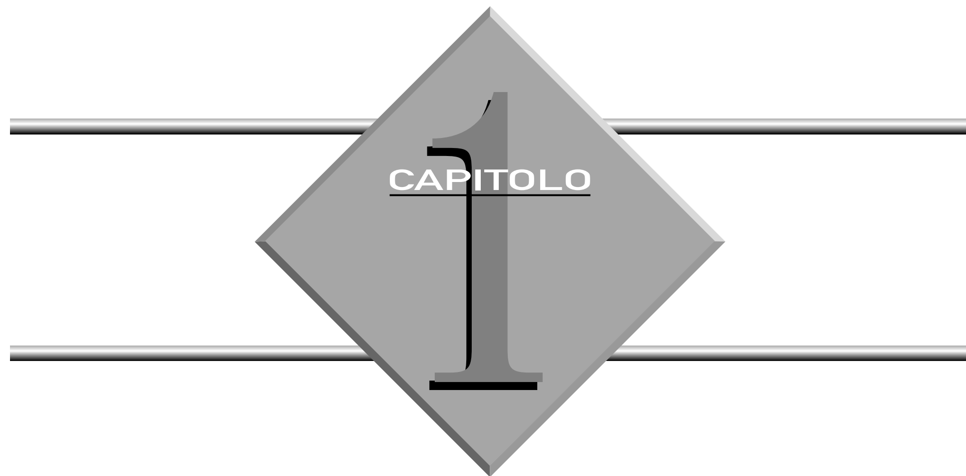


Introduzione

Obiettivi del capitolo

- Capire il significato e l'importanza dell'attività di programmazione
- Imparare a riconoscere le componenti più importanti dell'architettura dei computer
- Comprendere la distinzione fra i linguaggi macchina e i linguaggi di programmazione ad alto livello
- Prendere confidenza con il compilatore
- Compilare ed eseguire il primo programma Java
- Capire i concetti di classi e di oggetti
- Riconoscere gli errori logici e di sintassi

1.1 Che cos'è un computer?

Probabilmente, avete già usato un computer per lavoro o per svago. Molta gente utilizza i computer per attività quotidiane, come per esempio calcolare il saldo di un conto corrente bancario o scrivere un elaborato trimestrale. I computer sono ottimi per questi lavori, perché possono gestire operazioni ripetitive, per esempio sommare numeri o inserire parole in una pagina, senza ridurvi alla noia o all'esaurimento. Ancora più importante, presentano gli assegni o il saggio trimestrale sullo schermo, permettendovi di rimediare facilmente agli errori. I computer sono ottime macchine per giocare, perché possono riprodurre sequenze di suoni e di immagini, coinvolgendo l'utente umano nel processo. In realtà, ciò che rende possibile tutto questo non è soltanto il computer: il computer deve essere programmato per svolgere queste attività. Un dato programma calcola il saldo del conto corrente; un altro programma, probabilmente progettato e realizzato da una società diversa, elabora i testi; infine, un terzo programma esegue un gioco. Di per sé, un computer è una macchina che immagazzina dati (numeri, parole, immagini), interagisce con dispositivi (lo schermo del monitor, il sistema audio, la stampante) ed esegue programmi. I programmi sono sequenze di istruzioni e di decisioni, che il computer esegue per svolgere un'attività.

Gli attuali programmi per computer sono talmente sofisticati che è difficile credere che siano composti interamente da operazioni estremamente semplici. Qualche esempio di queste operazioni:

- Metti un punto rosso in questa posizione dello schermo.
- Invia la lettera A alla stampante.
- Estrai un numero da questa posizione della memoria.
- Somma questi due numeri.
- Se questo valore è negativo, continua il programma da quella istruzione.

È soltanto perché un programma contiene un numero enorme di operazioni come queste e perché il computer può eseguirle a grande velocità che l'utente ha l'illusione di un'interazione scorrevole.

La flessibilità di un computer è un fenomeno davvero affascinante. La stessa macchina può calcolare il saldo di conto corrente, stampare una relazione ed eseguire un gioco. In confronto, altre macchine svolgono una gamma di attività più ristretta: un'automobile viaggia, un tostapane tosta il pane. I computer possono svolgere un'ampia serie di attività perché eseguono programmi diversi, ciascuno dei quali indirizza il computer a lavorare su una specifica attività.

1.2 Che cos'è la programmazione?

Un programma indica al computer, nei minimi dettagli, la sequenza di passaggi che sono necessari per eseguire un determinato compito. L'attività di progettare e di implementare questi programmi è detta *programmazione*. In questo corso, imparerete come programmare un computer, ovvero come far sì che un computer esegua determinate attività.

Per usare un computer non è necessaria alcuna programmazione. Quando scrivete una relazione mediante un word processor, utilizzate un programma che il costruttore ha già definito e che, quindi, è pronto per l'uso. Questa è l'unica cosa che ci si aspetta: infatti, potete guidare un'automobile senza essere un meccanico e tostare il pane senza essere un elettricista. Molte persone, che nella loro professione usano il computer tutti i giorni, non hanno mai avuto bisogno di fare alcuna programmazione.

Naturalmente, un informatico professionista o un ingegnere del software svolge una grande attività di programmazione. Dal momento che seguite questo primo corso di informatica, potrebbe benissimo rientrare fra i vostri obiettivi di carriera diventare informatici di professione. La programmazione non è l'unica competenza richiesta a un informatico o a un ingegnere del software; anzi, non è nemmeno l'unica competenza necessaria per creare buoni programmi. Nondimeno, è una componente importante delle scienze informatiche ed è anche un'attività affascinante e piacevole, che continua ad attrarre e a motivare gli studenti. L'informatica è una disciplina particolarmente fortunata perché può mettere un'attività così interessante alla base del proprio percorso formativo.

Scrivere un gioco per computer, con movimenti ed effetti sonori, oppure un word processor che supporti caratteri tipografici personalizzati e immagini, è un compito complesso, che richiede una squadra di molti programmatori altamente specializzati. I vostri primi esercizi di programmazione saranno più terra terra, ma i concetti e le competenze che apprenderete in questo corso costituiscono una base importante, quindi non dovrete demoralizzarvi se il vostro primo programma non potrà competere con il software sofisticato che vi è familiare. In realtà, scoprirete che anche semplici attività di programmazione sono molto stimolanti. È un'esperienza assai gratificante vedere il computer svolgere con precisione e rapidità un'attività che avrebbe richiesto ore di fatica, oppure apportare a un programma piccole modifiche che producono miglioramenti immediati, e vedere il computer diventare un prolungamento dei vostri poteri mentali.

1.3 L'anatomia di un computer

Per capire il processo di programmazione, è necessario capire almeno per grandi linee gli elementi costitutivi che formano un computer. Pertanto, daremo uno sguardo a un personal computer: le macchine di maggiori dimensioni hanno componenti più veloci, più grandi o più potenti, ma hanno sostanzialmente la stessa struttura.

Nel cuore del computer si trova l'*unità centrale di elaborazione* (CPU, central process unit, osservare la Figura 1), che è formata da un unico *chip* o da un piccolo numero di chip. Un chip, o circuito integrato, di un computer è un componente con connettori metallici e collegamenti interni, costituito principalmente di silicio e alloggiato in un contenitore di plastica o di metallo. Nel chip di una CPU, i collegamenti interni sono enormemente complicati. Per esempio, il chip Pentium (una CPU molto diffusa nel periodo in cui scriviamo), è composto di oltre tre milioni di elementi strutturali, detti *transistor*. La Figura 2 mostra una vista ingrandita dei particolari nel chip di una CPU. La CPU svolge il controllo del programma e funzioni di calcolo e di trasferimento dati, ovvero individua ed esegue le istruzioni del programma, effettua le operazioni aritmetiche, quali addizioni, sottrazioni, moltiplicazioni e divisioni, e reperisce dati dalla

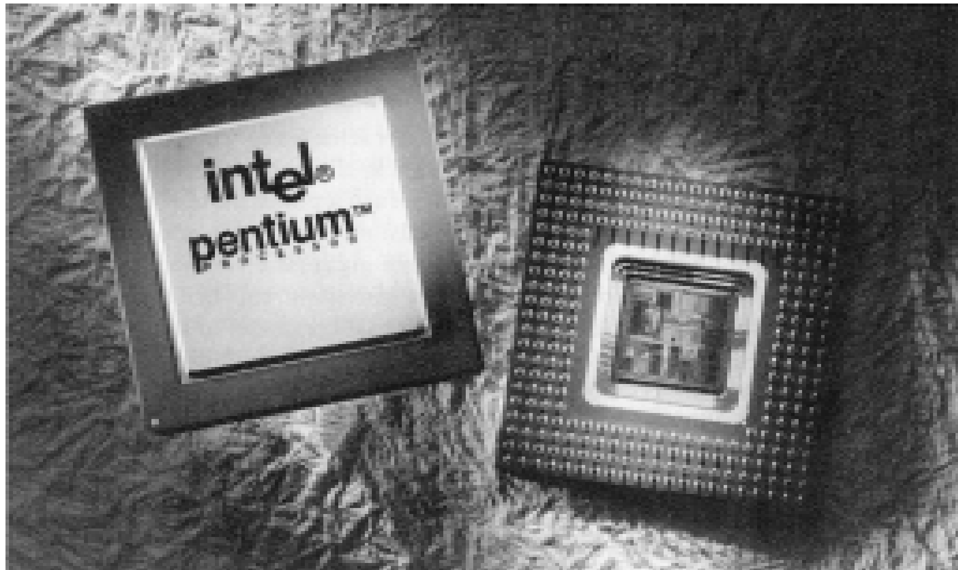


Figura 1
Unità centrale di elaborazione

memoria esterna o dalle apparecchiature periferiche, oppure li rimanda indietro. Tutti i dati devono transitare per la CPU, quando vengono spostati da una posizione all'altra, sebbene vi siano alcune eccezioni tecniche a questa regola, perché alcuni dispositivi possono interagire direttamente con la memoria.

Il computer immagazzina dati e programmi nella *memoria.* Esistono due tipi di memoria. La *memoria primaria* è veloce, ma costosa; è costituita da chip di memoria (osservare la Figura 3), che formano la *memoria ad accesso casuale* (RAM, random access memory) e la *memoria di sola lettura* (ROM, read only memory). La memoria di sola lettura contiene certi programmi che devono essere sempre presenti, quali, per esempio, il codice necessario per avviare il computer. La memoria ad accesso casuale si potrebbe chiamare più opportunamente"memoria di lettura e scrittura", perché la CPU può sia leggervi dati, sia scriverne. Questa caratteristica rende la RAM adatta per ospitare dati in fase di modifica e programmi che non devono essere sempre disponibili. La memoria RAM ha due svantaggi: è relativamente costosa e perde tutti i dati quando si spegne il computer. La *memoria secondaria*, generalmente un *disco rigido* (osservare la Figura 4), permette una registrazione dei dati meno costosa e che perdura anche in assenza di elettricità. Un disco rigido è formato da piatti rotanti, rivestiti da materiale magnetico, e da testine di lettura/scrittura, in grado di leggere e di modificare il flusso magnetico sui piatti. Sostanzialmente, si tratta dello stesso processo di registrazione magnetica utilizzato nei nastri audio o video. Generalmente, programmi e dati sono registrati nel disco rigido e vengono caricati nella RAM all'avvio del programma. Successivamente, il programma aggiorna i dati nella RAM e riporta i dati modificati sul disco rigido.

Spesso, si usa un altro tipo di dispositivo a registrazione magnetica, un cosiddetto *disco floppy* o *dischetto*. Originariamente, i dischi floppy erano di capacità piuttosto limitata, ma recentemente si sono diffusi dischetti ad alta capacità, quali i dischi Zip e Superdisk (osservare la Figura 5). Un disco floppy consiste di una base rotonda flessi-

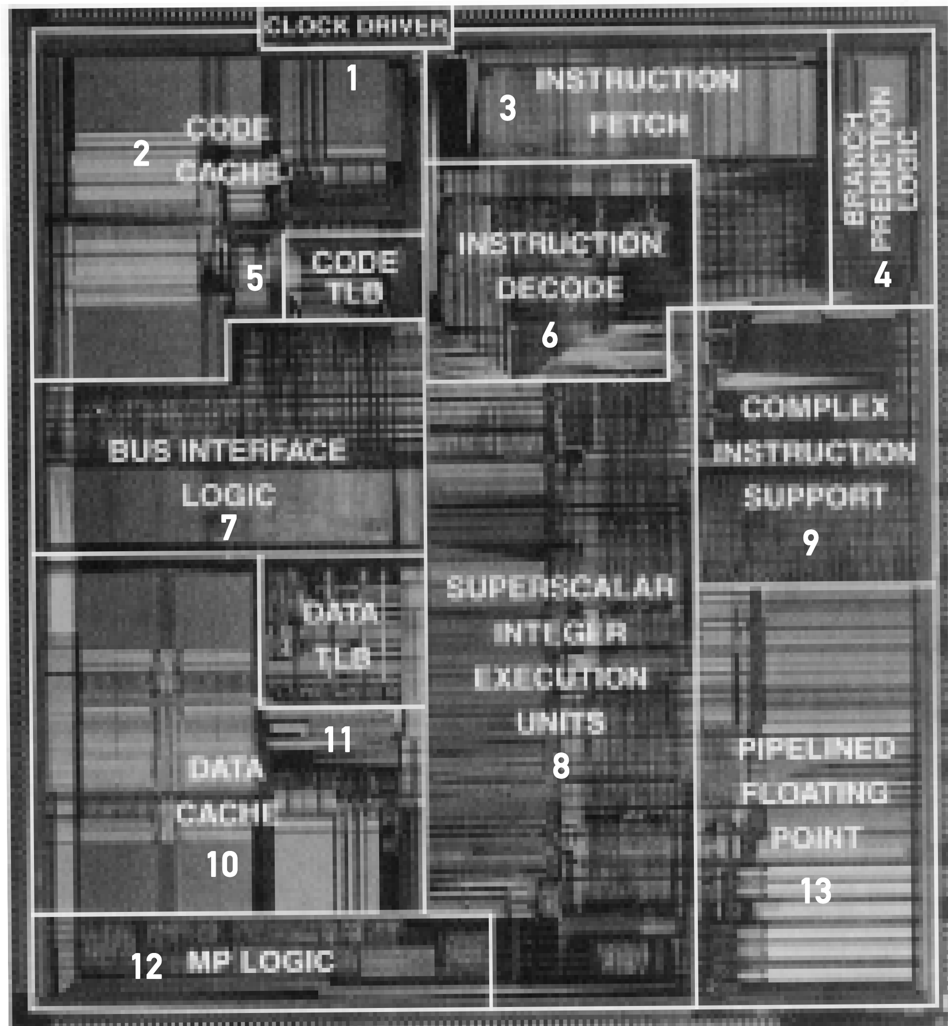


Figura 2
Particolare del chip della CPU
1) Driver del clock
2) Cache del codice
3) Reperimento delle istruzioni
4) Logica per la previsione dei salti
5) TLB del codice
6) Decodifica delle istruzioni
7) Logica dell'interfaccia bus
8) Unità superscalari per l'esecuzione dei numeri interi
9) Supporto per istruzioni complesse
10) Cache dei dati
11) TLB dei dati
12) Virgola mobile pipelined
13) Logica MP

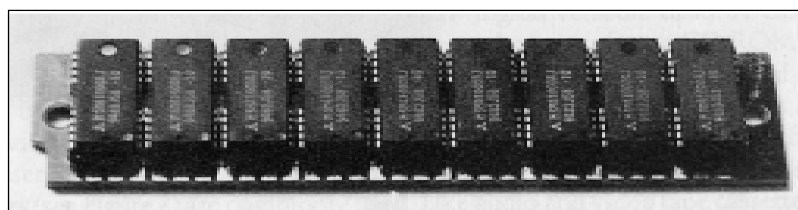


Figura 3
Diversi chip di RAM

Figura 4
Un disco rigido

Figura 5
Un disco floppy
ad alta capacità
e la sua unità

bile (da cui il nome *floppy*, flessibile), rivestita di materiale magnetico, collocata all'interno di una custodia di plastica (generalmente non flessibile). Come un disco rigido, anche un disco floppy può registrare dati e programmi, che si conservano in assenza di elettricità. I dischi floppy si usano soprattutto per trasferire dati da un computer all'altro: potete copiare dati dal vostro computer di casa su un dischetto e portarlo a scuola per continuare a lavorarvi, oppure potete spedire il dischetto per posta. Poiché un disco floppy non è una parte integrante del sistema del computer, viene detto *dispositivo di memoria esterno*.

Le unità a disco floppy sono poco costose, relativamente robuste e convenienti, ma soffrono di una limitazione: un dischetto non può contenere neppure lontanamente il volume di dati di un disco rigido. Non è un grosso problema per i vostri dati personali, dal momento che è abbastanza probabile che tutto il lavoro a casa che farete per questo corso starà comodamente in un solo dischetto. Tuttavia, dati audio e video consumano molto più spazio di quello disponibile su un floppy e, generalmente, informazioni di questo tipo si distribuiscono su CD-ROM (compact disc read-only memory, ovvero disco compatto per memoria di sola lettura, osservare la Figura 6) o DVD (digital versatile disc, ovvero disco digitale multiuso). Un CD-ROM si presenta esattamente come un CD audio (infatti, il lettore di CD-ROM di un personal computer può riprodurre anche un CD audio) e viene letto da un dispositivo laser. Un CD-ROM può contenere una grande quantità di informazioni e si produce a basso costo, ma è un dispositivo di sola lettura: si può usare solamente per rilasciare dati e programmi dal costruttore all'utente, mentre l'utente non può usarlo per registrare più informazioni.

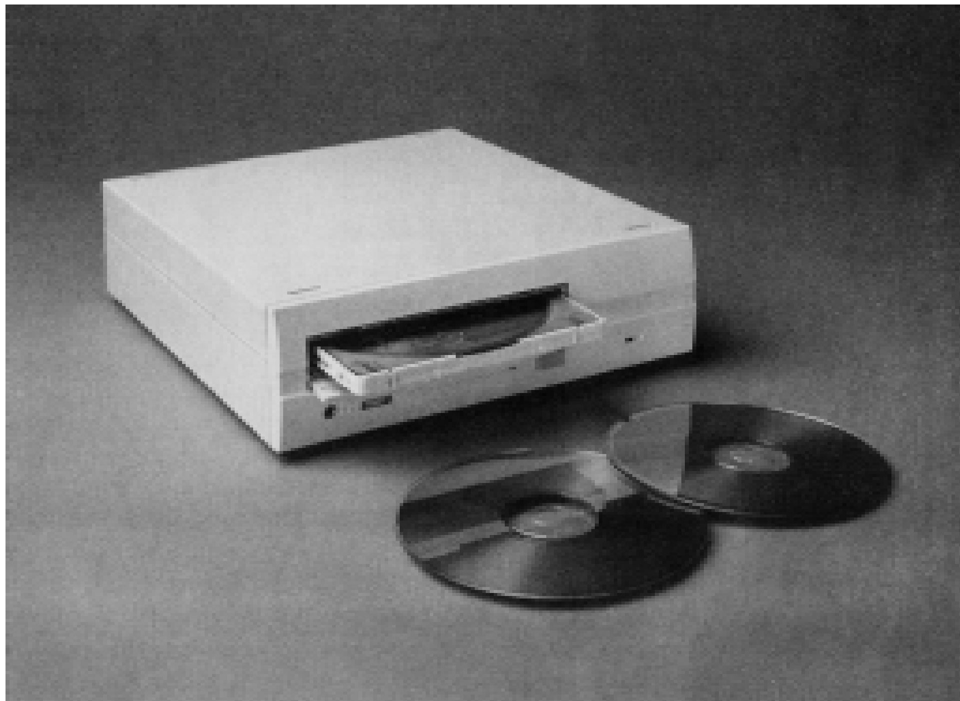


Figura 6
Un'unità CD-ROM

Per conservare grandi quantità di dati da parte dell'utente, solitamente si usano *nastri per dati* (osservare la Figura 7). Analogamente alle cassette audio e video, questi nastri contengono un lungo rotolo di nastro magnetico per leggere e per scrivere dati.

I nastri per dati sono economici, ma lenti. Per localizzare i dati a metà del nastro, bisogna svolgerlo fino al tratto che contiene le informazioni, un'operazione molto più lenta dello spostamento di una testina sopra un piatto rotante.

Alcuni computer sono unità autosufficienti, mentre altri sono interconnessi tramite *reti*. I computer per casa di solito vengono connessi a Internet saltuariamente, tramite un modem. I computer di un laboratorio informatico probabilmente sono connessi a una rete locale in modo permanente. Attraverso i cablaggi della rete, il computer può leggere programmi da una posizione di raccolta centralizzata oppure inviare dati ad altri computer. Per l'utente di un computer connesso, non sempre può essere semplice distinguere i dati che risiedono sulla propria macchina da quelli che vengono trasmessi attraverso la rete.

Per interagire con l'utente, un computer ha bisogno di altri dispositivi periferici, perché trasmette le informazioni all'utente mediante uno schermo di visualizzazione, altoparlanti e stampanti. L'utente può inserire informazioni e impartire ordini al computer tramite una tastiera o un dispositivo di puntamento, quale un mouse. La Figura 8 mostra un tipico personal computer, equipaggiato con questi dispositivi.

L'unità di elaborazione centrale, la memoria RAM, l'elettronica che controlla il disco rigido e gli altri dispositivi sono interconnessi mediante un insieme di linee elettriche, che formano un *bus*. I dati transitano lungo il bus, dalla memoria del sistema e dai dispositivi periferici verso la CPU e viceversa. La Figura 9 mostra una *scheda principale* o scheda madre, che contiene la CPU, la RAM e gli alloggiamenti per le schede, tramite i quali si connettono al bus le schede elettroniche che controllano i dispositivi periferici.

La Figura 10 presenta una vista schematica dell'architettura di un computer. Le istruzioni di programma e i dati (quali testi, numeri, sequenze audio o video) sono

Figura 7
Due unità a nastro per backup e un nastro dati

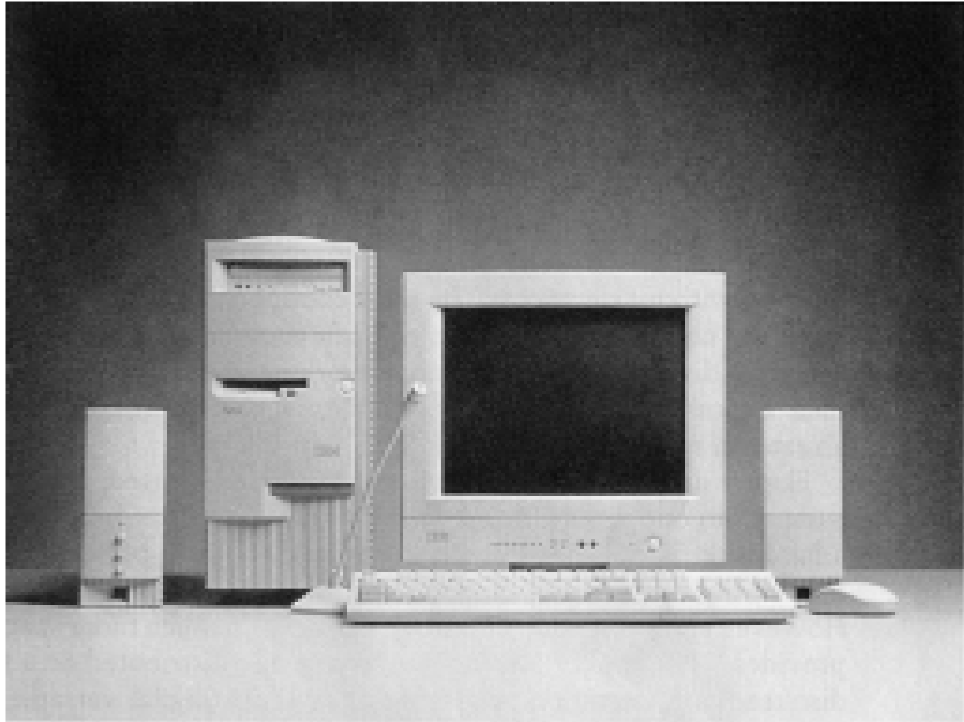


Figura 8
Un personal computer

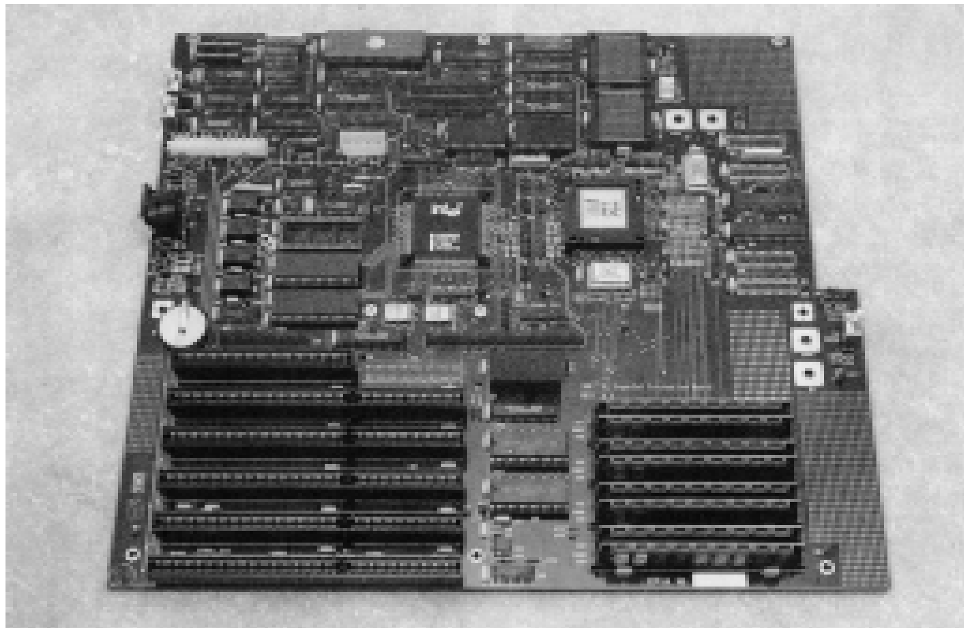


Figura 9
Una scheda principale

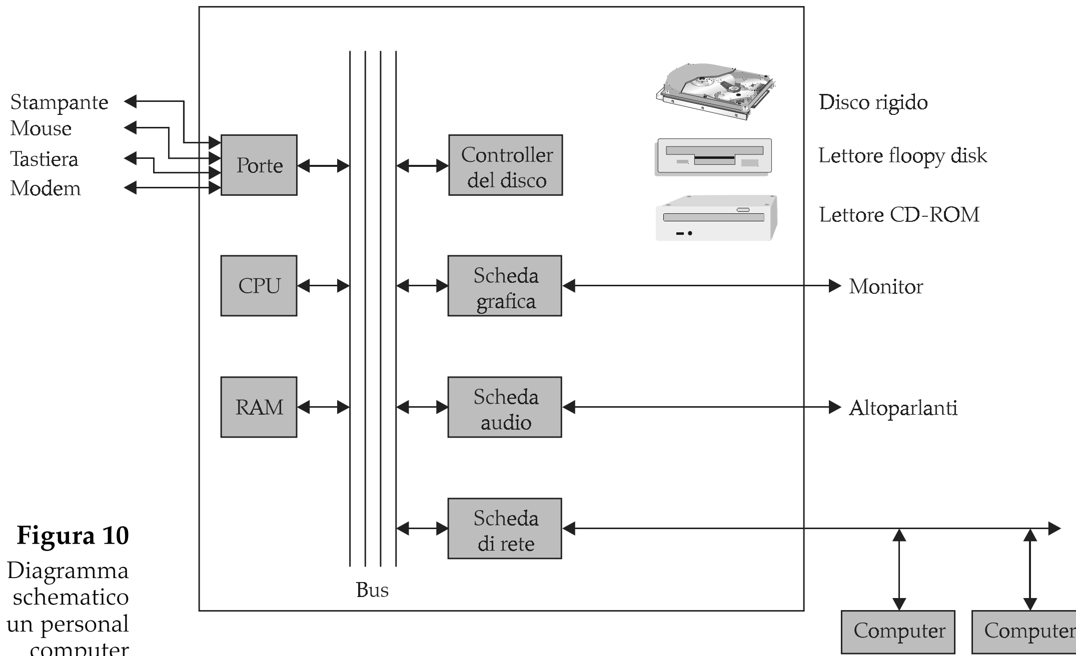


Figura 10 Diagramma schematico di un personal computer

conservati nel disco rigido, su un CD-ROM o in qualche punto della rete. Quando si avvia un programma, viene caricato nella memoria RAM, dove la CPU può leggerlo al ritmo di un'istruzione alla volta. A seconda delle direttive espresse da queste istruzioni, la CPU legge i dati, li modifica e li registra nuovamente nella memoria RAM o nel disco rigido. Alcune istruzioni di programma indurranno la CPU a posizionare punti sullo schermo o a inviarli alla stampante, oppure a far vibrare l'altoparlante. Mentre queste azioni si ripetono molte volte e a grande velocità, l'utente umano percepirà immagini e suoni. Alcune istruzioni di programma leggono gli input dell'utente dalla tastiera o dal mouse. Il programma esamina la natura di questi input ed esegue l'istruzione successiva appropriata.

Note di cronaca 1.1

L'ENIAC e gli albori dell'informatica

L'ENIAC (Electronic Numerical Integrator And Computer, integratore numerico elettronico e calcolatore), fu il primo computer elettronico utilizzabile. Fu progettato da J. Presper Eckert e John Mauchly, presso l'University of Pennsylvania, e venne completato nel 1946, due anni prima dell'invenzione dei transistor. Il computer era ospitato in uno stanzone ed era formato da molti armadi, che contenevano circa 18.000 valvole termoioniche (osservare la Figura 11). Le valvole termoioniche si bruciavano al ritmo di molte al giorno e un inserviente, con un carrello della spesa pieno di valvole, girava

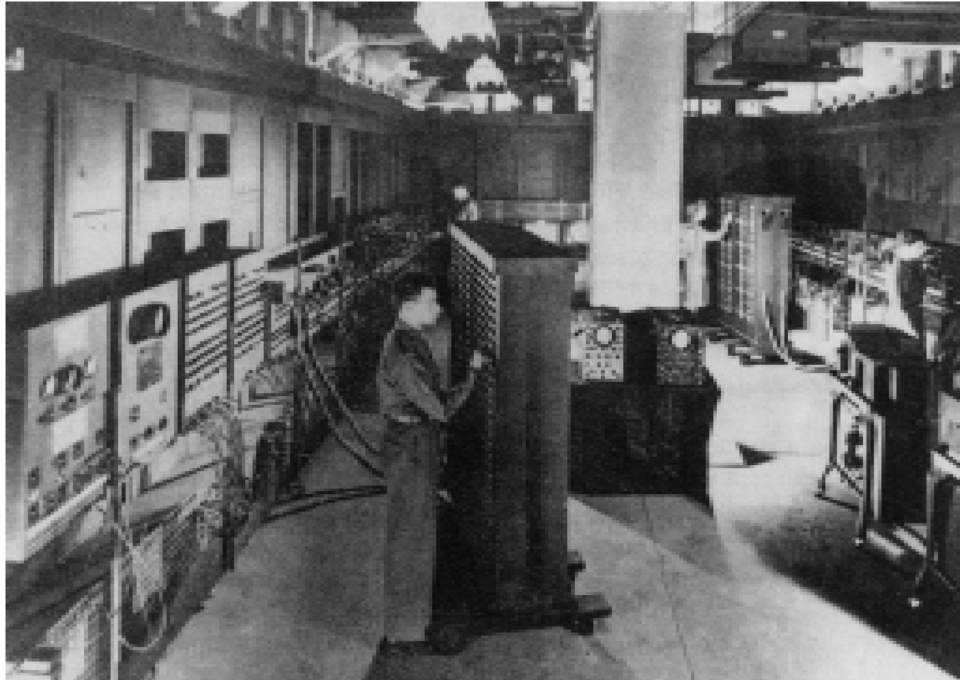


Figura 11
L'ENIAC

continuamente per sostituire quelle difettose. Il computer veniva programmato collegando cavi su pannelli e ciascuna configurazione delle connessioni impostava il computer per un problema specifico. Per consentire al computer di lavorare su un problema diverso, occorreva cambiare la connessione dei cavi.

Il lavoro sull'ENIAC era supportato dalla U.S. Navy, la Marina degli Stati Uniti, che era interessata al calcolo di tavole balistiche che dovevano fornire la traiettoria di un proiettile, in funzione della resistenza del vento, della velocità iniziale e delle condizioni atmosferiche.

Per calcolare le traiettorie, bisognava trovare le soluzioni numeriche di determinate equazioni differenziali, da cui sortì il nome di"integratore numerico". Prima che venissero sviluppate macchine come l'ENIAC, questo tipo di lavoro veniva svolto da esseri umani e, fino al 1950, il termine "computer"indicava queste persone. Più tardi, l'ENIAC venne impiegato per scopi pacifici, quali la catalogazione dei dati dello U.S. Census, il servizio demografico statunitense.

1.4 Tradurre in codice macchina programmi leggibili dalle persone

Al livello più basso, le istruzioni di un computer sono estremamente elementari. Il processore esegue *istruzioni macchina.* Le CPU di fornitori diversi, quali il Pentium

Intel o lo SPARC della Sun, hanno insiemi differenti di istruzioni macchina. Per consentire ai programmi Java di girare su numerose CPU senza adattamenti, la maggior parte dei compilatori Java genera una serie di istruzioni macchina per una cosiddetta "macchina virtuale Java", ovvero una CPU ideale che viene simulata sulla CPU effettiva, mediante l'esecuzione di un programma. Per noi, la differenza fra le istruzioni per la macchina reale e per quella virtuale non è importante e tutto quello che occorre sapere è che le istruzioni macchina sono molto semplici e si possono eseguire assai rapidamente. Ecco una tipica sequenza di istruzioni macchina:

- Carica il contenuto della posizione di memoria 40.
- Carica il valore 100.
- Se il primo valore è maggiore del secondo, prosegui con l'istruzione contenuta nella posizione di memoria 240.

In pratica, le istruzioni macchina sono codificate sotto forma di numeri, in modo da poterle conservare in memoria. Nella macchina virtuale Java, la serie di istruzioni precedente è codificata secondo questa sequenza di numeri:

21 40 16 100 163 240

Con un processore del tipo Pentium Intel, la codifica potrebbe essere abbastanza diversa. Quando la macchina virtuale reperisce questa sequenza di numeri, li decodifica ed esegue la serie dei comandi corrispondenti.

In quale maniera possiamo comunicare la sequenza di comandi al computer? Il metodo più semplice è quello di inserire i numeri effettivi nella sua memoria. Di fatto, questo è il modo in cui lavoravano i primissimi computer. Tuttavia, un programma lungo è composto da migliaia di comandi singoli ed è noioso, nonché causa di errori, cercare i codici numerici di tutti i comandi per poi inserirli manualmente nella memoria. Come già detto precedentemente, i computer si prestano molto bene all'automazione delle attività noiose e inclini agli errori, e ai programmatori non occorse molto per comprendere che potevano sfruttare i computer stessi per agevolare l'attività di programmazione.

Il primo passo fu di assegnare nomi abbreviati ai comandi. Per esempio, `iload` indica"carica un numero intero"(integer load), `bipush` significa"inserisci una costante numerica"(push integer constant), `if_icmpgt` sta per"se è maggiore nel confronto numerico" (if integer compare greater). Utilizzando questi comandi, la sequenza di istruzioni diventa:

```
iload      40
bipush     100
if_icmpgt  240
```

Per gli umani, questa forma è molto più facile da leggere. Tuttavia, per fare accettare le sequenze di istruzioni al computer, bisogna convertire i nomi nei rispettivi codici macchina. I primi computer utilizzavano un programma detto *assembler* (assemblatore), per effettuare queste conversioni. Un assemblatore prende la sequenza di caratteri, per esempio `iload`, la traduce nel codice di comando 21 e poi svolge la stessa operazione sugli altri comandi. Gli assemblatori hanno un'altra caratteristica: possono assegnare nomi sia alle *posizioni di memoria,* sia alle istruzioni. Per esempio, la nostra

sequenza di programma potrebbe dover rilevare se qualche tasso d'interesse è superiore al 100 per cento, con il tasso nella posizione di memoria 40. Generalmente, la posizione in cui viene conservato un valore non è importante e qualsiasi posizione di memoria disponibile va bene. Grazie all'impiego di nomi simbolici al posto degli indirizzi di memoria, il programma diventa ancora più facile da leggere:

```
iload     intRate
bipush    100
if_icmpgt intError
```

Il programma assemblatore ha il compito di cercare i valori numerici adatti per i nomi simbolici e inserire quei valori nella sequenza di codice generata.

Le istruzioni assembler rappresentarono un progresso significativo rispetto a programmare con codici macchina grezzi, ma sono afflitte da due problemi. Per prima cosa, occorre ancora un gran numero di istruzioni per eseguire anche le operazioni più semplici, e l'esatta sequenza delle istruzioni differisce da un processore all'altro. Per esempio, la sequenza precedente di codice macchina è valida solo per la macchina virtuale Java, non per processori Pentium o SPARC. Questo è un problema reale per coloro che investono un sacco di tempo e di denaro per produrre un pacchetto software: se un computer diventa obsoleto, bisogna riscrivere il programma completamente, affinché possa operare sul sistema sostitutivo.

A metà degli Anni 50 iniziarono ad apparire linguaggi di programmazione ad alto livello. In questi linguaggi, il programmatore esprime l'idea che sta dietro all'operazione da compiere, mentre un programma speciale, detto *compilatore,* traduce la descrizioni di alto livello nelle istruzioni macchina idonee per un processore specifico.

Per esempio, in Java, il linguaggio di programmazione ad alto livello che useremo in questo corso, potrete impartire l'istruzione seguente:

```
if (intRate > 100) System.out.print("Errore nel tasso di interesse");
```

L'istruzione significa"se il tasso d'interesse è maggiore di 100, visualizza un messaggio di errore". Poi, sarà compito del programma compilatore individuare la sequenza di caratteri `if (intRate > 100)` e tradurla in:

21 40 16 100 163 240

I compilatori sono programmi piuttosto sofisticati. Hanno il compito di tradurre enunciati logici, quali `if`, in serie di calcoli, verifiche e salti di programma, e devono reperire le posizioni di memoria delle variabili come `intRate`. In questo corso, generalmente daremo per scontata la presenza di un compilatore. Se diventerete informatici professionisti, potrete sempre approfondire le tecniche di scrittura dei compilatori proseguendo nei vostri studi.

I linguaggi di alto livello sono indipendenti dall'hardware sottostante. Per esempio, l'istruzione `if (intRate > 100)` non si basa su istruzioni macchina particolari. Di fatto, sarà compilata in un codice diverso quando verrà eseguita quale codice nativo su un processore Pentium o SPARC, piuttosto che su una macchina virtuale Java.

1.5 Linguaggi di programmazione

I linguaggi di programmazione sono indipendenti dall'architettura specifica del computer, ma sono creati dall'uomo e, quindi, seguono certe convenzioni. Per semplificare il processo di traduzione, queste convenzioni sono più rigide di quelle dei linguaggi umani. Quando parlate con un'altra persona e pasticciate o omettete una parola o due, solitamente il vostro interlocutore capirà comunque quanto dovete dire. I compilatori sono meno indulgenti. Per esempio, se tralasciate le virgolette di chiusura alla fine dell'istruzione:

```
if (intRate > 100) System.out.print("Errore nel tasso di interesse);
```

il compilatore Java rimarrà piuttosto confuso e si lamenterà di non poter tradurre un'istruzione che contiene questo errore. In realtà, questa è una cosa positiva. Se il compilatore tentasse di indovinare che cosa avete fatto di sbagliato e di rimediarvi, potrebbe non comprendere correttamente le vostre intenzioni. In questo caso, il programma risultante non farebbe quanto previsto, probabilmente con effetti disastrosi se il programma controlla un dispositivo dalle cui funzioni dipende il benessere di qualcuno. Quando un compilatore legge le istruzioni in un linguaggio di programmazione, le tradurrà in codice macchina solamente se l'input si attiene fedelmente alle convenzioni del linguaggio.

Proprio come esistono diverse lingue, vi sono molti linguaggi di programmazione. Osservate l'istruzione seguente:

```
if (intRate > 100) System.out.print("Errore nel tasso di interesse");
```

Questo è il formato che dovete impiegare per l'istruzione in Java. Java è un linguaggio di programmazione molto diffuso ed è quello che useremo in questo libro. In Pascal, un altro linguaggio di programmazione usato comunemente negli Anni 70 e 80, la stessa istruzione sarebbe scritta così:

```
if intRate > 100 then write('Errore nel tasso di interesse');
```

In questo caso, le differenze fra Java e Pascal sono lievi, ma, in altri costrutti, sarebbero di gran lunga più sostanziali. Per ciascun linguaggio esiste un compilatore specifico: il compilatore Java tradurrà solamente codice Java, mentre un compilatore Pascal rifiuterà qualsiasi cosa diversa dal codice Pascal omologato. Per esempio, se un compilatore Java legge l'istruzione `if intRate > 100 then ...` protesterà, perché la condizione dell'enunciato `if` non è racchiusa fra parentesi e il compilatore non si aspetta la parola `then`. La scelta di un formato per il costrutto di un linguaggio, come l'enunciato `if`, è piuttosto arbitraria e i progettisti dei vari linguaggi scelgono compromessi diversi fra leggibilità, facilità di interpretazione e coerenza con altri linguaggi.

1.6 Il linguaggio di programmazione Java

Nel 1991, un gruppo della Sun Microsystems, guidato da James Gosling e Patrick Naughton, progettò un linguaggio, chiamato in codice"Green", per l'utilizzo in appa-

recchi di consumo come le scatole"set-top"per televisori intelligenti. Il linguaggio era progettato per essere semplice e neutrale rispetto all'architettura, in modo da operare su hardware diversi, ma non si trovò mai alcun cliente per questa tecnologia.

Gosling racconta che nel 1994 la squadra si rese conto che:"Avremmo potuto scrivere un browser davvero eccellente. Nel filone client/server, era una delle poche cose che aveva bisogno di alcuni degli inconsueti risultati che avevamo ottenuto: neutralità rispetto all'architettura, esecuzione in tempo reale, affidabilità, sicurezza.". Il browser HotJava, che fu presentato a una folla entusiasta durante la mostra SunWorld del 1995, aveva una caratteristica unica: poteva scaricare programmi, chiamati *applet*, dal Web ed eseguirli. Le applet, scritte in un linguaggio che adesso si chiama Java, permettono agli sviluppatori Web di predisporre una serie di animazioni e di interazioni che possono migliorare molto le potenzialità di una pagina (osservare la Figura 12). Nel 1996, sia la Netscape, sia la Microsoft, supportarono Java nei loro browser.

Da allora, Java è cresciuto a un ritmo fenomenale. I programmatori l'hanno adottato perché è più semplice del suo rivale che più gli assomiglia, il C++. Oltre al linguaggio di programmazione, Java ha una ricca *libreria*, che permette di scrivere programmi trasferibili in grado di scavalcare i sistemi operativi proprietari, una funzionalità attesa ansiosamente da quanti volevano essere indipendenti da quei sistemi proprietari e decisamente osteggiata dai loro fornitori.

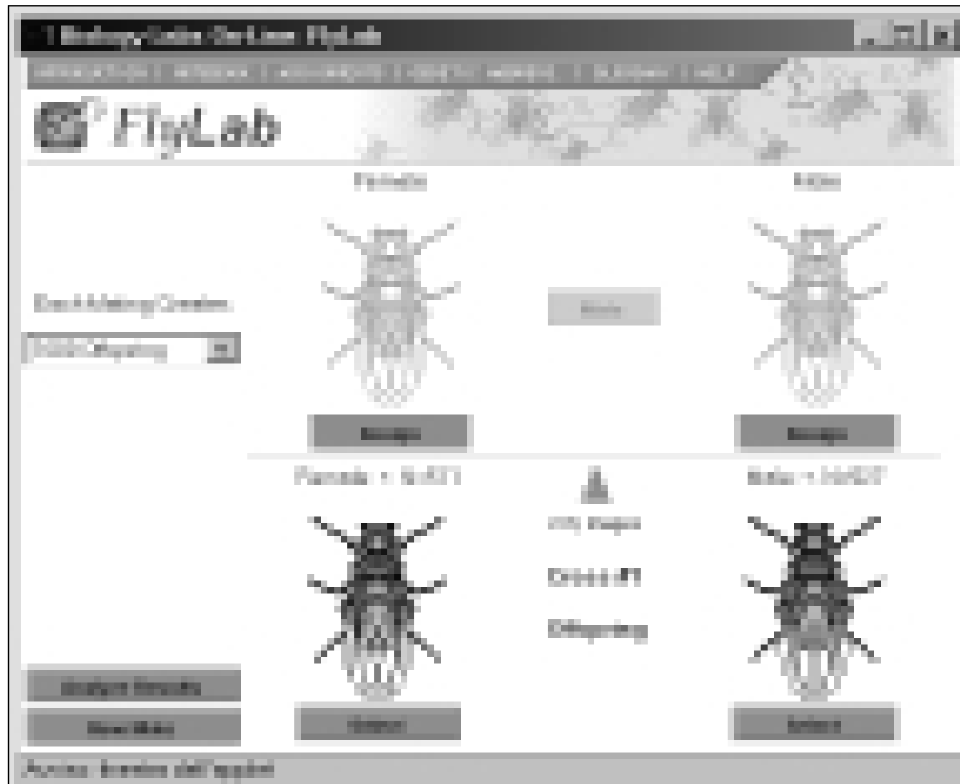


Figura 12
Un'applet in una pagina Web

Alcune delle aspettative riposte inizialmente nel linguaggio Java erano troppo ottimistiche e lo slogan "scrivi (il programma) una volta sola e fallo girare dovunque" diventò "scrivi (il programma) una volta sola e correggilo dappertutto", per i primi adepti che ebbero a che fare con implementazioni imperfette. Da allora, Java ha fatto molta strada e il linguaggio Java 2 e la sua libreria, rilasciati nel 1998, promettono di conferire stabilità allo sviluppo di Java e di mantenere la promessa "scrivi (il programma) una volta sola e fallo girare dovunque".

Poiché Java è stato pensato per Internet, ha due qualità che lo rendono molto adatto per i principianti: sicurezza e trasferibilità. Se si visita una pagina Web che contiene applet, queste si avviano automaticamente. È importante contare sul fatto che le applet sono intrinsecamente sicure. Se un'applet facesse qualcosa di male, come danneggiare dati o leggere informazioni personali sul vostro computer, vi trovereste in reale pericolo a ogni navigazione sul Web: un progettista senza scrupoli potrebbe impostare una pagina con del codice pericoloso, che si attiverebbe sulla vostra macchina appena visitate la pagina.

Al contrario, il linguaggio Java ha un assortimento di caratteristiche di sicurezza per garantire che non si possano scrivere applet nocive. Quale vantaggio ulteriore, queste caratteristiche aiutano anche a imparare il linguaggio più velocemente. La macchina virtuale Java può cogliere molti tipi di errori da principiante e segnalarli accuratamente (per contro, nel linguaggio C molti errori da principiante producono semplicemente programmi che agiscono in modo arbitrario e disorientante).

L'altro vantaggio di Java è la trasferibilità. Lo stesso programma Java opererà senza bisogno di modifiche su Windows, su UNIX, su Linux o su Macintosh. Anche questo è un requisito delle applet. Quando si visita una pagina Web, il server Web che distribuisce il contenuto della pagina non ha idea di quale computer stiate utilizzando per visualizzarla, ma vi restituisce semplicemente il codice trasferibile che è stato generato dal compilatore Java. Quindi, la macchina virtuale sul vostro computer esegue questo codice trasferibile. Anche qui esiste un vantaggio per lo studente: non è necessario imparare a scrivere programmi per sistemi operativi di computer diversi.

Allo stato attuale, Java si è già imposto come uno dei più importanti linguaggi per la programmazione generica e per l'apprendimento dell'informatica. Tuttavia, sebbene Java sia un buon linguaggio per principianti, non è perfetto per due ragioni.

Dal momento che Java non è stato progettato specificatamente per gli studenti, non è stato curato affinché fosse veramente semplice da utilizzare per scrivere programmi elementari ed è necessaria una certa dose di tecnicismi per scrivere anche il più semplice dei programmi. Per capire questi tecnicismi, bisogna conoscere qualcosa di programmazione. Questo non è un problema per un programmatore professionista con esperienze precedenti in un altro linguaggio di programmazione, ma per lo studente la mancanza di un percorso di apprendimento lineare è un aspetto negativo. Mentre imparate come programmare in Java, capiteranno occasioni in cui serviranno spiegazioni preliminari e dovrete attendere un capitolo successivo per avere tutti i particolari.

Inoltre, non potete sperare di imparare tutto di Java in un semestre. Di per sé, il linguaggio Java è relativamente semplice, ma Java contiene un'ampia raccolta di *pacchetti di libreria*, che sono necessari per scrivere utili programmi. Si tratta di pacchetti per la grafica, per la costruzione di interfacce utente, per la crittografia, per la connessione in rete, per l'audio, per la memorizzazione di database e per molti altri scopi.

Anche i programmatori esperti di Java non conoscono il contenuto di tutti i pacchetti, ma si limitano a usare quelli che servono per un particolare progetto. Usando questo libro, imparerete una grande quantità di nozioni sul linguaggio Java e sui pacchetti più importanti. Tenete presente che l'obiettivo principale non è quello di farvi imparare a memoria le minuzie di Java, ma di insegnarvi come ragionare sulla programmazione.

1.7 Prendere confidenza con il computer

Per molti lettori, questo potrebbe essere il loro primo corso di programmazione e potrebbero trovarsi a lavorare con un sistema che non conoscono. Se vi trovate in questa situazione, dovreste spendere un po'di tempo per acquisire dimestichezza con il computer. Dal momento che i sistemi variano notevolmente l'uno dall'altro, questo libro può solamente fornirvi una traccia dei passaggi che dovrete seguire. Usare un computer nuovo e sconosciuto può essere frustrante, specialmente se siete da soli. Cercate un corso di addestramento offerto dalla vostra Università, oppure chiedete semplicemente a un amico di farvi fare un breve giro di prova.

Passo 1. Login

Se usate il vostro computer di casa, probabilmente non occorre che vi preoccupiate di questo passaggio. I computer di un laboratorio, tuttavia, di solito non sono aperti a chiunque. Generalmente, l'accesso è limitato a chi ha pagato le tasse necessarie e a chi è ritenuto affidabile per non sconvolgere la configurazione. Probabilmente, per ottenere l'accesso al sistema avrete bisogno di un numero di account e di una parola d'ordine.

Passo 2. Individuare il compilatore Java

I sistemi di computer variano molto a questo proposito. Alcuni consentono di avviare il compilatore mediante la selezione di un'icona (osservare la Figura 13) o di un menu. In altri sistemi, bisogna usare la tastiera, per digitare un comando che avvia il compilatore. Nella maggioranza dei personal computer esiste un cosiddetto *ambiente integrato*, nel quale potete scrivere e provare i vostri programmi. In altri computer, è necessario prima lanciare un programma che funziona come un word processor, dove si possono inserire le istruzioni Java; quindi occorre avviare un altro programma, per convertirle nelle istruzioni in linguaggio macchina; infine, si fa girare l'interprete della macchina virtuale che le esegue.

Passo 3. Capire file e cartelle

In qualità di programmatori, scriverete programmi Java, li proverete e li perfezionerete. Individuerete un posto nel computer per conservarli e avrete bisogno di ritrovarli. I programmi sono memorizzati in *file*: un file Java è un contenitore di istruzioni Java. I file hanno un nome e le regole per i nomi validi sono diverse da un sistema all'altro.

Alcuni sistemi ammettono spazi nei nomi dei file e altri no. Alcuni distinguono fra lettere maiuscole e minuscole e altri no. La maggior parte dei compilatori Java esige che i file Java terminino con l'*estensione* `.java`; per esempio, `test.java`. I nomi dei

Figura 13
Una schermata iniziale con icone

file Java non possono contenere spazi e la differenza fra lettere maiuscole e minuscole è importante.

I file sono conservati in *cartelle* o *directory*. Si tratta di contenitori di file, che si possono annidare, ovvero inserire uno dentro l'altro. Una cartella può contenere sia file, sia altre cartelle, che a loro volta possono racchiudere più file e cartelle (osservare la Figura 14). Questa gerarchia può essere piuttosto estesa, specialmente nei computer connessi in rete, in cui alcuni file possono trovarsi sul vostro disco rigido e altri in un altro punto della rete. Se non vi dovete preoccupare di ciascuna diramazione della gerarchia, dovete invece prendere confidenza con il vostro ambiente locale. Sistemi operativi diversi adottano modi differenti per visualizzare file e cartelle. Alcuni usano una rappresentazione grafica, su cui è possibile spostarsi e fare clic con il mouse sulle icone delle cartelle. In altri sistemi, occorre immettere comandi per visitare o per ispezionare le varie posizioni.

Passo 4. Scrivere un programma semplice

Nel prossimo paragrafo, presenteremo un programma molto semplice. Vi occorrerà imparare come digitarlo, come eseguirlo e come correggere gli errori.

Passo 5. Salvare il vostro lavoro

Spenderete molte ore per digitare programmi Java e per perfezionarli. I file di programma che ne risulteranno saranno preziosi e dovreste trattarli come qualsiasi altro

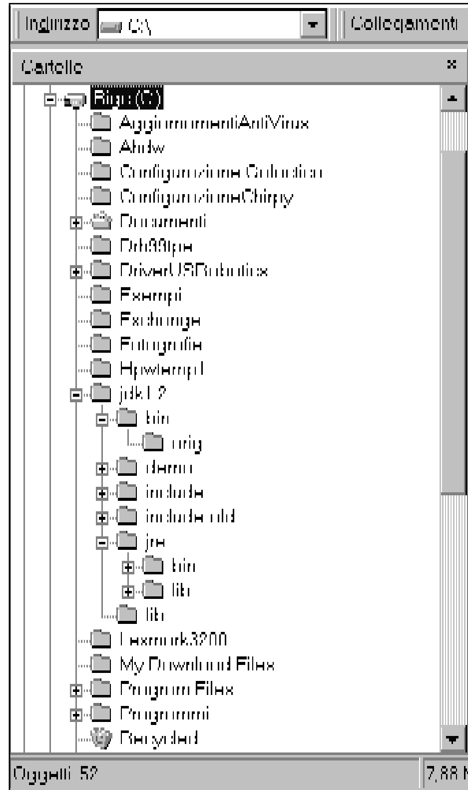


Figura 14
Una gerarchia di directory

oggetto di proprietà personale. Nel caso dei file di computer, una strategia di salvataggio scrupolosa è particolarmente importante, perché sono più fragili dei documenti cartacei o di altri oggetti tangibili. È facile eliminare un file per errore e, occasionalmente, si perdono file a causa di cattivi funzionamenti del computer. A meno di non avere un'altra copia, dovete ribatterne il contenuto. Dal momento che probabilmente non vi ricordate tutto, è plausibile che vi ritroviate a spendere nuovamente quasi tutto il tempo che avete impiegato la prima volta, per digitare e per provare il programma. Così facendo si perde tempo e si può rischiare di non rispettare una scadenza. Pertanto, è d'importanza cruciale imparare come salvaguardare i file, e prendere l'abitudine di farlo *prima* che avvenga un disastro. Potete fare copie di sicurezza o di *backup* dei file, salvando le copie su un disco floppy o in un'altra cartella.

Consigli per la produttività 1.1

Copie di backup

Per molte persone, fare copie di backup su dischi floppy è il metodo più facile e comodo. Se non potete farlo su dischi floppy, potete fare i salvataggi su cartelle separate del

vostro disco rigido, ma poi sarà necessario fare il backup di queste cartelle, solitamente su nastro, nell'eventualità che il disco rigido si guasti. Ecco alcuni punti da tenere presente:

- *Fare spesso il backup*. Salvare un file richiede solo pochi secondi e, se doveste perdere molte ore per ricreare un lavoro che avreste potuto salvare facilmente, vi detesterete. Io raccomando di salvare il lavoro ogni trenta minuti e ogni volta prima di eseguire un programma scritto da voi.
- *Utilizzare i supporti di backup secondo una sequenza*. Usare più dischi floppy per i salvataggi e usarli in sequenza. Ovvero, per prima cosa eseguire il salvataggio sul primo disco floppy e metterlo da parte. Poi, fare il salvataggio sul secondo dischetto, infine usare il terzo e quindi tornare al primo. In questo modo, avrete sempre tre salvataggi recenti: anche se uno dei dischi floppy è difettoso, potete usare uno dei rimanenti. Il giorno successivo, passate a una nuova serie di tre dischetti. Quanti salvataggi contemporanei bisogna tenere? Si tratta di un compromesso fra comodità e paranoia. Io suggerisco di impiegare sette serie di tre dischetti, una serie per ciascun giorno della settimana.
- *Fare il backup solo dei file sorgenti.* Il compilatore traduce i file che scrivete in file che contengono codice macchina. Non è necessario salvare i file con codice macchina, dal momento che li potete ricreare facilmente eseguendo di nuovo la compilazione. Indirizzate l'attività di salvataggio su quei file che contengono il vostro lavoro: in questo modo, i vostri dischi di backup non si riempiranno di archivi che non vi servono.
- *Fare attenzione alla destinazione del backup.* Il salvataggio consiste nel copiare file da un posto a un altro. È importante farlo correttamente, ovvero copiare dalla posizione del lavoro alla posizione del backup. Se lo fate nel modo sbagliato, sovrascriverete un file più recente con una versione più vecchia.
- *Controllare i backup di tanto in tanto.* Controllate accuratamente che i vostri salvataggi siano dove pensate. Non c'è nulla di più frustrante dello scoprire che i backup non si trovano lì quando se ne ha bisogno, soprattutto nel caso di un programma di salvataggio che memorizza i file su un supporto poco familiare (quale un nastro magnetico) o in formato compresso.
- *Rilassatevi prima di ripristinare.* Quando perdete un file e avete bisogno di ripristinarlo dal backup, probabilmente vi trovate in uno stato d'animo nervoso e poco felice. Fate un respiro profondo e riflettete sul processo di recupero prima di iniziare. Non è raro che un utente in preda all'agitazione distrugga l'ultimo salvataggio, nel tentativo di ripristinare un file danneggiato.

1.8 Compilare un semplice programma

Adesso siete pronti per scrivere e per eseguire il vostro primo programma Java. La scelta convenzionale, per il primo programma in assoluto con un linguaggio di programmazione nuovo, è un programma che visualizza un semplice saluto: "Hello, World!". Seguiamo la tradizione: ecco il programma "Hello, World!" in Java.

Programma Hello.java

```
public class Hello
{  public static void main(String[] args)
   {  System.out.println("Hello, World!");
   }
}
```

Spiegheremo questo programma fra un minuto. Per adesso, dovete creare un nuovo file di programma e chiamarlo `Hello.java`. Inserite le istruzioni, quindi compilate ed eseguite il programma, seguendo la procedura appropriata per il vostro compilatore.

Per inciso, Java *distingue fra maiuscolo e minuscolo*, quindi dovete inserire le lettere maiuscole e minuscole esattamente come appaiono nel listato del programma: non si può digitare `MAIN` oppure `PrintLn`. Per contro, Java ha una *disposizione a formato libero*. Gli spazi e le interruzioni di riga non sono importanti, salvo che per separare le parole, e potete ammassare quante parole è possibile in ciascuna riga:

```
public class Hello { public static void main(String[] args)
{ System.out.println("Hello, World!"); } }
```

Si può anche scrivere ciascuna parola o simbolo su una riga separata:

```
public
class
Hello
{
public
static
void
main
(
String
[
]
args
)
{
System
.
out
.
println
(
"Hello, World!"
)
;
}
}
```

Tuttavia, il buon gusto impone di disporre il programma in modo leggibile e, quindi, è meglio seguire la disposizione del listato.

Quando eseguite il programma, sullo schermo comparirà il messaggio:

```
Hello, World!
```

In alcuni sistemi, potrebbe essere necessario passare a una finestra diversa per trovare il messaggio.

Una volta visto come lavora il programma, è il momento di capire la sua struttura. La prima riga:

```
public class Hello
```

avvia una nuova *classe.* Le classi sono un concetto fondamentale in Java. Il loro ruolo principale è quello di fungere da“fabbriche”di *oggetti.* Gli oggetti sono un altro concetto centrale di Java e inizieremo a studiarli minuziosamente nel Capitolo 3. Per adesso, considerate un oggetto come un elemento che un programma può manipolare.

Nel nostro primo programma, non dobbiamo preoccuparci delle classi quali fabbriche di oggetti, vogliamo semplicemente visualizzare un messaggio. Java, come la maggior parte dei linguaggi di programmazione, esige che tutti gli enunciati del programma siano inseriti all'interno di *metodi*. (In molti altri linguaggi, i metodi si chiamano *funzioni* o *procedure*, ma in questo libro useremo la terminologia di Java.) Java, diversamente da molti altri linguaggi, esige anche che *ciascun* metodo venga inserito all'interno di una classe. Le classi sono i meccanismi centrali per organizzare il codice. Questo è il motivo per cui presentiamo la classe `Hello`, quale contenitore del metodo `main`.

La parola chiave `public` indica che la classe è utilizzabile dal "pubblico". Più avanti, incontrerete le caratteristiche `private`, che non lo sono.

A questo punto, dovete considerare semplicemente

```
Public class NomeClasse
{
...
}
```

come una parte indispensabile dell'"impiantistica" necessaria per scrivere qualsiasi programma Java. In Java, ciascun file di origine può contenere al massimo una classe pubblica, il cui nome deve corrispondere al nome del file che contiene la classe. Per esempio, la classe `Hello` *deve essere* contenuta in un file Hello.java. È molto importante che i nomi e le *lettere maiuscole e minuscole* corrispondano esattamente: potreste ricevere strani messaggi di errore se chiamate la classe `HELLO` o il file `hello.java`.

La costruzione

```
public static void main(String[] args)
{
}
```

definisce un *metodo,* chiamato `main` (principale). Un metodo è una raccolta di istruzioni di programma che descrivono come svolgere un determinato compito. Ciascuna applicazione Java deve avere un metodo `main`. La maggior parte dei programmi Java contiene altri metodi oltre a `main`, ma dovremo attendere fino al Capitolo 3 per imparare come scrivere gli altri.

Il *parametro* `String[] args` è una parte indispensabile del metodo `main` e contiene i cosiddetti argomenti della riga di comando (che non esamineremo fino al Capitolo 3). La parola chiave `static` indica che il metodo `main` non esamina o non modifica gli oggetti della classe `Hello`. Come vedrete nel Capitolo 3, la maggioranza dei metodi Java agisce sugli oggetti e i metodi cosiddetti `static` (statico) non sono comuni nei programmi Java di grosse dimensioni. Nondimeno, `main` deve sempre essere `static`. A questo punto, per ora considerate semplicemente

```
public class NomeClasse

{  public static void main(String[] args)
   {
 ...
    }
}
```

come un'altra parte dell'"impiantistica". Per il momento, inserite semplicemente tutte le istruzioni, che desiderate vengano eseguite, all'interno del metodo `main` di una classe.

Sintassi di Java
1.1 Programma semplice

```
public class NomeClasse
{  public static void main(String[] args)
   {   enunciati
    }
}
```

Esempio:

```
public class Greetings
{  public static void main(String[] args)
   {  System.out.println("Salute a te, Terrestre!");
   }
}
```

Obiettivo:
Eseguire un programma semplice

Le istruzioni o *enunciati* nel *corpo* del metodo `main`, ovvero gli enunciati racchiusi fra le parentesi graffe `{}`, sono eseguiti uno alla volta. Notare che ciascun enunciato termina con un punto e virgola `;`. Il nostro metodo ha un solo enunciato:

```
System.out.println("Hello, World!");
```

Questo enunciato stampa una riga di testo, vale a dire"Hello, World!". Tuttavia, esistono molti posti ai quali un programma può inviare questa stringa: una finestra, un file, oppure un computer connesso dall'altra parte del mondo. Pertanto, vi occorre specificare che la destinazione di questa stringa è l'*output standard*, ovvero una finestra di

terminale. In Java, la finestra di terminale è rappresentata da un oggetto chiamato `out`. Proprio come avete dovuto inserire il metodo `main` nella classe `Hello`, i progettisti della libreria di Java sono stati costretti a inserire `out` in una classe. L'hanno inserita nella classe `System` (sistema), che contiene oggetti e metodi per accedere alle risorse di sistema. Per utilizzare l'oggetto `out` nella classe `System`, dovete indicarlo nella forma `System.out`.

Per usare un oggetto, quale `System.out,` dovete specificare che cosa volete farne. In questo caso, desiderate stampare una riga di testo. Il metodo `println` (della classe `PrintStream`) svolge questo compito. Non è necessario implementare questo metodo, perché i programmatori che hanno scritto la libreria di Java l'hanno già fatto per noi, ma occorre chiamarlo, ovvero eseguire una *call*.

Quando chiamate un metodo in Java, dovete specificare tre elementi:

1. L'oggetto che volete usare (in questo caso, `System.out`).
2. Il nome del metodo che volete impiegare (in questo caso, `println`).
3. Una coppia di parentesi, che racchiudono eventuali altre informazioni necessarie per il metodo (in questo caso, `("Hello, World!")`).

Notare che i due punti singoli, in `System.out.println`, hanno due significati diversi. Il primo punto significa"individua l'oggetto `out` nella classe `System`". Il secondo punto sta per"applica il metodo `println` a quell'oggetto".

Una sequenza di caratteri, racchiusa fra virgolette, è detta *stringa:*

```
"Hello, World!"
```

Sintassi di Java
1.2 Metodo Call

`Oggetto.NomeMetodo (parametri)`

Esempio:

```
System.out.println("Buon giorno!");
```

Obiettivo:
Invocare il metodo di un oggetto e fornire eventuali parametri aggiuntivi.

È necessario inserire il contenuto della stringa all'interno delle virgolette, in modo che il compilatore sappia che intendete letteralmente `"Hello, World!"`. Esiste un motivo per questa regola: supponiamo di dover stampare la parola *main*. Se si racchiude la parola fra virgolette, `"main"`, il compilatore capisce che intendete la sequenza di caratteri `main`, non il metodo chiamato `main`. La regola è che dovete racchiudere semplicemente tutte le stringhe di testo fra virgolette, affinché il compilatore le consideri testo normale e non le interpreti quali istruzioni di programma. Si possono stampare anche valori numerici. Per esempio, l'enunciato seguente visualizza il numero 7:

```
System.out.println(3 + 4);
```

Il metodo `println` stampa una stringa o un numero e quindi inizia una riga nuova. Per esempio, la sequenza di enunciati seguente stampa due righe di testo:

```
System.out.println("Hello");
System.out.println("World!");
```

Esiste un secondo metodo, chiamato `print`, che potete utilizzare per stampare un elemento senza iniziare una nuova riga subito dopo. Per esempio, l'output di questi due enunciati:

```
System.out.print("00");
System.out.print(3 + 4);
```

è la riga singola seguente:

```
007
```

Errori comuni 1.1

Omettere i punti e virgola

In Java, ciascun enunciato deve terminare con un punto e virgola. Dimenticare di digitarlo è un errore comune che disorienta il compilatore, dal momento che si basa sul punto e virgola per stabilire dove termina un enunciato e ne inizia un altro. Per esempio, il compilatore considera

```
System.out.println("Hello")
System.out.println("World!");
```

un enunciato unico, come se fosse scritto in questo modo:

```
System.out.println("Hello") System.out.println("World!");
```

Pertanto, poi non è in grado di capire l'enunciato, perché non si aspetta la parola `System` che segue la parentesi di chiusura della stringa `"Hello"`. Il rimedio è semplice: scorrete tutti gli enunciati per verificare che terminino con un punto e virgola, proprio come controllereste che ciascun periodo in italiano termini con il punto.

Argomenti avanzati 1.1

Sequenze di escape

Supponete di voler visualizzare una stringa che contiene virgolette, come la seguente:

```
Hello, "World"!
```

Non potete scrivere:

```
System.out.println("Hello, "World"!");
```

Non appena il compilatore legge `"Hello, "`, deduce che la stringa sia terminata e poi rimane completamente disorientato dalla parola `World` seguita da due coppie di virgolette. Una persona probabilmente comprenderebbe che il secondo e il terzo paio di virgolette sono intese come parti della stringa, ma i compilatori ragionano a senso unico e, quindi, se per loro alla prima analisi la stringa non ha senso, si rifiutano semplicemente di proseguire e segnalano un errore. Pertanto, alla fine come fate a visualizzare le virgolette sullo schermo? Basta far precedere le virgolette all'interno della stringa da un carattere *barra rovesciata.* All'interno di una stringa, la sequenza \"indica le virgolette letterali, non la fine dei caratteri. Di conseguenza, l'enunciato corretto per la visualizzazione è:

```
System.out.println("Hello, \"World\"!");
```

Il carattere barra rovesciata si usa per definire un cosiddetto carattere di *escape*, o di uscita, e la sequenza di caratteri \"si chiama sequenza di escape. La barra rovesciata non rappresenta se stessa, ma si usa invece per codificare altri caratteri che, altrimenti, sarebbe difficile inserire in una stringa.

A questo punto, come vi comportate se volete stampare proprio una barra rovesciata (per esempio, per specificare il percorso di un file di Windows)? Bisogna inserirne due alla volta, in questo modo:

```
System.out.println("Il messaggio segreto è in C:\\Temp\\Secret.txt");
```

Questo enunciato stamperà:

```
Il messaggio segreto è in C:\Temp\Secret.txt
```

Un'altra sequenza di escape, che si usa talvolta, è \n, che indica una riga nuova o il carattere di ritorno carrello. Stampare un carattere di riga nuova produce l'inizio della nuova riga sullo schermo. Per esempio, l'enunciato:

```
System.out.print("*\n**\n***\n");
```

stampa i caratteri:

```
*
**
***
```

in tre righe separate. Naturalmente, si poteva ottenere lo stesso risultato anche con tre chiamate distinte di `println`.

Infine, le sequenze di escape sono utili per inserire caratteri internazionali in una stringa. Per esempio, supponete di volere stampare `"All the way to San José!"`, con la lettera accentata é. Se utilizzate una tastiera americana, potrebbe mancare il tasto per digitare questo carattere. Java utilizza uno schema di codifica, chiamato *Unicode,* per rappresentare i caratteri internazionali. Per esempio, il carattere é corrispon-

de alla codifica 00E9 in Unicode. Possiamo riprodurre questo carattere in una stringa, scrivendo \u, seguito dalla sua codifica Unicode:

```
System.out.println("All the way to San Jos\u00E9!");
```

Potete trovare i codici per i caratteri dell'inglese americano e dell'Europa occidentale nell'Appendice 3, e quelli per migliaia di caratteri nella guida [1].

1.9 Errori

Proviamo un po' il programma Hello.Vediamo che cosa succede con errori di battitura di questo tipo:

```
System.ouch.println("Hello, World!");
System.out.println("Hello, World!);
System.out.println("Hell, World!");
```

Nel primo caso, il compilatore protesterà, dicendo che non ha la più pallida idea di che cosa intendiate con `ouch`. L'esatta formulazione del messaggio di errore dipende dal compilatore, ma potrebbe assomigliare a"Simbolo ouch sconosciuto". Si tratta di un *errore di compilazione* o di un *errore di sintassi*, che si verifica quando vi è qualcosa di sbagliato secondo le regole del linguaggio e il compilatore lo trova. Quando il compilatore rinviene uno o più errori, rinuncia a tradurre il programma nel linguaggio macchina e, di conseguenza, vi ritrovate con un programma che non si può eseguire. Dovete allora rimediare all'errore e rifare la compilazione. Di fatto, il compilatore è piuttosto esigente e non è raro passare attraverso numerosi cicli di correzione degli errori di compilazione, prima di approdare alla prima compilazione completa.

Se il compilatore rileva un errore, non si limita a fermarsi e a rinunciare, ma tenterà di segnalare tutti gli errori che riesce a trovare, per permettervi di sistemarli tutti in una volta sola.Talvolta, tuttavia, un errore lo porta fuori strada. Probabilmente, questo succederà con l'errore nella seconda riga: poiché mancano le virgolette, il compilatore riterrà che i caratteri); facciano ancora parte della stringa. In questi casi, non è raro che il compilatore produca falsi messaggi di errore, che si riferiscono alle righe vicine. Quindi, dovrete considerare solo le segnalazioni di errore che vi sembrano fondate e quindi ricompilare. L'errore nella terza riga è di natura diversa. Il programma verrà compilato ed eseguito, ma l'output sarà sbagliato e visualizzerà:

```
Hell, World!
```

Questo è un *errore di esecuzione* o *errore logico,* che avviene quando il programma è corretto sintatticamente e fa qualcosa, ma non quello che ci si aspettava. Il compilatore non può rilevare il problema e tocca a voi programmatori sbloccare questo tipo di errore, eseguendo il programma e osservando attentamente il suo output.

Durante lo sviluppo dei programmi gli errori sono inevitabili. Quando un programma supera le poche righe, occorre una concentrazione sovrumana per digitarlo correttamente, senza incorrere in alcuna svista.Vi sorprenderete a trascurare punti e

virgola e virgolette più spesso di quanto vi piacerebbe ammettere, ma il compilatore scoverà questi errori per voi.

Gli errori logici sono più fastidiosi. Il compilatore non li rileva (di fatto, il compilatore convertirà allegramente qualsiasi programma finché la sintassi è corretta), ma il programma risultante farà qualcosa di sbagliato. È responsabilità dell'autore del programma collaudarlo e scoprire eventuali errori logici. Il collaudo dei programmi è un argomento importante, che incontrerete molte volte in questo corso. Un altro aspetto importante, per una buona qualità del lavoro, è la *programmazione difensiva*, che significa strutturare programmi e sviluppare processi in modo tale che un errore, situato in punto del programma, non inneschi esiti disastrosi.

Gli esempi di errori visti finora non erano difficili da diagnosticare e da correggere, ma, come imparerete tecniche di programmazione più sofisticate, parallelamente vi saranno anche più occasioni di errore. È un fatto spiacevole che trovare tutti gli errori di un programma sia molto difficile: anche vedendo che un programma produce un comportamento difettoso, potrebbe non essere affatto ovvio capire quale parte del programma lo causa e come sia possibile rimediarvi. Strumenti software specializzati, chiamati *debugger,* permettono di risalire attraverso un programma per trovare i *bug*, gli errori logici. In questo corso, imparerete come utilizzare un debugger efficacemente.

Notate che tutti questi errori sono diversi dal genere di quelli che di solito compaiono nelle operazioni di calcolo. Se sommate una colonna di numeri, potete dimenticare un segno meno o tralasciare un riporto, magari perché siete stanchi o annoiati. I computer non fanno questo tipo di errori: quando un computer somma numeri, restituisce il risultato corretto. In verità, le macchine possono incorrere in errori di overflow (superamento dei limiti di memoria o della capacità elaborativa) o di arrotondamento, proprio come fanno le calcolatrici tascabili, quando pretendete di eseguire calcoli il cui risultato va oltre le loro possibilità numeriche. Un errore di overflow o traboccamento avviene quando il risultato di un calcolo è troppo grande o troppo piccolo. Per esempio, la maggior parte dei computer e delle calcolatrici tascabili incorre nell'overflow quando si tenta di calcolare 10^{1000}. Un errore di arrotondamento si verifica quando non si può rappresentare un valore con precisione. Per esempio, 1/3 si può memorizzare nel computer come 0,3333333, un valore prossimo, ma non esattamente uguale a 1/3. Infatti, se calcolate $1 - 3 \times 1/3$, potete ottenere 0,0000001 invece di 0, quale risultato dell'errore di arrotondamento. Giudicheremo questi errori quali errori logici, perché un programmatore deve scegliere una procedura di calcolo più appropriata, per gestire correttamente l'overflow o l'arrotondamento.

In questo corso, presenteremo una strategia di gestione degli errori in tre parti. Per prima cosa, imparerete gli errori comuni e come evitarli. Poi, apprenderete strategie di programmazione difensiva, per ridurre al minimo la possibilità e gli effetti degli errori. Infine, imparerete strategie di debug, per risalire agli errori superstiti.

Errori comuni 1.2

Errori di ortografia

Se sbagliate accidentalmente l'ortografia di una parola, possono succedere strane cose e non sempre è facile capire che cosa è andato male dai messaggi di errore.

Ecco un buon esempio di come banali errori di ortografia possano causare noie:

```
public class Hello
{  public static void Main(String[] args)
   {  System.out.println("Hello, World!");
   }
}
```

Questo codice definisce un metodo chiamato `Main`. Il compilatore non lo ritiene uguale al metodo `main`, perché `Main` inizia con una lettera maiuscola e il linguaggio Java è *sensibile al maiuscolo e minuscolo.* Di fatto, le lettere maiuscole e minuscole sono considerate completamente diverse fra loro e, per il compilatore, `Main` non corrisponde a `main` più di quanto non corrisponda `rain`. Il compilatore compilerà allegramente il vostro metodo `Main`, ma, quando l'interprete Java sarà pronto per leggere il file compilato, si lamenterà dell'assenza del metodo `main` e rifiuterà di eseguire il programma. Naturalmente, il messaggio "missing main method" dovrebbe fornirvi un indizio di dove cercare l'errore.

Se ricevete un messaggio di errore che sembra indicare che il compilatore sia fuori strada, è il caso di controllare l'ortografia e le lettere maiuscole e minuscole. Tutte le parole chiave di Java utilizzano solo lettere minuscole. I nomi delle classi iniziano, generalmente, con una lettera maiuscola, mentre i nomi di metodi e di variabili iniziano con lettere minuscole. Se sbagliate l'ortografia del nome di un simbolo (per esempio, `ouch` al posto di `out`), il compilatore denuncerà un "undefined symbol" ("simbolo non definito"). Solitamente, questo messaggio di errore è un buon indizio di un possibile errore di ortografia.

1.10 Il processo di compilazione

Alcuni ambienti di sviluppo per Java sono molto pratici da utilizzare. Basta inserire il codice in una finestra, fare clic su un pulsante o su un menu per compilare, e fare clic su un altro pulsante o menu per eseguire il codice. I messaggi di errori compaiono in una seconda finestra e il programma si esegue in una terza. La Figura 15 mostra la schermata di un diffuso compilatore Java con queste caratteristiche. Con un ambiente simile siete completamente protetti dalle minuzie del processo di compilazione, mentre, in altri sistemi, dovete eseguire tutti i passaggi manualmente.

Anche se adoperate un ambiente Java comodo, è utile sapere che cosa succede dietro le quinte, principalmente perché conoscere il processo aiuta a risolvere i problemi quando qualcosa va storto.

In molti ambienti Java, dovete impostare un *progetto* per ciascun programma che volete scrivere. Le istruzioni per questo processo variano ampiamente fra i fornitori di compilatori e, quindi, dovete leggere la documentazione del vostro ambiente di sviluppo o chiedere al vostro istruttore.

Inserirete gli enunciati del programma in un editor di testo. L'editor memorizza il testo e gli fornisce un nome, del tipo `Hello.java`. Se la finestra dell'editor riporta un nome del tipo `Noname.java`, dovete cambiare il nome. È necessario *salvare* il file sul

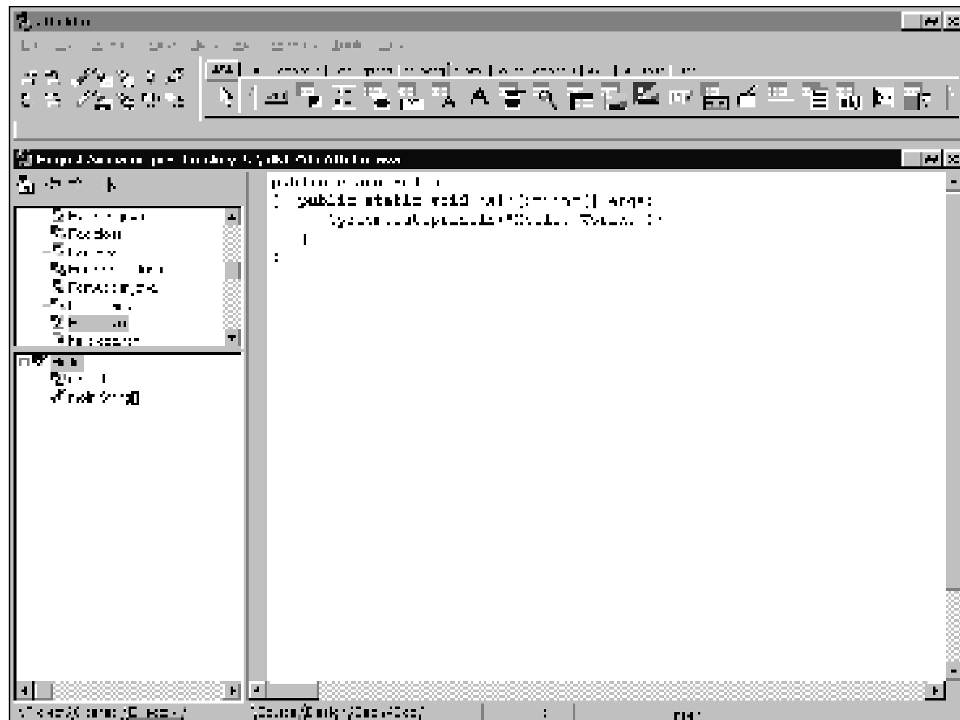


Figura 15
La schermata di un ambiente Java integrato

disco frequentemente, perché altrimenti l'editor conserva il testo solo in memoria. Se qualcosa non funziona nel computer e siete costretti a riavviarlo, il contenuto della RAM va perso, compreso il testo del vostro programma, mentre invece rimane qualsiasi cosa conservata in un disco rigido, o in un dischetto floppy, anche se dovete riavviare la macchina.

Quando compilate il programma, il compilatore traduce il *codice sorgente* Java (gli enunciati che avete scritto) nel cosiddetto *bytecode*, che consiste di istruzioni per la macchina virtuale e di altre informazioni, relative a come caricare il programma nella memoria prima di eseguirlo. Il bytecode di un programma si conserva in un file distinto, con estensione `.class`. Per esempio, il bytecode del programma Hello sarà conservato nel file `Hello.class`.

Il file del bytecode contiene la traduzione delle istruzioni che avete scritto, che non bastano per eseguire il programma. Per visualizzare una stringa in una finestra, è necessaria una certa dose di attività di basso livello. Gli autori delle classi `System` e `PrintStream` (che definiscono l'oggetto `out` e il metodo `println`) hanno implementato tutte le azioni necessarie e inserito i bytecode richiesti in una *libreria.* Una libreria è una raccolta di codice, programmato e tradotto da qualcun altro, pronto per l'utilizzo nel vostro programma. (Programmi più complicati si costruiscono mediante più file di bytecode e più librerie).

Un *interprete Java* carica il bytecode del programma che avete scritto, avvia il programma e carica i file di bytecode necessari della libreria, quando vengono richiesti.

Questi passaggi sono schematizzati nella Figura 16.

Gli strumenti Java più semplici impongono di richiamare manualmente l'editor, il compilatore e l'interprete, avviando questi programmi da un'interfaccia con una riga di comandi. Per modificare e per compilare il file Hello.java, e per eseguire il programma risultante, dovete digitare:

```
edit Hello.java
javac Hello.java
java Hello
```

In un ambiente Java più sofisticato, potete ottenere lo stesso effetto facendo clic su menu o su pulsanti nelle barre degli strumenti.

La vostra attività di programmazione si basa su questi passaggi. Iniziate nell'editor, dove scrivete il file di origine. Compilate il programma ed esaminate i messaggi di errore. Tornate all'editor e correggete gli errori di sintassi. Quando il compilatore va a buon fine, provate il file eseguibile. Se trovate un errore, potete adoperare il debugger per eseguire il programma una riga alla volta. Una volta individuata la causa dell'errore, tornate all'editor e la risolvete. Quindi compilate ed eseguite nuovamente, per vedere se l'errore è scomparso. In caso contrario, tornate all'editor. Questo è il cosiddetto *ciclo modifica-compila-correggi* (osservare la Figura 17), e vi trascorrerete una considerevole quantità di tempo nei mesi e negli anni a venire.

1.11 Un primo sguardo a oggetti e classi

Oggetti e classi sono concetti fondamentali per la programmazione in Java. Per padroneggiarli completamente occorrerà un po' di tempo, ma, dal momento che ciascun programma Java utilizza almeno un paio di oggetti e di classi, conviene averne una conoscenza di base fin dall'inizio.

Un *oggetto* è un'entità che potete manipolare nel vostro programma, generalmente mediante la chiamata di *metodi*. Per esempio, `System.out` si riferisce a un oggetto e abbiamo già visto come gestirlo, mediante la chiamata del metodo `println`. (In realtà, sono disponibili numerosi metodi diversi, tutti chiamati `println`: uno serve per stampare stringhe, uno per i numeri, uno per i numeri in virgola mobile e così via). Quando chiamate il metodo `println`, all'interno dell'oggetto si svolgono alcune attività, il cui effetto finale è che l'oggetto fa comparire il testo nella finestra della console. Per il momento, dovete considerare gli oggetti quali"scatole nere", fornite di un'*in-*

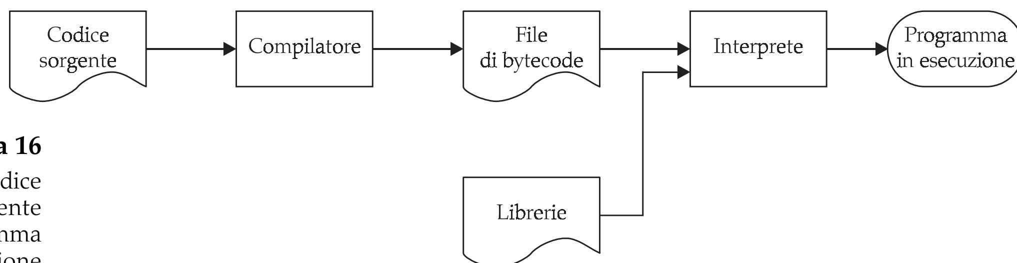


Figura 16
Dal codice sorgente al programma in esecuzione

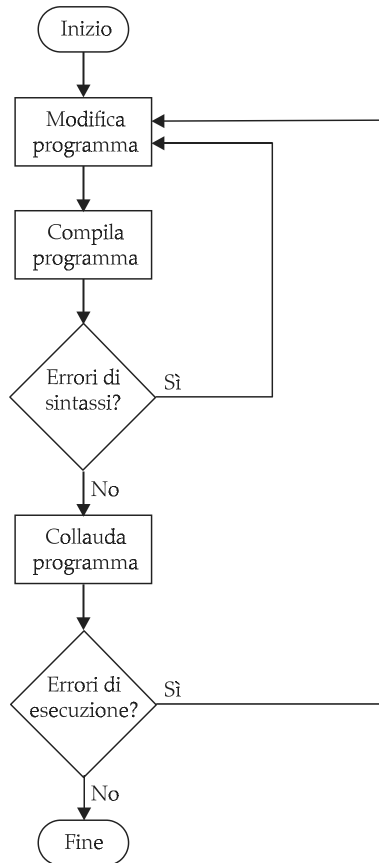


Figura 17
Il ciclo modifica-compila-correggi

terfaccia pubblica (ovvero, i metodi che potete chiamare) e di un'*implementazione* nascosta, ovvero il codice e i dati che sono necessari per fare funzionare i metodi.

Oggetti differenti supportano serie diverse di metodi. Per esempio, il metodo `println` si può applicare all'oggetto `System.out`, ma non all'oggetto stringa `"Hello, World!"`. Pertanto, questa chiamata sarebbe un errore:

```
"Hello, World!".println(); // Questa chiamata di metodo è un errore
```

La ragione è semplice: gli oggetti `System.out` e `"Hello, World!"` appartengono a *classi* diverse. Infatti, `System.out` è un oggetto della classe `PrintStream`, mentre `"Hello, World!"` è un oggetto della classe `String`. Si può applicare il metodo `println` a *qualsiasi* oggetto della classe `PrintStream`, ma la classe `String` non supporta il metodo `println`. La classe `String` prevede un buon numero di altri metodi, molti dei quali vedrete nel Capitolo 2. Per esempio, il metodo `length` conta il numero di caratteri in una stringa e si può applicare a qualsiasi oggetto del tipo `String`. Pertanto,

```
"Hello, World!".length(); // Questa chiamata di metodo va bene
```

è una chiamata di metodo corretta, che restituisce il numero 13, ovvero il numero di caratteri nell'oggetto stringa `"Hello, World!"` (le virgolette non vengono considerate).

Una *classe* ha quattro scopi:

1. Specifica i metodi che possiamo utilizzare per gli oggetti che appartengono alla classe.
2. È una fabbrica di oggetti.
3. È un contenitore di metodi statici e di oggetti statici.
4. Definisce i *particolari dell'implementazione*: la disposizione dei dati degli oggetti e il codice per i metodi.

Nel nostro primo programma, avete già visto il terzo (e meno importante) obiettivo: la classe `Hello` contiene il metodo statico `main`, mentre la classe `System` contiene l'oggetto statico `out`.

Per vedere come una classe possa fungere da fabbrica di oggetti, esaminiamo un'altra classe, `Rectangle`, situata nella libreria delle classi di Java. Gli oggetti di tipo `Rectangle` descrivono forme rettangolari, come quelle illustrate nella Figura 18.

Notate che un oggetto `Rectangle` non è una forma rettangolare, ma una serie di numeri che descrivono il rettangolo (osservare la Figura 19). Ciascun rettangolo è descritto dalle coordinate x e y del suo angolo superiore sinistro, dalla sua larghezza e dalla sua altezza. Per creare un nuovo rettangolo, è necessario specificare questi quattro valori. Per esempio, potete costruire un rettangolo con le coordinate dell'angolo superiore sinistro a (5, 10), con larghezza 20 e altezza 30, nel modo seguente:

```
new Rectangle(5, 10, 20, 30)
```

L'operatore `new` produce la creazione di un oggetto di tipo `Rectangle`. Il processo che crea un nuovo oggetto è detto *costruzione*. I quattro valori 5, 10, 20 e 30 rappresentano i *parametri di costruzione*. Classi diverse richiederanno parametri di costruzione differenti. Per esempio, per costruire un oggetto `Rectangle`, fornite quattro numeri che descrivono la posizione e le dimensioni del rettangolo. Per costruire un oggetto `Automobile`, potreste fornire il nome e l'anno del modello.

In realtà, alcune classi permettono di costruire oggetti in più maniere. Per esempio, potete ottenere un oggetto rettangolo anche senza fornire alcun parametro di costruzione (ma dovete sempre inserire le parentesi):

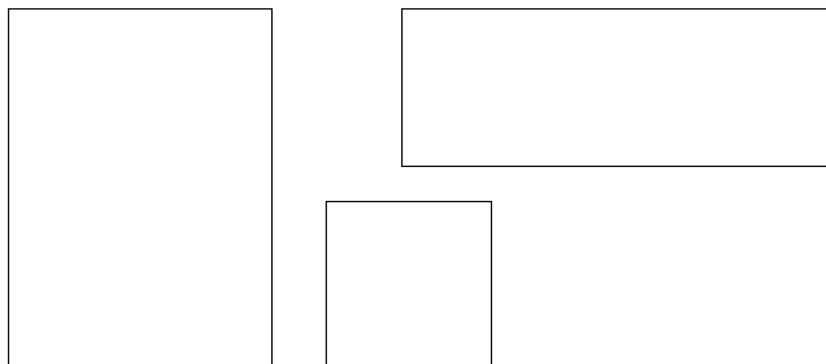


Figura 18
Forme rettangolari

Rectangle	
x	5
y	10
larghezza	20
altezza	30

Rectangle	
x	45
y	0
larghezza	30
altezza	20

Rectangle	
x	35
y	30
larghezza	20
altezza	20

Figura 19
Oggetti `Rectangle`

```
new Rectangle ()
```

Questo enunciato costruisce un rettangolo (piuttosto inutile) con l'angolo superiore sinistro posizionato all'origine (0, 0), con larghezza 0 e altezza 0. Una costruzione senza parametri viene detta *costruzione predefinita*.

Sintassi di Java
1.3 Costruzione di oggetti

`New` *`NomeClasse (parametri)`*

Esempio:

```
new Rectangle(5, 10, 20, 30);
new Automobile();
```

Obiettivo:
Costruire un nuovo oggetto, inizializzarlo tramite i parametri di costruzione e riportare un riferimento per l'oggetto costruito.

Per costruire qualsiasi oggetto, seguite questi passaggi:

1. Usate l'operatore `new`.
2. Date il nome della classe.
3. Fornite i parametri di costruzione (se esistono), all'interno delle parentesi.

Che cosa si può fare con un oggetto `Rectangle`? Non molto, per adesso. Nel Capitolo 4, imparerete come visualizzare rettangoli e altre figure in una finestra. Sapete già come stampare una descrizione dell'oggetto rettangolo nella finestra della console, mediante la semplice chiamata del metodo `System.out`:

```
System.out.println(new Rectangle(5, 10, 20, 30));
```

Questo codice stampa la riga:

```
java.awt.Rectangle[x=5, y=10, width=20, height=30]
```

Oppure, più specificatamente, questo codice crea un oggetto di tipo `Rectangle`, poi passa l'oggetto al metodo `println` e infine se ne dimentica.

Naturalmente, in genere, con un oggetto dovreste fare qualcosa di più che semplicemente crearlo, stamparlo e scordarvene. Per ricordare un oggetto, avete bisogno di conservarlo in una *variabile oggetto*. Una variabile oggetto è un luogo di deposito che conserva non l'oggetto reale, bensì informazioni sulla sua posizione (osservare la Figura 20).

Per creare una variabile oggetto, dovete fornire il nome della classe, seguito da un nome per la variabile, come in questo esempio:

```
Rectangle cerealBox;
```

Questo enunciato definisce una variabile oggetto, `cerealBox`. Il *tipo* di questa variabile è `Rectangle`. In Java, ciascuna variabile oggetto è di un tipo specifico. Per esempio, una volta definita la variabile `cerealBox` grazie all'enunciato precedente, da allora in poi, nel programma, la variabile dovrà sempre indicare un oggetto di tipo `Rectangle` e mai di tipo `Automobile` o `String`.

Tuttavia, finora la variabile `cerealBox` non fa riferimento ad alcun oggetto. È una variabile *non inizializzata* (osservare la Figura 21). Affinché `cerealBox` si riferisca a un oggetto, basta semplicemente impostarla su un altro riferimento di oggetto. Come ottenere un altro riferimento di oggetto? L'operatore `new` crea un riferimento verso un oggetto appena creato:

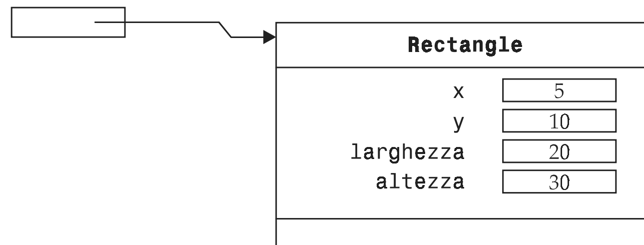


Figura 20
Una variabile oggetto che si riferisce a un oggetto

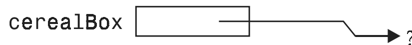


Figura 21 Una variabile oggetto non inizializzata

```
Rectangle cerealBox = new Rectangle(5, 10, 20, 30);
```

È essenziale ricordarsi che la variabile `cerealBox` *non contiene* l'oggetto, ma *si riferisce* all'oggetto. Infatti, potete avere due variabili oggetto che si riferiscono allo stesso oggetto:

```
Rectangle crispyCrunchyStuff = cerealBox;
```

Ora, potete accedere allo stesso oggetto `Rectangle` attraverso entrambe le variabili `cerealBox` e `crispyCrunchyStuff`, come illustrato nella Figura 22.

La classe `Rectangle` prevede oltre 50 metodi, alcuni utili e altri un po' meno. Per avere un'idea di che cosa significhi manipolare oggetti `Rectangle`, esaminiamo un metodo della classe `Rectangle`. Il metodo `translate` *sposta* un rettangolo per una certa distanza, nella direzione indicata dalle coordinate x e y. Per esempio:

```
cerealBox.translate(15, 25);
```

sposta il rettangolo di 15 unità nella direzione x e di 25 unità nella direzione y. Spostare un rettangolo non modifica la sua larghezza o la sua altezza, ma cambia la posizione dell'angolo superiore sinistro. La porzione di codice seguente:

```
Rectangle cerealBox = new Rectangle(5, 10, 20, 30);
cerealBox.translate(15, 25);
System.out.println(cerealBox);
```

stamperà questa riga:

```
java.awt.Rectangle[x=20, y=35, width=20, height=30]
```

Trasformiamo questo frammento di codice in un programma completo. Come nel caso del programma `Hello`, dovete seguire questi tre passaggi:

1. Inventate un nuova classe, per esempio `MoveRectangle`.
2. Fornite un metodo `main`.

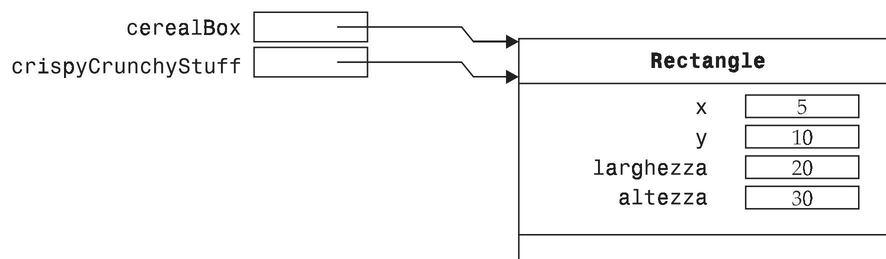


Figura 22 Due variabili oggetto che si riferiscono allo stesso oggetto

3. Inserite le istruzioni all'interno del metodo `main`.

Tuttavia, per questo programma, dovete eseguire un passaggio ulteriore: *importare* la classe `Rectangle` da un *pacchetto* ("package"), vale a dire da una raccolta di classi che hanno finalità simili. Tutte le classi nella libreria standard sono contenute in pacchetti. Per esempio, le classi `System` e `String` sono in un pacchetto chiamato `java.lang`, mentre la classe `Rectangle` appartiene al pacchetto `java.awt` (l'abbreviazione `awt` significa "Abstract Windowing Toolkit"). Il pacchetto `java.awt` contiene molte classi utili per disegnare finestre e forme grafiche.

Per usare la classe `Rectangle` dal pacchetto `java.awt`, inserite semplicemente la riga seguente all'inizio del vostro programma:

```
import java.awt.Rectangle;
```

Riguardo alle classi di `java.lang`, non avrete mai bisogno di importarle, perché tutte le classi di questo pacchetto sono importate automaticamente. Per esempio, potete utilizzare le classi `System` e `String` senza importarle.

Sintassi di Java
1.4 Importare una classe da un pacchetto

```
import Nomepacchetto.NomeClasse;
```

Esempio:

```
import java.awt.Rectangle;
```

Obiettivo:
Importare una classe da un pacchetto, per utilizzarla in un programma.

Pertanto, il programma completo è il seguente:

Programma MoveRectangle.java

```
import java.awt.Rectangle;

public class MoveRectangle
{  public static void main(String[] args)
   {  Rectangle cerealBox = new Rectangle(5, 10, 20, 30);
      cerealBox.translate(15, 25);
      System.out.println(cerealBox);
   }
}
```

Questo paragrafo ha presentato una prima introduzione agli oggetti e alle classi. Avete appreso che ciascun oggetto appartiene a una classe e che ciascuna classe definisce l'insieme di metodi che potete utilizzare con tutti gli oggetti di quella classe. Avete imparato anche come creare nuovi oggetti, mediante l'operatore `new`, come memoriz-

zare il riferimento a un oggetto in una variabile oggetto, e come importare un pacchetto. Nel Capitolo 2, imparerete come adoperare gli oggetti della classe `String`, e vedrete anche un certo numero di altre classi, che sono necessarie per leggere l'input della tastiera. Nel Capitolo 3, inizierete a implementare le vostre classi personali.

Argomenti avanzati 1.2

Importare le classi

Abbiamo visto qual è il metodo più chiaro e più semplice per importare classi dai pacchetti. Adoperate semplicemente un enunciato `import`, che indichi il pacchetto e la classe per ciascuna classe che volete importare, come in questo esempio:

```
import java.awt.Rectangle;
import java.awt.Point;
```

Ecco una scorciatoia che molti programmatori trovano conveniente: si possono importare *tutte* le classi, partendo dal nome di un pacchetto, mediante il seguente costrutto:

```
import nomepacchetto.*;
```

Per esempio, l'enunciato seguente importa tutte le classi del pacchetto `java.awt`:

```
import java.awt.*;
```

Si tratta di una forma meno complicata da digitare, ma non vogliamo usare questa forma nel libro per una semplice ragione. Se un programma importa più pacchetti e vi imbattete in un nome di classe sconosciuto, poi dovete cercare attraverso tutti i pacchetti per ritrovarla. Per esempio, supponiamo di vedere un programma con queste importazioni:

```
import java.awt.*;
import java.io.*;
```

In aggiunta, ipotizziamo di avere un nome di classe `Image`. In questo caso, non potete sapere se la classe `Image` appartenga al pacchetto `java.awt` o a `java.io`. Perché preoccuparsi di quale pacchetto si tratta? Occorre saperlo se volete usare la classe nel vostro programma. Questo problema non si presenta se si usa un enunciato `import` esplicito:

```
import java.awt.Image;
```

Notate che *non si possono* importare più pacchetti mediante un unico enunciato. Per esempio, questo è un errore di sintassi:

```
import java.*.*; // Errore
```

Potete evitare tutti gli enunciati `import` se utilizzate il nome *completo* (costituito da entrambi i nomi del pacchetto e della classe) ogni volta che adoperate una classe, come in questo esempio:

```
java.awt.Rectangle cerealBox =
    new java.awt.Rectangle (5, 10, 20, 30);
```

È un sistema abbastanza noioso e non troverete molti programmatori che lo usano.

1.12 Algoritmi

Presto imparerete come programmare calcoli e processi decisionali in Java. Ma prima di esaminare, nel prossimo capitolo, i meccanismi per implementare le operazioni, consideriamo il processo di pianificazione che precede il loro inserimento.

Vi può capitare di scorrere inserzioni pubblicitarie che offrono un servizio computerizzato, a pagamento, per incontrare il partner giusto. Immaginiamo come può funzionare: compilate un modulo e lo spedite; altri fanno lo stesso; infine, i dati sono elaborati da un programma informatico. È ragionevole presumere che il computer possa svolgere il compito di trovare il migliore accoppiamento per voi? Supponiamo che vostro fratello più giovane, invece del computer, abbia tutti i moduli sul suo tavolo. Quali istruzioni potreste fornirgli? Non potete dirgli "Individua la persona dall'aspetto migliore, del sesso opposto, che ama i pattini in linea e navigare in Internet". Non esistono standard obiettivi per un buon aspetto e l'opinione di vostro fratello (o quella di un programma informatico che esamina le foto digitalizzate) probabilmente sarà diversa dalla vostra. Se non potete fornire le istruzioni a qualcuno affinché risolva il problema manualmente, il computer non può assolutamente venirne a capo per magia: il calcolatore può fare solo quello che potreste fare anche voi, manualmente. Lo fa solo in modo più rapido, senza annoiarsi, né esaurirsi.

Adesso considerate questo problema di investimento:

> Depositate venti milioni in un conto bancario, che produce il 5% di interessi all'anno. Gli interessi sono capitalizzati annualmente. Quanti anni occorrono affinché il saldo del conto raddoppi la cifra iniziale?

Potreste risolvere questo problema manualmente? Certamente, è possibile. Per esempio, trascorso il primo anno, si guadagna un milione (il 5% di venti milioni), che viene sommato al conto. L'anno successivo, gli interessi sono di £. 1.050.000 (il 5% di 21 milioni) e il saldo arriva a £. 22.050.000. Si può proseguire in questo modo e la tabella seguente illustra il meccanismo:

Anno	Saldo
0	£. 20.000.000
1	£. 21.000.000
2	£. 22.050.000
3	£. 23.152.500
4	£. 24.310.125
…	…

Si può procedere finché il saldo arriva a 40 milioni, quindi si guarda nella colonna dell'anno e si ottiene la risposta al quesito.

Naturalmente, eseguire questo calcolo è decisamente noioso. Potreste tentare di farlo fare a vostro fratello più giovane. Parlando seriamente, il fatto che un calcolo sia difficile o noioso è irrilevante per il computer. Queste macchine vanno molto bene per eseguire calcoli ripetitivi in modo rapido e impeccabile. Ciò che conta per il computer (e per vostro fratello minore) è la presenza di un approccio sistematico per trovare la soluzione. La risposta si può trovare solamente seguendo una serie di passaggi che non implichi congetture. Ecco una serie di operazioni che risponde a questi criteri:

Passaggio 1. Iniziare la tabella.

Anno	Saldo
0	£. 20.000.000

Passaggio 2. Ripetere i passi da 2a a 2c, fin tanto che il saldo è inferiore a 40 milioni.
Passaggio 2a. Aggiungere una riga nuova alla tabella.
Passaggio 2b. Nella colonna 1 della nuova riga, inserire il valore dell'anno precedente, incrementato di 1.
Passaggio 2c. Nella colonna 2 della nuova riga, inserire il valore del saldo precedente, moltiplicato per 1,05 (ovvero 1 più il 5%).
Passaggio 3. Leggere l'ultima cifra nella colonna dell'anno e riportarla quale numero di anni necessari per raddoppiare l'investimento.

Chiaramente, questi passaggi non sono ancora espressi in un linguaggio comprensibile per il computer, ma presto impareremo come formularli in Java. Ciò che conta è che il metodo descritto:

- non sia ambiguo;
- sia eseguibile;
- arrivi a una conclusione.

Il metodo *non è ambiguo*, perché contiene precise istruzioni su che cosa bisogna fare in ciascun passaggio e su dove andare in seguito: non c'è spazio per congetture o inventiva. Il metodo è *eseguibile*, perché ciascun passaggio si può eseguire concretamente. Se, invece del tasso fisso del 5% all'anno, ci chiedessero di adoperare un tasso variabile, dipendente da fattori economici degli anni a venire, il nostro metodo non sarebbe eseguibile, perché non vi è alcuna maniera, per chiunque, di sapere quale sarà il tasso di interesse. Infine, il calcolo a un certo punto raggiungerà una conclusione. Durante

ciascun passaggio il saldo cresce almeno di un milione, quindi, alla fine, dovrà raggiungere i 40 milioni di lire.

Una soluzione tecnica, priva di ambiguità, eseguibile e che arriva a una conclusione, si chiama *algoritmo.* Dal momento che abbiamo individuato un algoritmo per risolvere il problema dell'investimento, possiamo arrivare alla stessa soluzione mediante il computer. La presenza di un algoritmo è un prerequisito essenziale per programmare un compito da svolgere. Talvolta, individuare un algoritmo è molto semplice, altre volte richiede inventiva o pianificazione. Se non riuscite a trovarne uno, non potete usare il computer per risolvere il vostro problema. Prima di iniziare a programmare, è necessario che vi assicuriate di avere un algoritmo e di capirne i passaggi.

Riepilogo del capitolo

1. I computer eseguono operazioni molto elementari, in rapida successione. La sequenza di operazioni si chiama programma. Mansioni differenti, quali calcolare il saldo di un libretto di assegni, stampare una lettera o eseguire un gioco, richiedono programmi diversi. I programmatori producono programmi, affinché il computer risolva nuove attività.
2. L'unità di elaborazione centrale (CPU) del computer esegue un'operazione alla volta. Ciascuna operazione specifica come elaborare i dati, come trasferirli all'interno o all'esterno della CPU, oppure quale operazione scegliere successivamente.
3. Per l'elaborazione, i valori dei dati si possono trasferire nella CPU dai loro depositi o da dispositivi di input, quali la tastiera, il mouse, o tramite un collegamento per comunicazioni. Una volta elaborate, le informazioni vengono rispedite dalla CPU ai loro depositi, oppure inviate a dispositivi di output, tipo uno schermo o una stampante.
4. I dispositivi che conservano i dati comprendono la memoria ad accesso casuale (RAM) e la memoria secondaria. La RAM è più veloce, ma costosa e perde tutti i dati quando si spegne il computer. I dispositivi di memoria secondaria utilizzano tecnologie magnetiche o ottiche per registrare le informazioni. Il tempo di accesso è più lento, ma i dati sono mantenuti senza necessità di erogazione elettrica.
5. I programmi del computer sono conservati nella forma di istruzioni macchina, secondo una codifica che dipende dal tipo di processore. Scrivere direttamente i codici delle istruzioni sarebbe improponibile per programmatori umani. I ricercatori informatici hanno trovato il modo di rendere più agevole questa attività, grazie all'impiego del linguaggio assemblatore e di linguaggi di programmazione ad alto livello. Pertanto, si scrive il programma in uno di questi linguaggi e un programma speciale del computer lo traduce nella sequenza corrispondente di istruzioni macchina. Le istruzioni del linguaggio assemblatore variano in funzione del processore o del tipo di macchina virtuale. Per contro, i linguaggi di alto livello sono indipendenti dal processore e lo stesso programma si può tradurre affinché operi su molti tipi di processori differenti, forniti da costruttori diversi.
6. Gli informatici progettano linguaggi di programmazione per impieghi diversi. Alcuni linguaggi sono pensati per un utilizzo specifico, per esempio l'elaborazione di database. In questo libro, utilizzeremo Java, un linguaggio per uso generale e adatto per un'ampia gamma di compiti di programmazione.

7. Riservate un po' di tempo per acquisire dimestichezza con il computer e con il compilatore Java, che impiegherete per le esercitazioni del vostro corso. Prima di incappare in un disastro, sviluppate anche una strategia per disporre di copie di backup.
8. I programmi Java contengono una o più classi. Le classi contengono definizioni di metodi. Un metodo è una sequenza di istruzioni che descrive come svolgere un'operazione. Ciascuna applicazione Java contiene almeno una classe con un metodo chiamato `main`.
9. Le classi sono fabbriche di oggetti. Possiamo costruire un nuovo oggetto in una classe tramite l'operatore `new`. Inserirete i riferimenti all'oggetto nelle variabili oggetto. Si possono avere più riferimenti per lo stesso oggetto.
10. Le classi Java sono raggruppate in pacchetti. Se utilizzate una classe di un pacchetto diverso da `java.lang`, dovete importare la classe.
11. Per un programmatore, gli errori sono una costante della vita. Gli errori di sintassi sono costrutti errati che non rispondono alle regole del linguaggio di programmazione. Vengono rilevati dal compilatore e il programma non viene generato. Gli errori logici sono costrutti che si possono tradurre in un programma eseguibile, ma il programma risultante non svolge l'azione prevista dal programmatore. Il programmatore ha la responsabilità di esaminare e di provare il programma per salvaguardarlo dagli errori logici.
12. Un programma, detto compilatore, traduce in bytecode i programmi Java. In un ulteriore passaggio, un altro programma, chiamato interprete, legge il bytecode del programma, insieme con il bytecode già tradotto precedentemente e destinato a funzioni di input/output e ad altri servizi che sono richiesti dal programma.
13. Un algoritmo è una descrizione di passaggi per risolvere un problema, privi di ambiguità, eseguibili e che prevedono una conclusione. In pratica, la descrizione non lascia spazio per interpretazioni diverse, i passaggi si possono eseguire concretamente e si garantisce la restituzione di un risultato, al termine di un periodo di tempo circoscritto. Allo scopo di risolvere un problema mediante il computer, dovete individuare un algoritmo per giungere alla soluzione.

Ulteriori letture

[1] The Unicode Cosortium, *The Unicode Standard Worldwide Charachter Encoding, Version 2.0*, Addoson-Wesley, 1996.

Classi, oggetti e metodi presentati nel capitolo

Ecco l'elenco di classi, metodi, variabili statiche e costanti presentati in questo capitolo. Per maggiori informazioni, consultare la documentazione nell'Appendice 2.

```
java.awt.Rectangle
   translate
java.io.PrintStream
   print
   println
java.lang.String
   length
java.lang.System
   out
```

Esercizi di ripasso

Esercizio R1.1. Spiegare la differenza fra adoperare un programma e programmare un computer.

Esercizio R1.2. Che cosa distingue un computer da un comune elettrodomestico?

Esercizio R1.3. Quali parti di un computer possono conservare il codice dei programmi? E quali possono conservare i dati dell'utente?

Esercizio R1.4. Quali parti di un computer servono per fornire le informazioni all'utente? Quali accettano l'input dell'utente?

Esercizio R1.5. Classificare i dispositivi di memoria che possono fare parte del sistema di un computer, ordinati per *(a)* velocità, *(b)* costo e *(c)* capacità di immagazzinamento.

Esercizio R1.6. Descrivere i vantaggi della rete di computer nel laboratorio informatico della vostra facoltà. Un computer del laboratorio a quali altri computer si connette?

Esercizio R1.7. Che cos'è la macchina virtuale Java?

Esercizio R1.8. Che cos'è un'applet?

Esercizio R1.9. Spiegare due vantaggi dei linguaggi di programmazione ad alto livello, rispetto al codice Assembler.

Esercizio R1.10. Elencare i linguaggi di programmazione citati in questo capitolo.

Esercizio R1.11. Che cos'è un ambiente di programmazione integrato?

Esercizio R1.12. Nel vostro computer personale, oppure in quello del laboratorio, trovare la posizione esatta (cartella o nome della directory) di:

- Il file di esempio `Hello.java`, che avete scritto con l'editor.
- L'interprete `java.exe`.
- Il file della libreria `rt.jar`, che contiene la libreria di esecuzione.

Esercizio R1.13. Spiegare il ruolo particolare del carattere di escape \ nelle stringhe di caratteri Java.

Esercizio R1.14. Scrivere tre versioni del programma `Hello.java`, con tre errori di sintassi diversi. Scriverne una versione con un errore logico.

Esercizio R1.15. In che modo si scoprono gli errori di sintassi? Come si scoprono gli errori logici?

Esercizio R1.16. Scrivere un algoritmo per risolvere il seguente quesito: si accende un conto bancario con 20 milioni di lire. Gli interessi sono capitalizzati alla fine di ciascun mese, secondo un tasso del 6 per cento annuale, che corrisponde allo 0,5 per cento mensile. All'inizio di ciascun mese, dopo l'accredito degli interessi, si preleva un milione di lire per fronteggiare le spese di studio. Dopo quanti anni il conto si esaurisce?

Esercizio R1.17. Esaminare il problema dell'esercizio precedente. Supponiamo che le cifre (20 milioni, 6 per cento e 1 milione) siano modificabili dall'utente. Esistono valori per cui il ciclo, basato sull'algoritmo che avete sviluppato, potrebbe non terminare mai? Se così, cambiare l'algoritmo per assicurarsi che il processo termini in qualunque caso.

Esercizio R1.18. Si può calcolare il valore di π secondo la formula seguente:

$$\pi/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - \ldots$$

Scrivere un algoritmo per calcolare il valore di π. Dal momento che la formula è una serie infinita di operazioni e che un algoritmo deve terminare dopo un numero finito di passaggi, interrompere il ciclo quando il risultato raggiunge sei cifre significative.

Esercizio R1.19. Supponete di incaricare vostro fratello minore del salvataggio del vostro lavoro. Scrivete una serie di istruzioni particolareggiate su come dovrebbe svolgere questo compito. Specificate la frequenza dei salvataggi e quali file è necessario copiare, da quali cartelle e in quali dischi floppy. Spiegate come verificare che il backup venga svolto correttamente.

Esercizio R1.20. Spiegare la differenza fra un oggetto e il riferimento a un oggetto.

Esercizio R1.21. Spiegare la differenza fra un oggetto e una classe.

Esercizio R1.22. Spiegare la differenza fra un oggetto e un metodo.

Esercizi di programmazione

Esercizio P1.1. Scrivere un programma per visualizzare sullo schermo del terminale il vostro nome all'interno di un rettangolo, come nell'esempio seguente:

```
+ —— +
¦ Dave ¦
+ —— +
```

Fate quanto possibile per rendere le linee con i caratteri `¦ - +`.

Esercizio P1.2. Scrivere un programma per stampare un albero di Natale:

```
    /\
   /  \
  /    \
 /      \
 ————
   "  "
   "  "
   "  "
```

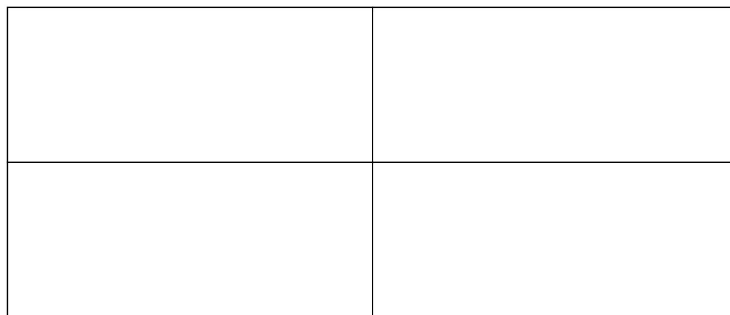
Ricordatevi di usare le sequenze di escape per rappresentare i caratteri `\` e `"`.

Esercizio P1.3. Scrivere un programma per calcolare la somma dei primi 10 numeri interi positivi: 1 + 2 + … + 10. *Suggerimento:* Utilizzare questo schema:

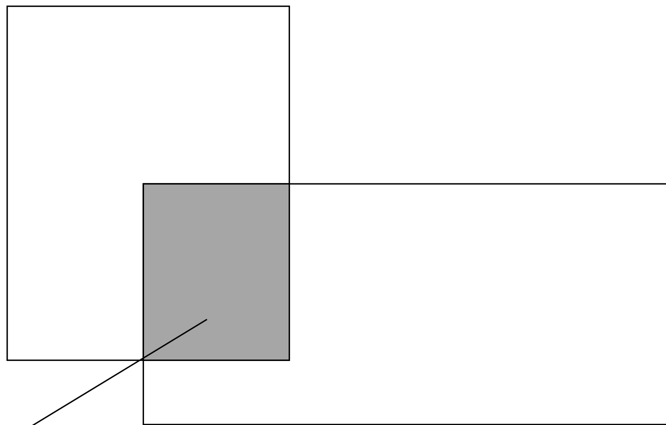
```
public class Sum10
{ public static void main(String[] args)
  { System.out.println(                    );
  }
}
```

Esercizio P1.4. Scrivere un programma per calcolare la somma dei numeri reciproci 1/1 + 1/2 + … + 1/10. È più difficile di quanto sembri. Provate a scrivere il programma e controllate il risultato con una calcolatrice tascabile. Probabilmente, il risultato non sarà corretto. Poi scrivete i numeri nella forma in *virgola mobile*: `1,0, 2,0, ... 10,0` ed eseguite nuovamente il programma. Siete in grado di spiegare la differenza dei risultati? Esamineremo questo fenomeno nel Capitolo 2.

Esercizio P1.5. Scrivere un programma che costruisca un oggetto `Rectangle`, lo stampi e che quindi lo sposti e lo stampi per altre tre volte, in modo che, se i rettangoli fossero disegnati, formerebbero un unico grande rettangolo:



Esercizio P1.6. Il metodo `intersection` calcola l'intersezione di due rettangoli, ovvero il rettangolo formato dalla sovrapposizione parziale di altri due rettangoli:



Intersezione

Il metodo si chiama in questo modo:

```
Rectangle r3 = r1.intersection(r2);
```

Scrivere un programma che costruisca due oggetti rettangolo, li stampi e che quindi stampi la loro intersezione. Che cosa succede quando i rettangoli non si sovrappongono?