Parte prima: le basi della programmazione Java

1. Java: una breve descrizione

Java è un linguaggio di programmazione creato dalla Sun Microsystems e concepito da James Gosling, Patrick Naughton, Chris Warth, Ed Frank e Mike Sheridan nel 1991. Questo linguaggio veniva inizialmente chiamato "Oak", ma il nome è stato poi cambiato in "Java" nel 1995. Ha raggiunto un notevole successo grazie all'importanza assunta negli ultimi anni dalle reti di computer e dall'internetworking.
Java offre la possibilità di scrivere piccole applicazioni per internet, gli **applet**, eseguibili direttamente dai web browser. Il paradigma di programmazione del linguaggio Java è orientato agli oggetti, un paradigma di programmazione che si è dimostrato il più adeguato per lo sviluppo di sistemi software complessi.
Le peculiarità di Java sono, come poi vedremo più da vicino, **l'ereditarietà**, **il polimorfismo** e **l'incapsulamento**. Java è inoltre un linguaggio *general purpose*, cioè adatto a sviluppare applicazioni di varia natura, e non solamente applicazioni per il Web.
Java si relaziona al linguaggio C e al C++. Eredita la propria sintassi dal linguaggio C, ma il suo modello ad oggetti è adattato da C++. Una delle filosofie principali di C e C++ è che comanda il programmatore. Anche Java ha questa filosofia. Java dà al programmatore il pieno controllo, tranne alcune restrizioni nel campo applicativo dell'ambiente Internet. Java, inoltre, possiede delle differenze significative da C++. Anche se è stato influenzato da C++, Java non è una versione avanzata di C++ e, sebbene sia più semplice da comprendere di C++, è anche vero che, a differenza di quanto accade per quest'ultimo, la creazione di una applicazione Java, per quanto semplice possa essere, richiede la conoscenza di alcuni concetti che potrebbero non essere di immediata comprensione per chi si avvicina per la prima volta ad un linguaggio di programmazione orientato agli oggetti. Anche se la sintassi del linguaggio Java risulta complessivamente semplice, lo studio di questo linguaggio richiede tempi medio-lunghi. Questo è dovuto al fatto che Java mette a disposizione del programmatore un enorme collezione di classi, con metodi predefiniti, già pronte all'uso, raggruppate in pacchetti detti *packages*. A differenza di altri linguaggi di programmazione, in cui si può anche programmare ad oggetti, in Java bisogna assolutamente programmare ad oggetti. Nelle parti successive vedremo più da vicino come funziona questo potente linguaggio e conosceremo il paradigma di programmazione ad oggetti. Vedremo, in questa parte, le nozioni basilari per imparare a programmare in Java e brevemente i concetti di incapsulamento, polimorfismo ed ereditarietà, tratti in comune a tutti i linguaggi orientati agli oggetti.

2. Java: un linguaggio interpiattafroma

Che cosa vuol dire: "Java è un linguaggio *interpiattaforma*"? Un programma scritto in Java può essere eseguito su qualsiasi tipo di computer, senza dover modificare il codice sorgente. Java, essendo interpiattaforma, si distingue dai linguaggi *multipiattaforma*. Il compilatore Java (javac.exe) traduce un programma scritto in Java non in linguaggio macchina, ma in **bytecode**, ovvero *una serie di istruzioni simili alle istruzioni in linguaggio macchina, ma non specifiche per un particolare processore*. Per essere eseguito, il bytecode deve essere interpretato da uno specifico programma, chiamato **interprete di bytecode** (java.exe), che provvederà anche ad eseguirlo.
L'interprete di bytecode è spesso denominato **JVM – Java Virtual Machine** (macchina virtuale Java) o **sistema runtime** di Java. Una JVM è implementata anche nei Web browser più importanti;

questa JVM serve, per l'appunto, ad eseguire gli applet. Questa macchina virtuale Java è denominata **compilatore Java JIT** (Just In Time).
Quindi, Java è un linguaggio compilato o interpretato? Per quanto abbiamo appena visto, possiamo affermare che Java è un linguaggio interpretato, anche se la produzione del bytecode avviene tramite un processo di compilazione.

3. Richiami sugli oggetti

Il paradigma di programmazione orientata agli oggetti (OOP) ha come concetto chiave quello di **oggetto**. In un linguaggio di programmazione OO si determinano le caratteristiche generiche che ogni singolo oggetto deve possedere per poter far parte di una famiglia di oggetti, detta **classe**.
Consideriamo, ad esempio, la classe "Automobile". Possiamo affermare che questo oggetto descrive le caratteristiche comuni che i singoli oggetti "Automobile" devono possedere per far parte di questa famiglia, e cioè "fare carburante", "muoversi", "cambiare marcia", "fermarsi" ecc.
La classe "Automobile", quindi, fa da **modello astratto** delle automobili concrete, dal momento che ne descrive le caratteristiche comuni.
Gli esemplari di una classe sono definiti **istanze** o **oggetti**. Pertanto, un'automobile concreta è un'istanza, o oggetto, della classe "Automobile".
Quindi:

- una **classe** è un modello generico per una famiglia di oggetti con caratteristiche simili.
- un'**istanza** (o **oggetto**) è un esemplare particolare di una determinata classe

Ad esempio:

CLASSE = Automobile
ISTANZA = OGGETTO = Punto bianca (un esemplare particolare della classe Automobile).

Un'applicazione Java può essere composta da una o più classi.

Una classe si compone generalmente di due parti: **attributi** (o **proprietà**) e **metodi.**

Le **proprietà** specificano le **caratteristiche** della classe, determinandone l'aspetto, lo stato e altre qualità. In Java, per fare riferimento alle proprietà di un oggetto si userà la seguente sintassi:

```
<Nome oggetto>.<Proprietà>
```

I **metodi** specificano le **funzionalità** che una classe offre, cioè le operazioni che un oggetto è in grado di compiere e che corrispondono ai comportamenti dell'oggetto. La sintassi Java per i metodi è:

```
<Nome oggetto>.<metodo>(<[Parametri]>)
```

4. Incapsulamento, polimorfismo ed ereditarietà

Accenneremo adesso tre tratti essenziali del linguaggio Java: incapsulamento, polimorfismo ed ereditarietà. Vedremo più da vicino questi argomenti nel corso di questa trattazione.

Incapsulamento
L'incapsulamento è un meccanismo di programmazione che unisce il codice e i dati che gestisce mantenendo entrambi al sicuro da interferenze ed utilizzi non appropriati. In un linguaggio di programmazione orientato agli oggetti, il codice ed i dati possono essere uniti in modo da formare una sorta di *scatola*, al cui interno vi sono tutti i dati ed il codice. Quando dati e codice vengono concatenati in questo modo viene creato un oggetto. Come vedremo in seguito, un oggetto è composto da un insieme di metodi e attributi incapsulati nell'oggetto. L'unità di base dell'incapsulamento in Java è la classe.

Polimorfismo
La parola 'polimorfismo' deriva dal greco e significa "molte forme". Il polimorfismo è la qualità che permette ad un'interfaccia di accedere ad una classe generale di azioni. In altre parole un'interfaccia è polimorfica nel senso che cambia forma a seconda della classe in cui si trova. Ad esempio, il manubrio di uno scooter (che rappresenta l'interfaccia) è lo stesso, qualunque sia il tipo di meccanismo usato. Il manubrio funziona nello stesso modo, sia che si tratti di una normale bicicletta, di una mountain bike o di un tandem. Generalmente, il concetto di polimorfismo viene espresso dalla frase: "un'interfaccia, più metodi". In definitiva, il polimorfismo aiuta a ridurre la complessità, permettendo alla medesima interfaccia di essere utilizzata per specificare una classe generale di azione.

Ereditarietà
L'ereditarietà è il processo grazie al quale un oggetto può acquisire le proprietà di un altro oggetto. Ad esempio, una comune bicicletta fa parte della classificazione "bicicletta", la quale si trova in una classe più ampia, la classe "veicoli a due ruote" che, a sua volta, fa parte di una classe ancora più ampia, la classe "veicolo". Quindi, la nostra comune bicicletta iniziale, *eredita* tutte le qualità delle classi che, nella gerarchia, occupano il livello più alto e definisce delle qualità specifiche che la rendono unica all'interno della sua classe. Quindi, il meccanismo di ereditarietà rende possibile per un oggetto essere un'istanza specifica di un contenitore più generale.

Non preoccupatevi se non riuscite a comprendere appieno questi aspetti. Come già detto, avremo modo di considerare meglio questi tratti del linguaggio Java nel corso della trattazione, ed ogni dubbio, al momento giusto, verrà chiarito.

5. Struttura base di un'applicazione Java

La più semplice applicazione Java avrà la seguente struttura di base:

```
class <Nome>
{
      public static void main(String[] args)
      {
            <Istruzioni>
      }
```

```
}
```
Dal metodo **main()** comincerà l'esecuzione dell'applicazione. Creiamo ora una piccola applicazione che scriva a video "Ciao Mondo".

```
class Saluta
{
      public static void main(String[] args)
      {
            // commento al codice
            System.out.println("Ciao Mondo");
      }
}
```

Analizziamo il codice:

- Le parentesi graffe { } sono usate per individuare un blocco, che può essere una classe, un metodo o un insieme di istruzioni.
- Ogni istruzione deve terminare con un punto e virgola (;)
- E' stato usato un commento. I commenti sono parti del programma che vengono ignorate dal compilatore, e quindi non modificano il programma finale, ma sono utili al programmatore per inserire chiarimenti o appunti. Un commento si inserisce con la doppia barra // (commento su una riga) e con barre e asterischi /* */ (commento su più righe).

Notiamo in particolare:

- **`System`** è il nome della classe che contiene oggetti e metodi per accedere alle risorse di sistema e per descriverne i comportamenti specifici.
- **`out`** è un oggetto della classe.
- **`println()`** è il metodo che ha come argomento il valore da visualizzare. Tale argomento è di tipo *string*. Esiste anche il metodo **print()** che, a differenza del metodo **println()**, stampa un messaggio a video e resta sulla stessa riga, mentre il secondo stampa un messaggio a video e va a capo alla riga successiva.

6. main(): gli argomenti

Nell'esempio visto sopra, main() è stato preceduto da alcune parole chiave, quali *public*, *static* e *void*. Vediamo sommariamente il loro significato, anche se verrà ripreso in maniera approfondita nelle parti successive:

- **public** indica che il metodo è pubblico, cioè può essere 'visto' da ogni classe.
- **static** indica che il metodo è associato alla classe e può essere richiamato facendolo precedere dal nome della classe.
- **void** indica che il metodo non restituisce alcun valore di ritorno.

Notiamo ancora che main() ha come parametro un vettore di stringhe chiamato **args**, che contiene i parametri con cui viene invocato il programma. Più precisamente, args[0] conterrà il primo parametro, args[1] il secondo ecc. Vedremo più da vicino i vettori nella terza parte.

7. Librerie e package

Per concludere questa parte, diremo anche che Java include un'insieme di librerie, costituite da un insieme di classi già compilate e di varia utilità, che possono essere utilizzate all'interno dei programmi. Queste librerie vengono chiamate **package**.
I package hanno una **struttura gerarchica**, ovvero ogni package può contenere, al suo interno, altre classi. Il package **java** è il package che contiene tutti gli altri. Al suo interno ve ne sono altri, come ad esempio **lang, io, awt** ecc. I package si richiamano per mezzo della *dot notation*. Ad esempio, per richiamare il package lang scriveremo 'java.lang'. Alcuni esempi di package sono:

- **java.lang** contiene le classi di base e viene automaticamente incluso in tutte le applicazioni.
- **java.io** contiene tutte le classi per la gestione dei file e dei flussi I/O.
- **java.awt** contiene le classi per la gestione della grafica
- **java.net** contiene le classi per la gestione di applicazioni che scambiano dati con la rete
- **java.applet** contiene le classi per la creazione di applet.

Per inserire un package all'interno di un'applicazione, si userà la sintassi

```
import <Package>;
```

Ad esempio, per inserire il package java.io:

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

Con l'istruzione vista sopra includeremo tutte le classi del package java.io. Se avessimo voluto inserire solamente la classe File:

```
import java.io.File;
```

Si conclude così la prima parte. Nella seconda parte vedremo le espressioni, gli operandi e gli operatori.