Capitolo 1

# Introduzione al linguaggio

Java è un moderno linguaggio di programmazione le cui origini risalgono al 1991, quando presso Sun Microsystems un team di programmatori, formato principalmente da James Gosling, Patrick Naughton, Chris Warth, Ed Frank e Mike Sheridan, lavorò al suo sviluppo. Inizialmente il linguaggio fu chiamato OAK (in onore della quercia che Gosling vedeva dalla finestra del suo ufficio, infatti *oak* in inglese significa quercia) e fu sviluppato in seno a un progetto chiamato Green Project.

Lo scopo del progetto era quello di dotare svariati dispositivi elettronici di consumo di un meccanismo tramite il quale fosse possibile far eseguire, in modo indipendente dalla loro differente architettura, i programmi scritti per essi.

Questo meccanismo si sarebbe dovuto concretizzare nella scrittura di un componente software, denominato *macchina virtuale*, da implementare per il particolare hardware del dispositivo ma che, tuttavia, sarebbe stato in grado di far girare il codice intermedio, denominato *bytecode*, generato da un compilatore del linguaggio Java.

Sostanzialmente l'obiettivo era quello di permettere agli sviluppatori di scrivere i programmi una sola volta e con la certezza che sarebbe stato possibile eseguirli dappertutto senza alcuna modifica sull'hardware destinatario (da qui lo slogan WORA, *Write Once Run Anywhere*).

Nel maggio del 1995 fu completato lo sviluppo di OAK, con l'annuncio alla conferenza Sun World '95. Con l'occasione, visto che il nome OAK apparteneva già a un altro linguaggio di programmazione, si decise di cambiare il nome del linguaggio di SUN in Java.

## In questo capitolo

- **Paradigmi di programmazione**
- **Elementi di un ambiente Java**
- **Il primo programma Java**
- **Compilazione ed esecuzione del codice**
- **Problemi di compilazione ed esecuzione?**

**CURIOSITÀ**

Probabilmente il nome Java fu deciso durante un incontro tra gli sviluppatori in un bar mentre bevevano caffè americano; infatti *java* è un termine utilizzato nello slang per indicare una bevanda fatta con miscele di chicchi di caffè.

Successivamente, nel 1996, alla prima JavaOne Developer Conference, venne rilasciata la versione 1.0 del JDK (*Java Development Kit*). A partire da quel momento iniziò una capillare e profonda diffusione di Java che è diventato un linguaggio di programmazione ampiamente diffuso e utilizzato per programmare ogni tipo di dispositivi (dai cellulari ai computer, agli elettrodomestici e così via) dotati di una macchina virtuale.
Il successo di Java aumentò notevolmente con l'esplosione di Internet e del Web dove, all'epoca, non esistevano programmi che permettessero di fornire contenuto dinamico alle pagine HTML (se si escludevano gli script CGI). Con Java infatti fu possibile, attraverso programmi detti *applet*, gestire nel Web contenuti dinamici e allo stesso tempo indipendenti dal sistema operativo e dal browser sottostante. Dopo di ciò il successo di Java fu inarrestabile e attorno al linguaggio furono create molteplici tecnologie, per esempio quelle per l'accesso indipendente ai database (JDBC), per lo sviluppo lato server del Web (Servlet/JSP/JSF) e così via. Al momento in cui scriviamo la versione ufficiale del linguaggio è la 1.8 (chiamata 8.0 per motivi di marketing), disponibile a partire dalla primavera del 2014.

**NOTA STORICA**

Sun Microsystems era una società multinazionale, produttrice di software e di hardware, fondata nel 1982 da tre studenti dell'università di Stanford: Vinod Khosla, Andy Bechtolsheim e Scott McNealy. Il nome è infatti l'acronimo di *Stanford University Network*. Nel gennaio del 2010 Sun è stata acquistata dal colosso informatico Oracle Corporation per la considerevole cifra di 7,4 miliardi di dollari. Tra i prodotti software ricordiamo il sistema operativo Solaris e il filesystem di rete NFS, mentre tra i prodotti hardware le workstation e i server basati sui processori RISC SPARC.

## Paradigmi di programmazione

Un paradigma, o stile di programmazione, indica un determinato modello concettuale e metodologico, offerto in termini concreti da un linguaggio di programmazione, al quale fa riferimento un programmatore per la progettazione e scrittura di un programma informatico e dunque per la risoluzione del suo particolare problema algoritmico. Si conoscono molti differenti paradigmi di programmazione, ma quelli che seguono ne rappresentano i più comuni.

- Il *paradigma procedurale*, dove l'unità principale di programmazione è, per l'appunto, la procedura o la funzione che ha lo scopo di manipolare i dati del programma. Questo paradigma è talune volte indicato anche come *imperativo* perché consente di costruire un programma indicando dei comandi (assegna, chiama una procedura, esegui un loop e così via) che esplicitano quali azioni si devono eseguire, e in quale ordine, per risolvere un determinato compito. Questo paradigma si basa dunque su due aspetti di rilevo: il primo è riferito al cambiamento di stato del programma che è causa delle istruzioni eseguite (si pensi al cambiamento del valore di una variabile in un determinato tempo durante l'esecuzione del programma); il secondo è inerente

allo stile di programmazione adottato che è orientato al "come fare o come risolvere" piuttosto che al "cosa si desidera ottenere o cosa risolvere". Esempi di linguaggi che supportano il paradigma procedurale sono FORTRAN, COBOL, Pascal e C.

- Il *paradigma ad oggetti*, dove l'unità principale di programmazione è l'oggetto (nei sistemi basati sui prototipi) oppure la classe (nei sistemi basati sulle classi). Questi oggetti, definibili come virtuali, rappresentano, in estrema sintesi, astrazioni concettuali degli oggetti reali del mondo fisico che si vogliono modellare. Questi ultimi possono essere oggetti più generali (pensate a un computer, per esempio) oppure oggetti più specifici, ovvero maggiormente specializzati (per esempio una scheda madre, una scheda video e così via). Noi utilizziamo tali oggetti senza sapere nulla della complessità con cui sono costruiti e comunichiamo con essi attraverso l'invio di messaggi (sposta il puntatore, digita dei caratteri) e mediante delle interfacce (il mouse, la tastiera). Inoltre, essi sono dotati di attributi (velocità del processore, colore del case e così via) che possono essere letti e, in alcuni casi, modificati. Questi oggetti reali vengono presi come modello per la costruzione di sistemi software a oggetti, dove l'oggetto (o la classe) avrà metodi per l'invio di messaggi e proprietà che rappresenteranno gli attributi da manipolare. Esempi di linguaggi che supportano il paradigma ad oggetti sono: Java, C#, C++, JavaScript, Smalltalk e Python.

**TERMINOLOGIA**

*Oggetto*, tecnicamente, significa istanza di una classe; questa terminologia verrà introdotta nel Capitolo 7 relativo alle classi.

- Il paradigma funzionale, dove l'unità principale di programmazione è la funzione vista in puro senso matematico. Infatti, il flusso esecutivo del codice è guidato da una serie di valutazioni di funzioni che, trasformando i dati che elaborano, conducono alla soluzione di un problema. Gli aspetti rilevanti di questo paradigma sono: nessuna mutabilità di stato (le funzioni sono side-effect free, ossia non modificano alcuna variabile); il programmatore non si deve preoccupare dei dettagli implementativi del "come" risolvere un problema ma piuttosto di "cosa" si vuole ottenere dalla computazione. Esempi di linguaggi che supportano il paradigma funzionale sono: Lisp, Haskell, F#, Erlang e Clojure.
- Il *paradigma logico*, dove l'unità principale di programmazione è il predicato logico. In pratica con questo paradigma il programmatore dichiara solo i "fatti" e le "proprietà" che descrivono il problema da risolvere lasciando al sistema il compito di "inferirne" la soluzione e dunque raggiungerne il "goal" (l'obiettivo). Esempi di linguaggi che supportano il paradigma logico sono Datalog, Mercury, Prolog e ROOP.

Il linguaggio Java supporta, principalmente, il paradigma di programmazione ad oggetti, dove l'unità principale di astrazione è rappresentata dalla classe e dove vi è piena conformità con i principi fondamentali di tale paradigma: incapsulamento, ereditarietà e polimorfismo.

**NOTA**

Java è considerato un linguaggio di programmazione multiparadigma poiché, di fatto, supporta anche quello procedurale e, con la sua ultima versione, ha iniziato a supportare, seppure limitatamente all'introduzione delle lambda expression, anche quello funzionale.

Vediamo in breve che cosa significano questi principi, che saranno poi approfonditi nei capitoli di pertinenza.

- L'*incapsulamento* è un meccanismo attraverso il quale i dati e il codice di un oggetto sono protetti da accessi arbitrari (*information hiding*). Per dati e codice intendiamo tutti i membri di una classe, ovvero sia i dati membro (come le variabili), sia le funzioni membro (definite anche, in molti linguaggi di programmazione orientati agli oggetti, semplicemente come *metodi*). La protezione dell'accesso viene effettuata applicando ai membri della classe degli specificatori di accesso, definibili come *pubblico*, con cui si consente l'accesso a un membro di una classe da parte di altri metodi di altre classi; *protetto*, con cui si consente l'accesso a un membro di una classe solo da parte di metodi appartenenti alle sue classi derivate; *privato*, con cui un membro di una classe non è accessibile né da metodi di altre classi né da quelli delle sue classi derivate, ma soltanto dai metodi della sua stessa classe.
- L'*ereditarietà* è un meccanismo attraverso il quale una classe può avere relazioni di ereditarietà nei confronti di altre classi. Per relazione di ereditarietà intendiamo una relazione gerarchica di parentela *padre-figlio*, dove una classe figlio (definita *classe derivata* o *sottoclasse*) deriva da una classe padre (definita *classe base* o *superclasse*) i metodi e le proprietà pubbliche e protette, e dove essa stessa ne definisce di proprie. Con l'ereditarietà si può costruire, di fatto, un modello orientato agli oggetti che in principio è generico e minimale (ha solo classi base) e poi, man mano che se ne presenta l'esigenza, può essere esteso attraverso la creazione di sottomodelli sempre più specializzati (ha anche classi derivate).
- Il *polimorfismo* è un meccanismo attraverso il quale si può scrivere codice in modo generico ed estendibile grazie al potente concetto che una classe base può riferirsi a tutte le sue classi derivate cambiando, di fatto, la sua *forma*. Ciò si traduce, in pratica, nella possibilità di assegnare a una variabile `A` (istanza di una classe base) il riferimento di una variabile `B` (istanza di una classe derivata da `A`) e, successivamente, riassegnare alla stessa variabile `A` il riferimento di una variabile `C` (istanza di un'altra classe derivata da `A`). La caratteristica appena indicata ci consentirà, attraverso il riferimento `A`, di invocare i metodi di `A` che `B` o `C` hanno ridefinito in modo specialistico, con la garanzia che il sistema run-time di Java saprà sempre a quale esatta classe derivata appartengono.

**TERMINOLOGIA**

La discriminazione automatica, effettuata dal sistema run-time di Java, di quale oggetto (istanza di una classe derivata) è contenuto in una variabile (istanza di una classe base) è effettuata con un meccanismo definito *dynamic binding* (binding dinamico).

## Elementi di un ambiente Java

Per poter programmare in Java è necessario scaricare il JDK (*Java Development Kit*) e installarlo sulla propria piattaforma (si rimanda all'Appendice A per spiegazioni più dettagliate). Dopo l'installazione del JDK avremo a disposizione tutti gli strumenti per compilare ed eseguire i programmi creati, incluse svariate librerie di classi dette API (*Application Programming Interface*).

Nella Figura 1.1 è riportato un esempio di struttura di directory e file creata da un'installazione tipica (sono elencate, per motivi di spazio, tutte le directory e solo un file).

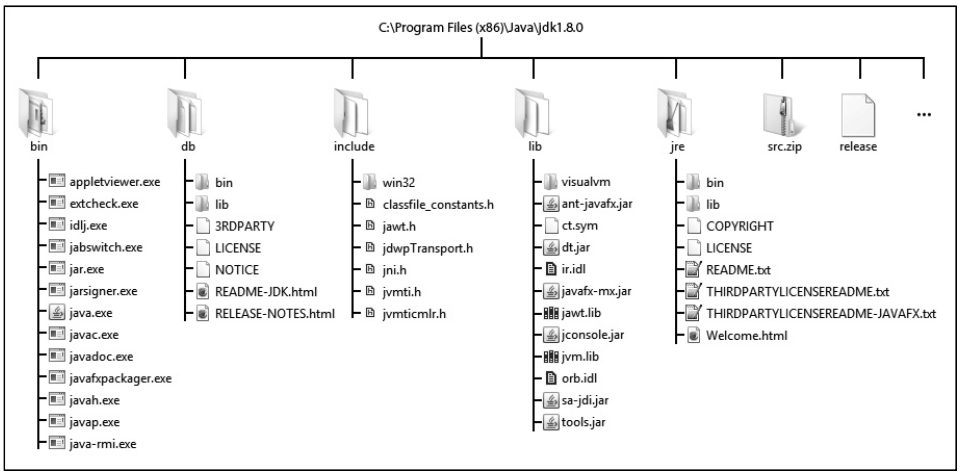


**Figura 1.1** Struttura ad albero del JDK 1.8 a 32 bit creata in un sistema Windows.

La Figura 1.1 evidenzia:

- una cartella di nome `bin` che contiene tutti i file eseguibili dei tool di sviluppo di Java come `javac.exe` (il compilatore del codice sorgente Java), `java.exe` (l'interprete ed esecutore di un programma Java), `jar.exe` (il gestore per file di archivi basati sul formato Java ARchive o JAR), `javap.exe` (il disassemblatore di file `.class`) e così via;
- una cartella di nome `db` che contiene Java DB, un sistema per la gestione di basi di dati relazionali realizzato da Oracle e basato sul software open source Apache Derby che è anch'esso un sistema per la gestione di database ma realizzato in seno all'Apache Software Foundation. In dettaglio troviamo le directory: `bin`, al cui interno sono presenti file di script per la configurazione dell'ambiente di utilizzo del server (modalità *embedded* o *client/server*), per l'avvio e per l'arresto del server e così via; `lib`, che contiene file di archivio `.jar` come `derby.jar` (l'engine library), `derbyclient.jar` (la network client library, necessaria per utilizzare il network client driver), `derbynet.jar` (la network server library, necessaria per avviare il network server) e così via;
- una cartella di nome `include` che contiene file header in linguaggio C per la programmazione in codice nativo mediante l'utilizzo della *Java Native Interface* (JNI) e della *Java Virtual Machine Tools Interface* (JVMTI);
- una cartella di nome `lib` che contiene librerie di classi e altri file utilizzati dagli eseguibili di sviluppo (per esempio `tools.jar` e `dt.jar`);
- una cartella di nome `jre` che contiene un'implementazione di un ambiente di runtime di Java (JRE, *Java Runtime Environment*) utilizzato dal JDK. In breve un JRE include una virtual machine (JVM, *Java Virtual Machine*), librerie di classi e altri file necessari per l'esecuzione dei programmi scritti con il linguaggio Java;

- un file di archivio basato sul formato ZIP di nome `src.zip` che contiene il codice sorgente di tutte quelli classi che rappresentano il core delle API di Java (per esempio quelle che si trovano nei package `java.*`, `javax.*` e così via);
- un file di nome `release` che contiene informazioni di servizio varie codificate come coppie di chiave/valore (per esempio, `JAVA_VERSION="1.8.0"`, `OS_NAME="Windows"`, `OS_VERSION="5.1"` e così via).

**TERMINOLOGIA**

Con il termine *codice nativo* si intende codice che è compilato per una specifica piattaforma/sistema operativo, come il codice C. Il codice Java, invece, nel momento in cui viene compilato produce un codice in linguaggio "intermedio" (detto *bytecode*) che è interpretabile da una qualsiasi macchina virtuale Java e quindi risulta indipendente dalla specifica piattaforma.

# Il primo programma Java

Vediamo, attraverso la disamina del Listato 1.1, quali sono gli elementi basilari per scrivere un programma in Java.

**Listato 1.1** PrimoProgramma.

```
package com.pellegrinoprincipe;

/*
 Primo programma in Java
 */
public class PrimoProgramma
{
    private static int counter = 0;

    public static void main(String[] args)
    {
        String testo = "Primo programma in Java:",
                testo2 = " Buon divertimento!";

        int a = 10, b = 20;

        // stampa qualcosa...
        System.out.println(testo + testo2);

        String s = "Stamperò un test condizionale: " + "tra a=" + a + " e b=" + b;

        System.out.println(s);

        if (a < b)
        {
            System.out.println("a<b");
        }
        else
        {
            System.out.println("a>b");
```

```
        }

        s = "Stamperò un ciclo iterativo, " + "dove leggerò per 10 volte il valore di a";

        System.out.println(s);

        for (int i = 0; i < 10; i++)
        {
            System.out.print("Passo " + i);
            System.out.println("--> " + "a=" + a);
        }
    }
}
```

Il Listato 1.1 inizia con l'istruzione `package`, che consente di creare librerie di tipi correlati. Infatti la classe denominata `PrimoProgramma`, definita con la keyword `class`, è un nuovo tipo di dato che apparterrà al package (o libreria) denominato `com.pellegrinoprincipe`.
Successivamente abbiamo un'istruzione di commento multiriga che consente di scrivere, tra i caratteri di inizio commento `/*` e fine commento `*/`, qualsiasi cosa che possa servire a chiarire il codice. L'istruzione di commento a singola riga prevede l'utilizzo dei caratteri `//`.
Le note esplicative scritte come commento possono contenere qualsiasi carattere, poiché saranno ignorate dal compilatore.
Dopo il commento multiriga abbiamo, come già detto, la definizione della classe denominata `PrimoProgramma` con la scrittura tra le parentesi graffe aperta `{` e chiusa `}` dei suoi membri (dati e metodi). La nostra classe ha un dato membro denominato `counter` e un metodo denominato `main`.
Il dato membro `counter` è una variabile, ovvero una locazione di memoria che può contenere valori che possono cambiare nel tempo. A ogni variabile deve essere associato un insieme di valori che può contenere, tramite la specificazione di un tipo di dato di appartenenza. La variabile `counter` ha, infatti, associato un tipo di dato intero.
Il metodo `main` rappresenta una sorta di metodo di avvio (o di *entry point*) per la nostra applicazione e deve essere sempre presente almeno in una classe, poiché è invocato automaticamente dall'interprete `java` all'atto dell'esecuzione dell'applicazione.
`main` è preceduto dalle keyword descritte di seguito.

- `void` indica che il metodo non restituisce alcun valore. Ovviamente un metodo può restituire un valore e in questo caso occorre indicarne il tipo di appartenenza, per esempio `int` per la restituzione di un tipo intero.
- `static` indica che si tratta di un metodo di classe che può essere invocato senza la creazione del relativo oggetto.
- `public` indica che il metodo è accessibile da client esterni alla classe dove il metodo stesso è stato definito.

Le keyword `static` e `public` permettono al metodo `main` di essere invocato pubblicamente dall'interprete `java`, in quanto client esterno, e di avviarne il relativo programma.
Seguono il `main` una serie di parentesi tonde all'interno delle quali è posta una variabile denominata `args` che rappresenta il parametro del metodo. Ogni metodo, infatti, può avere zero o più parametri che rappresentano, se presenti, delle variabili che saranno riempite con valori (detti *argomenti*) passati al metodo medesimo all'atto della sua invocazione. I

parametri di un metodo devono avere, inoltre, un tipo di dato associato; infatti, il parametro `args` è dichiarato come di tipo array di stringhe (`String[]`).
Il parametro `args` permette al metodo `main` di ottenere in input gli argomenti eventualmente passati quando si invoca dalla riga di comando il programma che lo contiene.
All'interno del metodo `main` sono scritte delle istruzioni che nel loro complesso rappresentano le operazioni che il metodo deve eseguire.
Troviamo infatti:

- la dichiarazione di variabili locali come `String testo`, `int a` e così via;
- l'invocazione di un metodo di stampa di dati su console (`println`);
- l'esecuzione di un'istruzione di selezione doppia `if/else` che valuta se una data espressione è vera o falsa eseguendone, a seconda del risultato della valutazione, il codice corrispondente (quello del ramo valutato vero oppure quello del ramo valutato falso);
- l'esecuzione di un'istruzione di iterazione `for` che consente di eseguire ciclicamente una serie di istruzioni finché una data espressione è vera.

Concludiamo con alcune indicazioni.

- Ogni istruzione deve terminare con il carattere ; (punto e virgola).
- Le parentesi graffe aperta `{` e chiusa `}` delimitano un blocco di codice contenente delle istruzioni.
- Il codice può essere scritto secondo il proprio personale stile di indentazione utilizzando i caratteri di spaziatura (spazio, tabulazione, Invio e così via) desiderati.

## Compilazione ed esecuzione del codice

Dopo aver scritto, con un qualunque editor di testo, il programma del Listato 1.1, vediamo come eseguirne la compilazione che, lo ricordiamo, è un processo mediante il quale il compilatore `javac` di Java legge un file sorgente (nel nostro caso `PrimoProgramma.java`) per trasformarlo in un file (`PrimoProgramma.class`) che conterrà istruzioni scritte in un linguaggio intermedio detto *bytecode*.

**Shell 1.1** Invocazione del comando di compilazione (Windows).

```
javac -d c:\MY_JAVA_CLASSES PrimoProgramma.java
```

**Shell 1.2** Invocazione del comando di compilazione (GNU/Linux).

```
javac -d /opt/MY_JAVA_CLASSES PrimoProgramma.java
```

Dopo la fase di compilazione segue la fase di esecuzione, nella quale un file `.class` (nel nostro caso `PrimoProgramma.class`) viene letto dall'interprete `java` per convertire il bytecode in esso contenuto in codice nativo del sistema dove eseguire il programma stesso.

**Shell 1.3** Invocazione dell'interprete java che esegue il programma.

```
java com.pellegrinoprincipe.PrimoProgramma
```

**Output 1.1** Esecuzione di Shell 1.3.

```
Primo programma in Java: Buon divertimento!
Stamperò un test condizionale: tra a=10 e b=20
a<b
Stamperò un ciclo iterativo, dove leggerò per 10 volte il valore di a
Passo 0--> a=10
Passo 1--> a=10
Passo 2--> a=10
Passo 3--> a=10
Passo 4--> a=10
Passo 5--> a=10
Passo 6--> a=10
Passo 7--> a=10
Passo 8--> a=10
Passo 9--> a=10
```

Dalla Shell 1.3 vediamo che il comando `java` esegue il programma `PrimoProgramma` che stampa quanto mostrato nell'Output 1.1.
È utile sottolineare alcuni aspetti.

- Il nome del programma `PrimoProgramma` è il nome della classe contenuta nel file omonimo.
- Il nome del file `PrimoProgramma.class` contenente il programma da eseguire viene passato al comando `java` senza l'indicazione dell'estensione `.class`.
- La classe `PrimoProgramma` invocata è preceduta dal nome del package di appartenenza `com.pellegrinoprincipe`, poiché quando una classe appartiene a un package, il suo nome deve sempre far parte di quel package.

## Problemi di compilazione ed esecuzione?

Elenchiamo alcuni problemi che si potrebbero incontrare durante la fase di compilazione o di esecuzione del programma appena esaminato.

- I comandi `javac` o `java` sono inesistenti? Verificare che il path del sistema operativo contenga la directory `bin` del JDK tra i percorsi di risoluzione. Si rimanda all'Appendice A per i dettagli su come impostare correttamente il path.
- Il compilatore `javac` non trova il file `PrimoProgramma.java`? Verificare che la directory corrente sia `c:\MY_JAVA_SOURCES` (per Windows) oppure `/opt/MY_JAVA_SOURCES` (per GNU/Linux).
- L'interprete `java` non trova il file `PrimoProgramma`? Verificare che la directory corrente sia `c:\MY_JAVA_CLASSES` (per Windows) oppure `/opt/MY_JAVA_CLASSES` (per GNU/Linux).