dal file system del sistema operativo.

Disco supporto di memoria di massa che consente la registrazione di dati sulla sua superficie, con tecnologie di tipo magnetico od ottico.

Disk Controller circuiti elettronici che sono in grado di convertire i dati del computer e i comandi in una forma utilizzabile da un disco.

DLL (Dynamic Linked Library) un insieme di funzioni e procedure, comuni a più applicazioni, che possono essere caricate e scaricate dai programmi quando servono.

DNS (Domain Name System) servizio distribuito per l'assegnazione dei nomi nella rete Internet.

Dominio suddivisione della rete Internet che corrisponde ad una Nazione oppure ad Enti o società; i domini sono organizzati nella rete Internet in modo gerarchico.

Download operazione che, nel corso di un collegamento ad un host computer per l'esecuzione di una ricerca in linea, permette di registrare sulla memoria di massa del proprio computer i risultati della ricerca o la copia di un file.

Downsizing decentramento della potenza elaborativa da sistemi di grandi dimensioni verso stazioni di lavoro di prestazioni elevate oppure verso sistemi distribuiti.

Drive dispositivo hardware per la gestione di supporti di memoria di massa.

Driver programma software che consente di collegare una periferica con il computer e di utilizzarla all'interno di un'applicazione dell'utente.

E

e-Business compravendita di beni e servizi attraverso Internet tra aziende.

e-Commerce compravendita di beni e servizi attraverso Internet anche tra aziende e privati.

e-Government utilizzo delle tecnologie dell'informazione e della comunicazione per rendere più efficienti i rapporti e le transazioni tra Pubblica Amministrazione e cittadini o aziende.

e-Learning uso di nuove tecnologie multimediali e di Internet per migliorare la qualità dell'apprendimento mediante l'accesso a risorse e servizi e a collaborazioni e interscambi a grande distanza.

e-mail (Electronic Mail) metodo di comunicazione elettronica che sfrutta le reti di computer.

Editing attività di modifica e trattamento di un testo o di un programma sorgente utilizzando un apposito programma di utilità (editor).

Elaborazione trattamento di informazioni acquisite dall'esterno per restituire risultati all'esterno.

Elettronico riferito alle applicazioni basate sull'emissione e il comportamento delle correnti di elettroni, nel vuoto, in un gas oppure in conduttori o semiconduttori; diverso da elettrico che si riferisce al flusso di correnti nei fili o nei conduttori convenzionali.

emoticon uso di caratteri, segni grafici e icone per esprimere emozioni nei messaggi sui cellulari o nella posta elettronica.

Entità oggetto (concreto o astratto) che ha un significato anche quando viene considerato in modo isolato ed è di interesse per la realtà che si vuole modellare.

Esadecimale sistema di rappresentazione di un numero che utilizza 16 cifre.

Ethernet nome con il quale si indica lo standard IEEE 802.3 di rete locale che opera a 10/100 Mbps e che utilizza come metodo di accesso il CSMA/CD.

EULA (contratto con l'utente finale) permesso per l'uso di un prodotto software che l'utente ottiene con l'acquisto.

Evento qualcosa che accade durante il processo di elaborazione per effetto di un'azione dell'utente o di un'attività del sistema operativo.

Extranet uso di parti della rete aziendale (Intranet) per utenti selezionati esterni all'azienda.

F

Facsimile (abbreviato con fax) un metodo per trasmettere l'immagine di una pagina da un punto all'altro collegati da linee telefoniche.

FAQ (Frequently Asked Questions) documento Web contenente un elenco di domande e risposte su argomenti comuni.

Fault tolerance tecniche che consentono di non interrompere l'attività di elaborazione nel caso si verifichino guasti al sistema o a parti di esso, attivando automaticamente sistemi, dispositivi e risorse alternative.

Feed RSS flusso di dati RSS visualizzabile con programmi detti feed reader (lettori di feed) o aggregatori di notizie.

Fibra ottica fibra di materiale vetroso usata per la trasmissione a grandi velocità dei segnali di comunicazione generati da una fonte luminosa.

Field parte di un record individuabile come dato (traduzione inglese di campo).

File (in generale) tutto ciò che può essere memorizzato su un supporto di memoria di massa (testo, programma, comando del sistema operativo, archivio di dati).

File System struttura dati implementata da un sistema operativo per memorizzare, organizzare e gestire file su memoria di massa.

Firewall sistema di protezione di una rete locale o di un personal computer che cerca di impedire l'accesso ad utenti esterni non autorizzati.

Firma digitale codice associato ad un documento che permette di riconoscere l'entità dell'autore.

Firmware il software che la Casa costruttrice del computer programma all'interno del sistema hardware.

Floating Point rappresentazione dei numeri reali secondo la notazione esponenziale o in virgola mobile.

Flow-Chart (diagramma di flusso) linguaggio grafico utilizzato per rappresentare algoritmi.

Focus proprietà assunta da un oggetto dell'interfaccia grafica quando l'utente fa clic su di esso, rendendolo attivo per l'interazione.

Foglio elettronico (spreadsheet) software orientato alla gestione di tabelle di calcolo.

Font serie completa di caratteri utilizzabili per scrivere un testo.

Form (modulo) insieme di campi di una pagina Web per l'immissione di dati.

Formattazione procedimento con il quale un disco viene preparato per la registrazione dei dati in tracce e settori; questa operazione cancella tutti i dati eventualmente già registrati sul disco.

Forum spazi Web per dibattiti su un argomento di interesse generale; possono essere visitati per leggere i messaggi lasciati dagli altri o per lasciare il proprio commento.

Frame sottofinestra all'interno di una pagina Web, identificata con un nome.

Freeware software utilizzabile e distribuibile liberamente senza vincoli nè pagamenti.

FTP (File Transfer Protocol) servizio di rete per trasferire file da un computer a un altro.

G

Gateway dispositivo che connette due reti che hanno architetture diverse, convertendo i protocolli dell'una in quelli dell'altra. Indica anche i router nell'architettura di rete Internet Protocol Suite.

GIF (Graphics Interchange Format) un tipo standard nel formato dei file contenenti immagini.

GIF animata file contenente una serie di immagini GIF visualizzate in sequenza rapida dal browser, in modo da ottenere un effetto animato.

Gigabyte (GB) unità di misura della capacità di memoria corrispondente a un miliardo di byte (esattamente 1024 Mbyte).

GUI (Graphical User Interface) programma che presenta in forma grafica con finestre e icone le funzioni disponibili di un ambiente operativo (per esempio Windows).

H

Hacker utente non autorizzato (pirata) che riesce ad accedere a un sistema.

Hard copy trasferimento di dati su supporto cartaceo.

Hard disk (disco fisso) disco rigido di tipo magnetico sul quale vengono memorizzati i dati.

Hardware le apparecchiature meccaniche, magnetiche, elettroniche ed ottiche che formano la parte fisica di un sistema di elaborazione.

Help messaggi di poche righe o di intere videate che forniscono all'utente di un'applicazione informazioni utili per proseguire correttamente nel lavoro; le informazioni sono richiamabili in ogni momento; l'aiuto può essere tutoriale se fornisce spiegazioni di carattere generale sul software in uso, contestuale se le informazioni sono relative alla fase del lavoro nella quale è stato richiesto l'aiuto.

Hertz (Hz) unità di misura della frequenza corrispondente a un ciclo per secondo.

Home page la pagina di apertura di un sito WWW in Internet.

Host (in generale) un computer al quale si possono collegare altri computer o terminali.

HTML (HyperText Markup Language) linguaggio di formazione di una pagina con marcatori per documenti WWW.

HTTP (HyperText Transfer Protocol) protocollo di rete utilizzato in Internet dalla tecnologia WWW.

Hub dispositivo attivo per reti locali che collega nodi appartenenti allo stesso dominio di collisione ricoprendo il ruolo di ripetitore multiporta.

I

Icona rappresentazione sul video di un programma o di una routine del sistema operativo attraverso un simbolo grafico che ne richiama la funzione, attivabile con il mouse.

Implementazione trasferimento su computer di un procedimento applicativo attraverso l'uso di programmi software.

Indirizzo (address) un numero o un insieme di caratteri che identifica un registro, una locazione di memoria oppure una periferica.

Indirizzo IP metodo standard per identificare una connessione in una rete che implementa l'architettura TCP/IP. La versione V4 utilizza per la rappresentazione 4 numeri naturali, compresi tra 0 e 255, separati dal punto. Esempio 201.173.54.34.

Inizializzare assegnare un valore iniziale alle variabili di un programma.

Input ingresso di dati in un processo di elaborazione.

Instant Messaging (IM) sistema di comunicazione sincrona per lo scambio di messaggi o brevi testi tra due persone connesse ad Internet in quel momento e che stanno utilizzando a loro volta un programma di messaggistica istantanea.

Integrazione su larga scala (VLSI, Very Large Scale Integration) in microelettronica indica l'integrazione di un grandissimo numero di componenti elettronici e di posizioni di memoria su un unico chip.

Integrità garanzia della consistenza dei dati contenuti in un database, rispetto alle operazioni di manipolazione che vengono effettuate dagli utenti.

Intelligenza artificiale la possibilità che i computer sviluppino processi di apprendimento e che producano prestazioni sulla base di un'esperienza acquisita, secondo meccanismi tipici della mente umana.

Interattivo modalità di comunicazione tra sistema di elaborazione e utente che consente di ottenere messaggi di risposta e di fornire i dati di input durante l'esecuzione del programma; il termine viene usato in contrapposizione a batch.

Interfaccia dispositivo hardware che consente lo scambio di dati tra l'unità centrale di un computer e una periferica; dal punto di vista software il termine interfaccia indica un programma che consente di utilizzare in modo facile le funzioni di un sistema operativo o di un programma applicativo.

Interfacciare mettere in comunicazione tra loro due sistemi, oppure un sistema e una periferica, oppure due applicazioni software.

Internet un sistema di reti di computer distribuito su tutto il pianeta.

Interprete programma che traduce, durante l'esecuzione, istruzione per istruzione un programma scritto con un linguaggio di programmazione.

Interrogazione richiesta dell'utente per ritrovare i dati contenuti negli archivi.

Interrupt (interruzione) segnalazione al processore che è accaduto un evento di più alta priorità e deve essere servito; la normale sequenza degli eventi viene temporaneamente sospesa.

Intranet rete locale che implementa la stessa architettura e gli stessi protocolli di Internet fornendo gli stessi servizi.

Ipertesto organizzazione non lineare di testi, grafici, immagini e documenti collegati tra loro attraverso richiami non sequenziali stabiliti in precedenza o definiti dall'utente durante il percorso di ritrovamento dei dati (navigazione).

Istruzione un insieme di caratteri che definisce un'operazione su dati o indirizzi di dati e che causa un'attività del computer.

J

Java linguaggio di programmazione particolarmente utilizzato per le applicazioni Web.

JavaScript linguaggio procedurale standard per il WWW, il cui codice viene inserito direttamente nella pagina HTML e interpretato dal browser.

JPEG (Joint Photographic Experts Group) un tipo standard nel formato di compressione dei file contenenti immagine fotografiche.

K

Key (chiave) campo di un record che identifica univocamente il record e che serve per accedere ai record contenuti in un archivio indicizzato.

Keyboard la tastiera di un computer, utilizzata come unità di input per introdurre dati o comandi.

Kilobyte (KB) unità di misura della capacità di memoria corrispondente a mille byte (esattamente 1024 byte).

L

LAN (Local Area Network), rete locale che non si può estendere oltre un comprensorio in pratica non può attraversare suolo pubblico.

Laser dispositivo elettronico in grado di produrre luce in forma di impulsi rapidi o di flussi intensi.

Layout disposizione del testo e degli oggetti grafici in una videata o in una pagina stampata.

Lettura (Read) il trasferimento di dati dalla memoria di massa, o da un'altra periferica, alla memoria centrale.

Libreria insieme organizzato di programmi utilizzati frequentemente nelle applicazioni e memorizzati su una memoria di massa.

Linguaggio codice utilizzato per scrivere i programmi per l'elaboratore, oppure, in generale, insieme di parole chiave utilizzate per comunicare istruzioni e comandi al computer.

Linguaggio macchina le operazioni che sono comprensibili al computer al livello dei circuiti hardware.

Link parola, frase, immagine o altro oggetto di un ipertesto, evidenziata con diverso colore e con la sottolineatura, che consente in qualunque momento di saltare ad un'altra pagina che contiene l'informazione desiderata e che risiede sullo stesso o su altri computer.

Linker programma di sistema che consente di collegare al programma oggetto dell'utente routine di sistema e personali per generare codice eseguibile.

Linux sistema operativo sviluppato da Linus Torvalds e reso disponibile in modo gratuito per utenti e sviluppatori, per computer personali e server della rete Internet.

Login fase iniziale del lavoro al computer nella quale l'utente si identifica fornendo anche la parola d'ordine (password) che viene controllata dal computer.

Logoff (o logout) termine usato per descrivere la fine della connessione di un utente con un host computer.

Loop situazione di errore nella quale un programma ripete continuamente la stessa sequenza di operazioni senza poter raggiungere la fine.

M

Macroistruzione (o più semplicemente macro) comando simbolico che richiama l'esecuzione di una sequenza di operazioni elementari frequentemente utilizzate.

Mainframe sistema di elaborazione di dimensioni medie o grandi in grado di gestire il collegamento contemporaneo di un elevato numero di terminali e di elaborare grandi quantità di dati con memorie di massa di capacità molto elevate.

Malware software creato per scopi malevoli ed eseguito all'insaputa dell'utente.

Manutenzione intervento di miglioramento o eliminazione di guasti nell'hardware di un sistema di elaborazione; nell'ambito del software il termine indica i processi di modifica, aggiornamento o ampliamento dei programmi di base o applicativi esistenti, richiesti da esigenze aziendali o da nuove normative.

Mappa immagine immagine di una pagina Web contenente una o più aree sensibili al clic del mouse, alle quali sono stati assegnati dei collegamenti ipertestuali.

Maschera presentazione dei dati contenuti in un database e loro aggiornamento (inserimento, modifica e cancellazione) usando moduli in finestre grafiche.

Megabyte (MB) unità di misura della capacità di memoria corrispondente al milione di byte (esattamente 1024 Kbyte).

Megahertz (Mhz) unità di misura della frequenza corrispondente al milione di cicli per secondo.

Memoria (storage) un dispositivo che gestisce unità di informazioni nelle quali è possibile registrare o prelevare dati.

Memoria di massa supporto magnetico od ottico di grandi capacità, esterno rispetto all'unità centrale, destinato a contenere programmi e archivi di dati in modo permanente.

Menu elenco di funzioni o lavori che un programma è in grado di svolgere, all'interno del quale l'utente può scegliere.

Metodo termine utilizzato nella programmazione OO (Object-Oriented) per indicare una funzione.

Microprocessore un chip di circuiti integrati che controlla il processo di elaborazione di un computer.

Microsecondo (µs) milionesimo di secondo (0,000001 sec).

Millisecondo (ms) il millesimo di secondo (0.001 sec.).

Minicomputer sistema di elaborazione di medie dimensioni in grado di servire più terminali, di livello più alto rispetto al Personal Computer ma più limitato rispetto al mainframe.

Mirroring duplicazione di parti degli archivi registrati sui supporti di memoria di massa con scopi di sicurezza.

Modem (modulatore/demodulatore) un dispositivo che trasmette e riceve dati per il computer con le linee telefoniche, attraverso la trasformazione dei segnali digitali, utilizzati all'interno del computer, in segnali analogici che possono viaggiare sulle linee telefoniche (modulazione), e la trasformazione inversa al ricevimento dei dati da un altro computer (demodulazione).

Modulo (in inglese form) insieme di campi di una pagina Web per l'immissione di dati.

Monitor sinonimo di video o display.

Motore di ricerca server della rete Internet che consente di individuare siti di interesse per l'utente, attraverso la specificazione di criteri di ricerca.

Mouse (letteralmente topo) dispositivo elettronico che controlla il movimento del cursore sul video in corrispondenza del movimento del dispositivo da parte della mano dell'utente su una superficie piana.

MP3 standard nel formato di compressione di file contenenti suoni e motivi musicali.

MS-DOS sistema operativo della Microsoft, dedicato, interattivo e monoutente; è stato il sistema operativo standard de facto per i Personal Computer.

Multimedialità insieme di strumenti diversi (media), hardware e software, per gestire in modo integrato testi, immagini fisse e in movimento, suoni, utilizzando il computer.

Multiprogrammazione metodo di allocazione di più programmi contemporaneamente nella memoria centrale di elaborazione.

Multitasking capacità di un sistema di elaborazione di eseguire più lavori contemporaneamente.

Multiutente sistema di elaborazione in grado di servire più utenti nello stesso tempo.

N

Nanosecondo (ns) unità di misura del tempo che corrisponde a un miliardesimo di secondo.

Nastro magnetico (tape) nastro di materiale magnetizzabile utilizzato come supporto di memoria di massa.

Navigazione possibilità di muoversi all'interno di un insieme organizzato di archivi alla ricerca dei dati richiesti secondo un percorso deciso dall'utente e non necessariamente predefinito.

Netiquette regole per il comportamento corretto in Internet.

Network rete composta da nodi e linee di collegamento tra i computer di utenti diversi che si trovano anche a grandi distanze.

Nodo un punto della rete dove le linee di comunicazione sono interconnesse.

Notebook personal computer portatile delle dimensioni di un quaderno.

O

Object-Oriented (orientato agli oggetti) modalità di lavoro informatico basate sull'uso di oggetti.

OCR (Optical Character Reader o Recognition) dispositivo di riconoscimento dei caratteri a partire da documenti stampati o scritti.

Off-line un sistema che opera in modalità di non connessione con un computer host.

OLE (Object Linking and Embedding) collegamento e incorporamento di oggetti: tecnologia Microsoft che consente di utilizzare all'interno di un documento oggetti creati con un'altra applicazione Windows.

On-line una risorsa (periferica, banca dati, altro computer) collegata ad un computer e pronta per essere utilizzata.

Open Source (in generale) software il cui codice sorgente è reso gratuito e disponibile per l'uso e la modifica di utenti o altri programmatori; di solito è il frutto di collaborazione tra gruppi di sviluppatori che poi rendono liberamente utilizzabile il risultato del loro lavoro.

OR operatore logico che rappresenta la disgiunzione di due proposizioni; il risultato è una proposizione vera se almeno una delle due proposizioni è vera.

Ottale sistema per rappresentare numeri che utilizza 8 cifre.

Output le informazioni trasferite dalla memoria centrale del computer ad una memoria secondaria o ad un'altra periferica.

Outsourcing modalità organizzativa nella gestione aziendale con la quale un'azienda si appoggia a un'altra azienda per ottenere servizi o lavori che non è in grado di realizzare al proprio interno.

Overflow il risultato di un calcolo aritmetico superiore al valore massimo trattabile dal computer; in generale, superamento dei limiti imposti dal sistema o dal programma.

P

Pagina singolo documento in un sito Internet creato utilizzando il linguaggio HTML o le tecnologie Web.

Parallelo computer con più processori che lavorano per risolvere un problema simultaneamente.

Parità metodo per l'individuazione dell'errore che controlla la validità dei caratteri trasmessi.

Parola (word) un insieme di caratteri che occupano una locazione di memoria e che vengono trattati e trasferiti dai circuiti del computer come un pacchetto di informazione, contenente un'istruzione per l'unità di controllo o una quantità per l'unità aritmetico-logica.

Partizione una porzione del disco dedicata ad un particolare sistema operativo o ad una applicazione.

Password parola d'ordine necessaria per accedere alle risorse di un sistema di elaborazione, oppure ai programmi o agli archivi di dati protetti da accessi non autorizzati.

Pathname nome di un file descritto indicando il cammino (path) da percorrere nella struttura gerarchica ad albero delle directory per ritrovare il file.

Peer to peer rete di computer semplice, senza un particolare server, nella quale gli utenti condividono le risorse di rete e spazio su disco, con livelli di sicurezza non elevati.

Periferica (device) un dispositivo collegato all'unità centrale di un sistema di elaborazione, come l'unità a dischi o la stampante.

Personal computer piccolo sistema di elaborazione con sistema operativo dedicato e monoutente, orientato all'informatica individuale.

Phishing truffa ai danni degli utenti di Internet con lo scopo di ottenere dati personali e privati, per esempio il numero di carta di credito o il codice di accesso a un conto bancario o postale.

PHP linguaggio per la creazione di script da eseguire su un server Web.

Piattaforma l'hardware di base o il software di base sui quali vengono aggiunti i moduli necessari per far funzionare le diverse applicazioni.

Pila (in inglese stack) una struttura di elementi nella quale l'ultimo entrato è il primo ad essere elaborato (struttura LIFO, Last In First Out).

Pixel (picture element) elemento di immagine dal punto di vista grafico; il termine indica anche il punto del video che può essere acceso o spento.

Plotter unità di output dedicata al tracciamento di grafici.

Plug-in programmi che ampliano le funzionalità del browser per poter utilizzare documenti multimediali presenti nei siti Web.

Podcast file, normalmente musicale o video, messo a disposizione degli utenti Internet che possono scaricarlo liberamente e in modo automatico attraverso un programma di podcasting.

POP (Post Office Protocol) protocollo utilizzato nell'architettura TCP/IP per ricevere e-mail.

Popup finestra di piccole dimensioni che viene aperta sullo schermo, indipendentemente dalle azioni dell'utente, quando si usa un browser per la visualizzazione di pagine Web.

Porta canale di comunicazione tra un terminale, o una periferica, e l'unità centrale di un elaboratore.

Porta (nelle reti) connessione logica a un computer dall'esterno tramite un programma software o un servizio Internet.

Portabilità un sistema operativo o in generale un'applicazione che è in grado di funzionare su diverse piattaforme hardware.

Portale un sito Web che consente l'accesso guidato ad altri siti Internet, attraverso motori di ricerca, indici, categorie e selezione di notizie.

POS (Point Of Sale) punto di vendita in un esercizio commerciale dotato di lettore di carta di credito.

Posta elettronica (e-mail) scambio di messaggi tra utenti attraverso i computer situati nei nodi di una rete di comunicazione.

Postscript linguaggio standard per la formazione della pagina in una stampante laser.

Precisione il numero delle cifre significative utilizzate per rappresentare una grandezza.

Priorità il livello di un processo che determina la sua preferenza nell'assegnazione delle risorse del sistema.

Procedura modulo di istruzioni che svolge una specifica funzione all'interno di un programma; il termine viene anche usato in generale per indicare un insieme di programmi che risolvono un problema aziendale in modo automatizzato.

Processo un programma che si trova in stato di esecuzione nella memoria di un computer.

Processore dispositivo che svolge i compiti di un'unità di elaborazione per lo svolgimento sistematico nel tempo di operazioni aventi una precisa finalità (processo).

Programma sequenza di istruzioni comprensibili ed eseguibili da un elaboratore.

Programma oggetto programma che si ottiene dalla traduzione del programma sorgente effettuata dal compilatore.

Programmatore figura professionale del settore dell'informatica specializzata nella preparazione di programmi per l'elaboratore, utilizzando le specifiche decise nella fase di analisi.

Prompt (letteralmente sollecito) simbolo evidenziato sul video dalla shell del sistema operativo per indicare che il sistema è pronto ad accettare una richiesta da parte dell'utente.

Protocollo regole comuni applicate alle procedure usate per lo scambio di informazioni tra entità cooperanti.

Provider organizzazione o società che fornisce connettività alla rete.

Proxy server Internet che regola il traffico tra una rete locale e Internet.

Puntatore piccolo disegno, di solito a forma di freccia, per indicare una scelta in un menu o un'icona in una finestra sul video.

Q

Query interrogazioni sui dati contenuti in un database; permettono di ricavare i dati dalle tabelle, estraendo i dati secondo i criteri scelti dall'utente.

R

RAID (Redundant Array of Independent Disks) sistema di organizzazione di dischi fissi per aumentare le prestazioni, per avere tolleranza ai guasti e per ottenere l'integrità dei dati in un sistema di storage.

RAM (Random Access Memory) memoria ad accesso costante che permette la scrittura e la lettura di dati.

Random accesso diretto alle posizioni di memoria dove registrare o leggere informazioni, in contrapposizione all'accesso sequenziale, nel quale la posizione dell'informazione dipende da quella dell'informazione precedente.

RFC è seguito da un numero progressivo ed indica un documento redatto e pubblicato con il relativo stato di proposta, in forma di bozza (draft), nella discussione sugli standard per la comunità Internet.

Record insieme di informazioni organizzate e legate fra loro da un nesso logico, facenti parte di un archivio di dati.

Recovery possibilità di ripristinare i dati nel caso di interruzione anomala del lavoro del sistema di elaborazione.

Registro unità di memoria di lavoro utilizzata dalla CPU nell'elaborazione.

Report prospetto contenente i risultati di un'elaborazione disposti in modo ordinato sul foglio della stampante.

Reset ripristino delle condizioni iniziali di un sistema di elaborazione.

Restore recupero di dati o programmi da supporti contenenti copie di sicurezza delle memorie di massa realizzate con l'operazione di backup.

Rete (network) insieme di sistemi di elaborazione o di terminali interconnessi a distanza e che possono scambiare tra loro i dati con la possibilità di utilizzare risorse comuni.

Rimovibile disco di memoria che può essere rimosso dal drive, consentendo l'uso di supporti multipli.

ROM (Read-Only Memory) una memoria in cui i dati possono essere scritti una sola volta, ma da cui possono poi essere solo letti.

Router dispositivo di rete attivo in grado di instradare pacchetti provenienti da una rete verso un'altra rete.

Routine un insieme di istruzioni codificate; il termine viene usato come sinonimo di programma per il computer.

RSS formato standard, basato sul linguaggio XML, per la rappresentazione e la distribuzione di notizie e contenuti nel Web.

Run l'esecuzione di un programma in un computer.

S

Scanner un'apparecchiatura in grado di acquisire immagini o informazioni in un formato predefinito, trasformandole in segnali digitali per il computer.

Script codice che può essere eseguito direttamente da un programma in grado di interpretare il linguaggio con cui è stato creato lo script.

Scrittura (write) registrazione di dati nella memoria centrale o su memoria di massa.

Scrolling spostamento per righe, verso l'alto o il basso, all'interno di un testo visualizzato sul video.

Segnale rappresentazione elettrica o ottica di messaggi in modo che possano viaggiare su un mezzo trasmissivo.

Sequenziale tipo di accesso ai dati contenuti in una memoria in sequenza a partire dal primo; in contrapposizione con l'accesso diretto o random.

Server unità di elaborazione che mette a disposizione una o più risorse informatiche per altre unità (client) in una rete di computer.

Servlet programma scritto in linguaggio Java ed eseguito su un server Web.

Settore il minimo segmento della lunghezza di una traccia che può essere utilizzato per registrare informazioni in un disco.

Shareware software distribuito gratuitamente, ma che deve essere registrato e pagato dopo un certo periodo di utilizzo in prova.

Shell interprete dei comandi forniti dall'utente al sistema operativo, che controlla la correttezza formale e, in caso positivo, attiva la funzione corrispondente.

Sicurezza garanzia dei dati contenuti in un database contro i danni causati da malfunzionamenti di componenti hardware o software o da interventi dolosi e protezione dei dati dagli accessi non autorizzati.

Sincronizzazione scambio di dati tra apparecchiature portatili (notebook o palmari) e computer tradizionali.

Sincrono controllo delle attività del computer per il quale gli eventi sequenziali avvengono secondo tempi predefiniti.

Sistema un insieme di componenti che interagiscono tra loro per svolgere un'attività.

Sistema di elaborazione insieme di risorse hardware e software finalizzate alla risoluzione automatica di problemi e al trattamento automatico di informazioni.

Sistema operativo insieme organizzato di programmi che governa le risorse hardware e software di un sistema di elaborazione consentendone l'uso da parte degli utenti.

Sito Web insieme di pagine realizzate con la tecnologia World Wide Web residenti su un server Internet e visibili a tutti gli utenti che si collegano.

Slot alloggiamenti nella piastra principale di un computer predisposta per accogliere schede aggiuntive per l'espansione dell'hardware del sistema.

Smart Card tessera contenente un microprocessore con una serie di funzioni e dotato di memoria.

Smiley combinazioni di caratteri, segni grafici e icone utilizzati nei cellulari o nella posta elettronica.

SMTP (Simple Mail Transfer Protocol) protocollo utilizzato nell'architettura TCP/IP per inviare e-mail.

Social network (reti sociali) siti Web che consentono la pubblicazione in modo semplice di notizie, commenti, video e fotografie condividendole con altre persone.

Socket identificatore di una connessione TCP/IP formato dalla concatenazione dell'indirizzo di porta all'indirizzo IP.

Software insieme dei programmi e delle procedure che fanno funzionare una macchina.

Software applicativo programmi per il computer che servono ad automatizzare la gestione di uno specifico problema o settore aziendale.

Software di base insieme dei programmi fondamentali (sistema operativo e programmi di utilità) che consentono all'utente di utilizzare l'hardware del sistema.

Software house azienda che produce e commercializza prodotti software.

Sorgente testo di un programma scritto usando un linguaggio di programmazione.

Sort ordinamento di stringhe in ordine alfabetico o di numeri in ordine crescente o decrescente.

Spool (Simultaneous Peripheral Operations On-Line) tecnica del sistema operativo per dare la possibilità di continuare e concludere l'esecuzione di programmi che richiedono l'uso di periferiche, anche se non sono in quel momento disponibili, dirottando temporaneamente i dati di output su disco.

Spyware virus che raccoglie informazioni sul computer e sugli utenti che lo utilizzano.

SQL (Structured Query Language) linguaggio standard per le interrogazioni ai database.

Stack organizzazione di dati o informazioni secondo una struttura a pila (LIFO, Last In First Out).

Stick supporto di memoria di massa di piccole dimensioni e di grande capacità che può essere facilmente connesso a un computer tramite l'interfaccia USB.

Stringa sequenza di caratteri alfabetici o alfanumerici.

Subnet Mask sequenza di 32 bit, formata da un numero di bit di valore 1 consecutivi seguita da una sequenze di bit di valore 0 consecutivi con la quale un host individua la propria rete e la rete del destinatario per decidere se il pacchetto può essere inviato direttamente al destinatario o con l'appoggio al router di default. Esempio di netmask: 255.255.255.0.

Subroutine l'insieme di istruzioni inviate ad un computer per svolgere una specifica operazione; il termine è usato come sinonimo di sottoprogramma.

Supporto oggetto fisico per la registrazione di dati.

Switch (letteralmente interruttore) dispositivo fisico che può assumere stati diversi definendo la configurazione di un'unità hardware, facente parte del sistema di elaborazione.

Switch dispositivo di rete locale attivo che consente di interconnettere LAN dello stesso tipo e di inoltrare frame di una LAN verso un'altra LAN per separare il dominio di collisione e per migliorare l'efficienza complessiva della LAN.

T

Tabella insieme di dati organizzati nel modello logico dei database come una struttura a due dimensioni, righe e colonne.

Tablet computer a forma di tavoletta per l'informatica mobile dotato di touch screen.

Tag codici utilizzati da linguaggi per la descrizione della pagina (per esempio HTML).

TCP/IP (Transmission Control Protocol / Internet Protocol) protocollo standard utilizzato per formare e instradare pacchetti di informazioni su una rete locale o geografica; è il protocollo standard di Internet e delle reti locali.

Tempo di accesso (access time) l'intervallo di tempo che intercorre tra la richiesta dei dati, registrati in una memoria, e il momento in cui i dati sono disponibili per l'elaborazione.

Terminale un'apparecchiatura formata da una tastiera e da un video per inviare e ricevere dati in un collegamento con un computer centrale (host) o con accesso remoto a un sistema di elaborazione; differisce da un Personal Computer perché non possiede capacità elaborative proprie (processore e memoria).

Time-sharing distribuzione del tempo di utilizzo delle stesse risorse di un sistema fra utenti diversi.

Tool pacchetto software, che può essere utilizzato per una classe di problemi dello stesso tipo.

Topologia disposizione dei nodi di una rete locale.

Touch screen video sensibile al tatto con il quale l'utente può selezionare comandi puntando il dito sulla superficie del video di un computer.

TPI (Tracks Per Inch) numero delle tracce registrate all'interno di un inch (pollice) della superficie del disco magnetico; misura la densità di traccia.

Traccia (track) uno dei cerchi concentrici definiti sulla superficie di un disco magnetico come guida per la registrazione e la lettura dei dati.

Transfer rate velocità di trasferimento dalla memoria di massa (disco) alla memoria centrale, che si misura in MB/sec.

Trojan (Trojan Horse) tipo di virus informatico che tenta di prendere il controllo di un computer per svolgere alcune operazioni di danneggiamento.

U

UML sistema di notazione grafica per rappresentare classi utilizzato nel modello di programmazione orientata agli oggetti.

UNICODE codice di rappresentazione dei caratteri e dei simboli che comprende anche le lingue diverse da quelle occidentali (per esempio arabo, russo, cinese).

Unità componente del sistema di elaborazione dedicato a una specifica funzione.

Unità aritmetico-logica (Arithmetic Logic Unit) la parte dell'hardware di un processore nella quale vengono svolte le operazioni aritmetiche e logiche.

Unità di controllo (Control Unit) la parte dell'hardware di un processore che governa la sequenza delle operazioni, interpretando le istruzioni codificate e avviando gli appositi comandi verso le componenti del computer che devono effettuare l'esecuzione.

UNIX sistema operativo utilizzato su server di reti e minicomputer.

Upgrade aggiornamento di hardware o di software per rispondere a nuove esigenze.

Upload operazione che permette l'invio ad un host computer di informazioni o di file memorizzati sul proprio computer.

URL (Uniform Resource Locator) schema di rappresentazione di un indirizzo di un sito WWW nella rete Internet.

USB (Universal Serial Bus) interfaccia standard tra computer e periferiche di diverso tipo, che possono essere connesse senza schede aggiuntive e senza spegnere il computer (plug-and-play).

Username nome con il quale un utente viene riconosciuto da un sistema informatico al momento della connessione.

Utente persona che utilizza le prestazioni di un sistema di elaborazione.

Utilità software orientato a facilitare il lavoro del programmatore o dell'utente nell'uso delle risorse del sistema.

V

Virus programma in grado di rovinare o distruggere i file registrati su disco o presenti in memoria durante la loro esecuzione; il danneggiamento può coinvolgere altri file, trasferendosi anche su altri computer, come un'infezione elettronica.

VoiP (Voice over IP) tecnologia che consente di telefonare usando la rete Internet, anziché la linea telefonica tradizionale.

W

WAN (Wide Area Network) una rete di computer che si estende su un'area vasta, per esempio corrispondente al territorio di una Nazione, con distanze anche di migliaia di Km, e con velocità di trasmissione relativamente basse (da 10Kbit/sec a 10Mbit/sec).

WiMax tecnologia di reti wireless di tipo metropolitano che raggiunge una banda di 70 Mbps coprendo un'area di 50 Km circa. Standardizzata con la sigla IEEE 802.16.

Windows (letteralmente finestre) sistema operativo con interfaccia grafica della Microsoft, standard per i Personal Computer e per i server di rete, con menu, icone, mouse, finestre che consentono di rendere più amichevole l'interazione con il computer.

Wireless (in generale) tecniche di comunicazione tra apparecchiature che trasmettono e ricevono segnali senza l'uso di cavi.

Word processing software dedicato al trattamento dei testi con l'utilizzo del computer.

Workstation (stazione di lavoro) computer da scrivania dotato di hardware che consente elevate prestazioni e il collegamento con altri sistemi.

Worm tipo di virus informatico che si autoreplica occupando memoria e usando parti del sistema operativo; l'esecuzione delle altre applicazioni risulta quindi rallentata o bloccata.

WWW (World Wide Web) sistema ipermediale per l'organizzazione delle informazioni sulla rete Internet.

WYSIWYG (What You See Is What You Get) modalità di lavoro grafica con la quale il documento o il software in corso di creazione viene visualizzato nel modo in cui apparirà all'utente finale.

X

XML linguaggio standard per rappresentare e memorizzare dati strutturati attraverso tag semantici.

Soluzioni

Capitolo 1
1. b
2. c
3. V F V V
4. a) sia p che q sono veri;
   b) sia p che q sono falsi;
   c) p e q hanno valori di verità diversi;
   d) almeno uno dei due predicati è falso
5. b
6. a
7. c
8. a-4,5,9, b-2,7, c-1,6, d8, e3
9. a, c
10. c
11. F V V V

Capitolo 2
1. b
2. a
3. a) istruzioni
   b) processo
   c) processore
4. a) algoritmo,
   b) diagramma a blocchi
   c) pseudocodifica,
   d) programma software
5. c
6. a, b, e
7. 2
8. 1, 1
9. c
10. V F V V
11. b
12. d, a, b, c, e
13. a

Capitolo 3
1. V V F V
2. F V F V V
3. a
4. b
5. c
6. d
7. a3, b1, c2
8. no, no, si, no, si, si
9. a
10. d
11. c
12. a3, b4, c1, d5, e6, f2
13. F V F F F
14. F V V V F
15. a) 1, b) 3, c) 13
16. a) 8, b) 7
17. V F V F F
18. (AND) &&, (OR) ||, (NOT) !
19. a) 5; b) 3; c) 4
20. 10
21. 11
22. 5
23. dichiarazione, allocazione, inizializzazione
24. c
25. c
26. a) length,
   b) quadre,
   c) ArrayIndexOutOf BoundsException
   d) null
27. c
28. V F V V

Capitolo 4
1. c
2. V F V V F
3. c
4. b
5. F V F F
6. d
7. b
8. Obb, Obb, Obb, Fac, Fac
9. a
10. b
11. c
12. V F V V F
13. b
14. V F V F
15. F V F V F
16. get, set, interfaccia, privata
17. c
18. V V F F V V
19. null, this, super, overriding, overloading
20. b
21. sottoclasse, sopraclasse, gerarchia, grafo di gerarchia
22. d
23. a
24. d
25. a4, b2, c1, d3
26. F V V V

Capitolo 5
1. V F F V V
2. d
3. b
4. F V V V
5. Pila: Lifo – push, pop – top, vuota, size
   Coda: Fifo – aggiungi, togli – vuota, size
6. b
7. d
8. V F V V V
9. testa, coda, attraversamento, riferimento
10. a2, b3
11. b
12. V V V F V
13. a3, b1, c4, d2
14. d
15. a1, b1-2-3-4, c2, d3, e4, f1-3, g3, h1
16. b

Capitolo 6
1. F V V V F
2. c
3. a
4. V F F V V
5. a
6. c
7. b
8. b
9. d
10. c
11. showStatus, showDocument, Java Console, URL
12. V F V F
13. a
14. F V V F V
15. a6, b10, c1, d2, e9, f3, g8, h4, i5, l7
16. repaint, setColor, drawArc, drawString
17. d
18. a
19. b

Capitolo 7
1. ODBC, DSN, JDBC
2. V F V F V
3. c
4. d
5. a
6. a
7. c

Capitolo 8
1. V F V F V
2. a
3. WWW, server Web, browser Web, HTTP
4. V F V V
5. c
6. c, b, a, e, d
7. b
8. V V F F
9. V F V V F
10. c
11. c
12. b
13. b
14. c
15. d
16. a
17. b

Capitolo 9
1. V V F V
2. d
3. a4, b1, c2, d3
4. F V F V
5. widget, back, stato, home
6. V F V V
7. a
8. c
9. c
10. AndroidManifest.xml, /res/layout/, Code, Graphical Layout, Properties
11. V V F F
12. b
13. a3, b1, c4, d2
14. c
15. d
16. F V V V
17. b
18. d
19. V V V F

Indice analitico

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

ISTITUTO ITALIANO EDIZIONI ATLAS
24123 BERGAMO - VIA CRESCENZI, 88 - TEL. 035/24.97.11 - FAX 035/21.60.47
www.edatlas.it

La proposta editoriale mista *Java Programmazione ad oggetti e applicazioni Android* comprende:

MATERIALI A STAMPA

- Volume unico

E-BOOK PER COMPUTER, TABLET E LIM

- Disponibile anche la versione digitale con espansioni multimediali:
  - test strutturati interattivi
  - lezioni multimediali (videoanimazioni con commento vocale)
  - progetti aggiuntivi di approfondimento
  - aggiornamenti sui software presentati nel testo.

MATERIALI INTEGRATIVI *ON LINE*

- Approfondimenti e integrazioni dei contenuti trattati nel testo
- Note operative sull'uso di strumenti per lo sviluppo software.

PER IL DOCENTE

- Materiali didattici disponibili nell'area riservata del sito dell'Atlas, in particolare:
  - traccia per la compilazione dei Piani di lavoro per i Consigli di classe
  - tracce di soluzione ai problemi del testo
  - repertorio di esercizi da assegnare come verifiche in classe.

- CD-ROM con materiali multimediali e interattivi:
  - presentazioni in *PowerPoint* e in *pdf* che illustrano i contenuti dei capitoli e che possono essere utilizzati con la LIM per lezioni multimediali in classe;
  - codici sorgente completi dei programmi presentati nel volume;
  - ulteriore repertorio di esercizi che possono essere assegnati come autoverifiche per gli studenti.

Java
Programmazione ad oggetti e applicazioni Android