Introduzione

Imparare un linguaggio di programmazione, e nello specifico Java, è come fare un viaggio in una terra lontana e sconosciuta. Dunque è, sì, un viaggio meraviglioso, affascinante e gratificante, ricco di sorprese e scoperte, ma anche un viaggio non semplice, che richiede tanta pazienza e il giusto tempo. In ogni caso, al termine del viaggio, la ricompensa sarà elevata: si sarà infatti appreso un linguaggio di programmazione, potente e flessibile, che permetterà di utilizzare API di qualsiasi tipo, da quelle più *tediose* per interfacciarsi a un database a quelle più *divertenti* per scrivere videogiochi, e dunque scrivere software di qualsiasi tipo.
Java, come avremo modo di verificare durante tutto il percorso di apprendimento che il libro intende offrire, è un linguaggio di programmazione estremamente espressivo e ricco di costrutti sintattici, e che dà grande libertà operativa al programmatore, il cui unico limite sarà solo la sua fantasia e la preparazione sulle regole sintattiche dei costrutti o sulla semantica delle operazioni.
Desidero spendere qualche parola anche sui principi ispiratori che mi hanno guidato nella scrittura del presente testo.

- Gli argomenti propri di ogni capitolo hanno un preciso e chiaro ordine. Ogni capitolo esprimerà compiutamente il relativo obiettivo didattico e non si sovrapporrà o comprenderà contenuti di altri capitoli (per esempio, se nel Capitolo 2 si parlerà delle variabili, solo nel Capitolo 3 si parlerà degli array). Quanto detto può apparire abbastanza ovvio ma non lo è. Infatti, oggi, si sta assistendo alla proliferazione di testi nei quali gli argomenti sono scritti "a spirale" dove cioè un argomento può contenere riferimenti iniziali ad altri argomenti, i quali saranno poi trattati approfonditamente solo nel capitolo di pertinenza. Questo, a mio parere, laddove non strettamente necessario e comunque non legato ai costrutti del linguaggio (è evidente che per mostrare il valore di una variabile bisogna dire qualcosa sul metodo `System.out.printf`) induce a distrazioni e fa perdere inutile tempo per la corretta comprensione della corrente unità didattica.
- Il modo espositivo che ho seguito è rigoroso, laddove necessario piuttosto formale, e ho prestato molta attenzione al corretto uso della terminologia propria di Java. Questo non è un libro del tipo "*impariamo Java in 24 ore*" oppure "*Java for Dummies*". Java è un linguaggio complesso e ricco di costrutti; per insegnarlo ci vuole "serietà" e giusto rigore; per impararlo ci vuole pazienza e disciplina.

- Ho concepito i listati e gli snippet di codice in modo che dessero una chiara indicazione pratica dei relativi argomenti teorici; sono piuttosto brevi, autoconclusivi e decorati da commenti che danno, talvolta, ulteriori spiegazioni teoriche.
- Non ho trattato API specifiche della piattaforma Java ma solo quelle "connesse" con il linguaggio. Questo è un libro su Java, ossia sul linguaggio di programmazione, non un libro sulla programmazione in Windows, GNU/Linux o macOS. Ha senso spendere centinaia di pagine per parlare della programmazione di GUI, di database, del Web e così via, elencando di fatto una pletora di API? Ha senso sottrarre centinaia di pagine ai dettagli sui costrutti del linguaggio stesso per "donarle" a quelle API? Credo di no. Credo che un libro su Java *sia* un libro su Java, ossia un libro, per esempio, che spende centinaia di pagine per spiegare in modo compiuto, dettagliato e rigoroso che cos'è la OOP, la programmazione funzionale e così via per tutti gli altri argomenti a esso pertinenti. In definitiva, credo che un testo così strutturato dia al lettore, paziente e volenteroso, una marcia in più per poi affrontare con la giusta comprensione e consapevolezza le API che avrà interesse di apprendere e che, nel caso di Java, godono di un'abbondante documentazione.

Organizzazione del libro

Il libro è organizzato nelle seguenti parti.

- Parte I – *Concetti e costrutti fondamentali*, costituita dai seguenti capitoli: Capitolo 1, *Introduzione al linguaggio*; Capitolo 2, *Variabili, costanti, letterali e tipi*; Capitolo 3, *Array*; Capitolo 4, *Operatori*; Capitolo 5, *Istruzioni e strutture di controllo*; Capitolo 6, *Metodi*. Questa parte enuclea le nozioni essenziali e fondamentali che sono propedeutiche di un qualsiasi corso sulla programmazione: dal concetto di variabile e array, passando per gli operatori e le strutture di controllo del flusso di esecuzione del codice, finendo con un dettaglio su come scrivere blocchi di codice attraverso il costrutto di *metodo*.

  NOTA

  Nell'organizzazione dei capitoli ho preferito introdurre i metodi prima del costrutto di classe, poiché sono "strutture" sintattiche più semplici e in modo da affrontare i concetti essenziali del linguaggio in ordine crescente di complessità. Filosoficamente, questa scelta si sposa bene con l'inquadramento del paradigma a oggetti (basato sulle classi) inteso come estensione del paradigma procedurale (basato sulle procedure). Non a caso, le strutture di controllo della programmazione ad oggetti, che vengono poi utilizzate all'interno dei metodi, sono le stesse del paradigma procedurale.

- Parte II – *Paradigmi, stili di programmazione e gestione degli errori*, costituita dai seguenti capitoli: Capitolo 7, *Programmazione basata sugli oggetti*; Capitolo 8, *Programmazione orientata agli oggetti*; Capitolo 9, *Programmazione generica*; Capitolo 10, *Programmazione funzionale*; Capitolo 11, *Eccezioni e asserzioni*. Questa parte illustra come avvantaggiarsi, nella costruzione di sistemi software, dei più noti paradigmi di programmazione: da quello maggiormente utilizzato, proprio della OOP, a quello, definito funzionale, che oggi sta riscuotendo una profonda rivalutazione e un certo successo. Analizza anche come impiegare nei programmi uno stile di programmazione *generico* ossia che fa uso

di tipi e funzionalità che sono in grado di compiere una medesima operazione su più tipi di dato differenti. Spiega, infine, come intercettare e gestire adeguatamente gli errori software grazie all'utilizzo del meccanismo di gestione delle eccezioni.

- Parte III – *Concetti e costrutti supplementari e avanzati*, costituita dai seguenti capitoli: Capitolo 12, *Package*; Capitolo 13, *Moduli*; Capitolo 14, *Annotazioni*; Capitolo 15, *Documentazione del codice sorgente*. Questa parte dettaglia come organizzare, tramite i *package*, in modo raggruppato, strutturato e gerarchico, dei tipi che sono connessi a una particolare funzionalità applicativa; come decomporre un'applicazione in un insieme di *moduli* e utilizzare, dunque, nel suo insieme, l'importante *sistema a moduli* introdotto a partire dalla versione 9 del linguaggio; come associare, tramite *metadati* (annotazioni), informazioni descrittive agli elementi di un programma che decorano; come utilizzare tag "speciali" che consentono di formattare il codice sorgente in modo che sia possibile generare per esso un'adeguata documentazione.
- Parte IV – *Introduzione ai tipi e alle librerie essenziali*, costituita dai seguenti capitoli: Capitolo 16, *Caratteri e stringhe*; Capitolo 17, *Espressioni regolari*; Capitolo 18, *Collezioni*; Capitolo 19, *Programmazione concorrente*; Capitolo 20, *Input/Output: stream e file*; Capitolo 21, *Programmazione di rete*. Questa parte analizza i caratteri e le stringhe, le espressioni regolari, le collezioni di dati, la programmazione concorrente, le operazioni sui file e la programmazione di rete.
- Parte V - *Appendici*, costituita da: Appendice A, *Installazione e configurazione della piattaforma Java SE 11*; Appendice B, *Installazione e utilizzo di NetBeans*; Appendice C, *Sistemi numerici: cenni*. Questa parte mostra come installare la piattaforma Java e usare l'IDE NetBeans, oltre a fornire un'introduzione dei sistemi numerici decimale, ottale, esadecimale e binario.

Struttura del libro e convenzioni

Gli argomenti del libro sono organizzati in capitoli. Ogni capitolo è numerato in ordine progressivo e denominato significativamente in base al suo obiettivo didattico (per esempio, Capitolo 2, *Variabili, costanti, letterali e tipi*). I capitoli sono poi suddivisi in paragrafi di pertinenza, al cui interno possiamo avere dei blocchi di *testo* o di *grafica*, a supporto alla teoria, denominati in accordo con il seguente schema:

Tipologia NrCapitolo.NrProgressivo Descrizione.

Tipologia può esprimere: un listato di codice sorgente (*Listato*), un frammento di codice sorgente (*Snippet*), la sintassi di un costrutto Java (*Sintassi*), dei comandi di shell (*Shell*), l'output di un programma (*Output*), una figura (*Figura*), una tabella (*Tabella*).
Per esempio, il blocco *Listato 6.2 ByValue.java (ByValue)* indica il secondo listato di codice del Capitolo 6, avente come descrizione il nome del file `.java` di codice sorgente e, tra parentesi, il nome del progetto dell'IDE.
Quanto ai listati, abbiamo adottato anche la seguente convenzione: i puntini di sospensione (...) eventualmente presenti indicano che in quel punto sono state omesse alcune parti di codice, presenti però nei file `.java` allegati al libro. Gli stessi caratteri possono talvolta trovarsi anche negli output di un programma eccessivamente lungo.

NOTA

I comandi di shell devono essere scritti *così come sono*, ossia senza inserire i caratteri di fine riga che invece sono presenti nel libro per garantire un'adeguata formattazione del testo.

Codice sorgente e progetti

L'archivio `.zip` contenente il codice del libro ha la seguente struttura di cartelle:

`[Windows | Linux | MacOS]` → `CapNr` → `[Listati | Snippet | Sorgenti]`.

Avremo, cioè, una cartella per ciascun sistema operativo, denominata `Windows` o `Linux` o `MacOS`; ciascuna di esse conterrà la cartella dei capitoli (`Cap01`, `Cap02` e così via). Queste ultime cartelle conterranno, a loro volta, le cartelle dei progetti dei listati (`Listati`) o degli snippet di codice (`Snippet`) per l'IDE NetBeans; la cartella dei sorgenti (`Sorgenti`) conterrà, invece, i sorgenti `.java` e i file annessi per chi non desiderasse usare l'IDE menzionato.

Compilare ed eseguire direttamente i listati e gli snippet di codice

Per rendere agevole la compilazione e l'esecuzione dei listati e degli snippet di codice, indipendentemente dall'IDE NetBeans, ecco i seguenti consigli:

- per GNU/Linux o macOS creare le directory `MY_JAVA_SOURCES`, `MY_JAVA_CLASSES`, `MY_JAVA_PACKAGES`, `MY_JAVA_JARS`, `MY_JAVA_MODS`, `MY_JAVA_RUNTIMES` e `MY_JAVA_DOCUMENTATION`, in `C:` per Windows e in `$HOME`;
- per Windows, prima di eseguire i programmi Java digitare, nel command prompt, il comando `chcp 65001` che cambierà il code page di default in UTF-8 e consentirà l'output corretto delle stringhe di testo (per esempio, quelle che conterranno le lettere accentate).

Compilare ed eseguire con gli IDE i listati e gli snippet di codice

Se lo si desidera, è comunque possibile utilizzare l'IDE NetBeans per editare, compilare, eseguire e svolgere il debugging del codice sorgente presente nel libro.
A tal fine consultate l'Appendice B, *Installazione e utilizzo di NetBeans*, per una spiegazione in merito all'utilizzo introduttivo di tale IDE e alla creazione e all'impiego dei progetti.

Il nuovo sistema di rilasci della piattaforma Java

Il 6 settembre 2017 un avvenimento importante ha scosso tutta la comunità Java ed è collegato a un articolo pubblicato sul blog di Mark Reinhold, *Chief Architect of the Java Platform Group* in Oracle, nel quale viene proposto, in breve, un cambiamento epocale nella temporizzazione dei rilasci della piattaforma Java e del JDK.

Secondo Reinhold, infatti, l'ecosistema Java si è evoluto negli anni in maniera piuttosto irregolare e non ha mai avuto uno *schedule* programmato in modo temporalmente omogeneo. Si è data sempre maggiore priorità alle *big feature* integrabili nel linguaggio a scapito, magari, di piccoli anche se significativi cambiamenti. Ciò ha comportato che, finché queste feature non fossero completate e adeguatamente testate, la release finale della piattaforma slittasse di conseguenza.
Questi ritardi di rilascio, accettabili diversi anni fa quando piattaforme concorrenti si aggiornavano con la stessa lentezza, oggi non lo sono più perché le stesse piattaforme evolvono con una certa rapidità. Ecco dunque la necessità che anche l'ecosistema Java inizi a evolversi molto più rapidamente, al fine di garantirne un'adeguata competitività e cercando, comunque, di mitigare anche la normale tensione che esiste tra gli sviluppatori, ansiosi di vedere sempre più innovazioni e in tempi rapidi, e il mondo *enterprise* che invece desidera più stabilità e sicurezza.
Nelle parole di Reinhold:

> Una cadenza di rilasci biennale, in retrospettiva, è semplicemente troppo lenta. Per raggiungere una cadenza regolare dobbiamo pubblicare versioni feature a ritmo più rapido. Posporre una feature da una versione a un'altra dovrebbe essere una decisione tattica dalle conseguenze minime più che una decisione strategica con impatto significativo. Per cui pubblichiamo una versione feature ogni sei mesi.

Ciò detto, vediamo dunque di esplicitare il predetto cambiamento sulle release di Java proposto da Reinhold e che sarà, tranne modifiche dell'ultima ora, operativo a partire dal 20 marzo 2018.
I rilasci di Java cambieranno da un modello definito di tipo *feature-driven* (ogni rilascio era vincolato al completamento di importanti feature che poteva causare anche diversi anni di ritardo) a un modello definito di tipo *time-driven* che garantirà quanto segue.

- Una *feature release* ogni sei mesi che conterrà qualsiasi cosa: da API nuove o migliorate a feature proprie del linguaggio o della JVM. Ogni release dovrà essere disponibile a marzo e a settembre di ogni anno, a partire da marzo 2018 (JDK 10).
- Un *update* ogni tre mesi. Saranno limitati all'eliminazione di eventuali bug o alla risoluzione di problemi di sicurezza delle nuove feature. Ogni update dovrà essere disponibile a gennaio, aprile, luglio e ottobre di ogni anno.
- Una *long-term support release* (LTS) ogni tre anni, a partire da settembre 2018 (JDK 11). Questa release avrà la garanzia di ricevere update almeno per tutti e tre gli anni della sua esistenza.

Nella pratica questo nuovo modello di rilascio *time-based* potrà soddisfare sia gli sviluppatori, più inclini a volere in tempi rapidi innovazioni e nuove feature per il linguaggio (adotteranno le feature release con update semestrali ma dovranno sottostare ad aggiornamenti comunque semestrali: Java 10, Java 11 eccetera), sia il mondo enterprise più incline, invece, a usare una release Java meno innovativa ma stabile (adotteranno una long-term support release con update garantiti per tutto il triennio e l'aggiorneranno solo dopo almeno tre anni).
Chiudiamo infine con una nota pratica su questo cambiamento dei rilasci di Java, che è esplicitato bene nel JEP 322: *Time-Based Release Versioning* e ha sostituito la possibile stringa di versione proposta sempre da Reihnold, la quale avrebbe dovuto avere il for-

mato `$YEAR.$MONTH` (per esempio, per il 2018, la release di marzo sarebbe stata designata 18.3, quella di settembre 18.9 e così via).
In accordo, dunque, con il JEP 322, la futura stringa di versione per Java sarà costituita dai seguenti elementi:

- `$FEATURE`: incrementata ogni sei mesi. A marzo 2018 varrà 10 (JDK 10), a settembre 2018 varrà 11 (JDK 11) e così via.
- `$INTERIM`: varrà sempre 0 perché in questo modello di rilascio ogni sei mesi non si avrà mai una versione intermedia. Verrà comunque lasciata per motivi di flessibilità e possibilità di utilizzo.
- `$UPDATE`: verrà incrementata un mese dopo l'incremento di `$FEATURE` e poi ogni tre mesi. Ad aprile 2018 avremo per esempio il valore 1 (JDK 10.0.1), a luglio 2018 il valore 2 (JDK 10.0.2).

A partire da marzo 2018 avremo quindi un nuovo *release schedule* per la piattaforma Java: piuttosto che una big release con una moltitudine di cambiamenti ogni tre o quattro anni, avremo tante piccole release ogni sei mesi. Dunque, dopo la release di JDK 9 (settembre 2017), avremo un JDK 10 a marzo 2018, un JDK 11 a settembre 2018 e così via.