Guida Pratica allo Sviluppo di Applicazioni Java

Simone Lungarella (simone.lungarella-cic@ibm.com)

2025-08-19

Indice

1 Introduzione alla Guida e all'Ecosistema Java per il Web

Lo sviluppo di applicazioni web in Java è un campo vasto e in continua evoluzione, che si fonda su un'architettura robusta e su un ecosistema di strumenti maturo. La scelta di Java per lo sviluppo di software aziendale, applicazioni web, piattaforme finanziarie e sistemi di big data è un indicatore della sua rilevanza duratura.

Dal 1996, anno del suo lancio ufficiale da parte di Sun Microsystems, Java si è affermato come un linguaggio universale e multipiattaforma, diventando il più utilizzato a livello globale. Il successo iniziale si deve in gran parte al principio "Write once, run anywhere" (WORA), che ha permesso agli sviluppatori di compilare il codice in un bytecode indipendente dalla piattaforma. Questo bytecode viene quindi eseguito su qualsiasi sistema dotato di una Java Virtual Machine (JVM), sia esso Windows, Linux o macOS. Questa indipendenza ha rappresentato un vantaggio competitivo che continua a definire la sua architettura.

Inizialmente, lo sviluppo web in Java si basava su tecnologie come le Servlet e le JavaServer Pages (JSP), che rappresentavano i mattoni fondamentali per la creazione di contenuti dinamici. Nel corso degli anni, l'ecosistema si è evoluto per affrontare la complessità crescente delle applicazioni aziendali, passando da architetture monolitiche a modelli orientati ai microservizi.

Frameworks reattivi e leggeri come Spring Boot e Quarkus sono nati per supportare l'architettura dei microservizi, offrendo tempi di avvio rapidi e un ingombro di risorse minimo, ideali per gli ambienti cloud-native. Questo percorso di apprendimento è progettato per guidare uno sviluppatore attraverso gli strumenti e i concetti essenziali di questo ecosistema moderno, fornendo un fondamento solido e una via pratica per la creazione della prima applicazione web.

La presente guida è stata concepita per fornire un percorso didattico chiaro e progressivo. Inizialmente, si affronteranno i prerequisiti e gli strumenti fondamentali, dalla scelta di un Java Development Kit (JDK) alla configurazione di un ambiente di sviluppo. Successivamente, si esploreranno in dettaglio due pilastri dello sviluppo moderno: Git per il controllo di versione e Apache Maven per la gestione del progetto e del build.

2 Preparazione dell'Ambiente di Sviluppo

Prima di iniziare a scrivere codice, è fondamentale configurare un ambiente di sviluppo adeguato. Ciò include l'installazione del JDK, la scelta di un IDE o di un editor di testo, e la preparazione degli strumenti di base per la gestione del codice e del progetto.

2.1 Scelta e Installazione del Java Development Kit (JDK)

Il primo e più importante passo è l'installazione di un JDK. Apache Maven, ad esempio, richiede un JDK installato per poter funzionare correttamente. Il JDK è l'insieme di strumenti necessari per compilare il codice Java in bytecode e include una Java Runtime Environment (JRE) e altri componenti essenziali. La maggior parte delle distribuzioni JDK moderne si basa su OpenJDK, l'implementazione di riferimento open-source della specifica Java SE. Sebbene Oracle fornisca i binari del proprio JDK, esistono numerose distribuzioni di terze parti, ciascuna con le proprie politiche di licenza e supporto. Tra le più diffuse figurano Adoptium Eclipse Temurin, Azul Zulu, Amazon Corretto e BellSoft Liberica JDK.

Per un progetto a lungo termine, la raccomandazione è optare per una versione con supporto a lungo termine (LTS), come Java 11, 17 o 21, che ricevono aggiornamenti di sicurezza per un periodo prolungato, garantendo stabilità e riducendo la necessità di frequenti aggiornamenti di versione.

2.1.1 Risorse utili

- Archivio open-jdk;
- Guida setup JDK — www.geeksforgeeks.org;

2.2 Guida alla Scelta e Configurazione di un IDE o un Editor di Testo

La scelta dell'editor influenzerà la tua produttività per i prossimi mesi di corso e stage. Non è solo una questione di preferenze personali: l'editor giusto accelera l'apprendimento, quello sbagliato lo rallenta.

La scelta solitamente ricade su uno dei seguenti editor:

- Eclipse: È un IDE open-source con una vasta quota di mercato e un ecosistema di plugin molto esteso. Le sue principali virtù sono la flessibilità e le opzioni di personalizzazione. Tuttavia, la sua interfaccia può risultare complessa e travolgente per i principianti, e la gestione di numerosi plugin può a volte portare a conflitti o problemi di prestazioni.
- Visual Studio Code (VS Code): Sviluppato da Microsoft, è un editor gratuito e leggero. È un'ottima opzione per i principianti o per progetti su scala ridotta. Sebbene non raggiunga l'intelligenza avanzata di Eclipse out-of-the-box, il suo supporto per Java può essere notevolmente potenziato tramite l'installazione del "Java Extension Pack" e altre estensioni dal suo vasto marketplace.

Di seguito è riportata una matrice decisionale per semplificare la scelta ma esistono molteplici editor validi non riportati per semplicità:

Strumento	Punti di forza	Punti di debolezza	Livello di astrazione	Target Utente Consigliato
Eclipse	Completo per Java, ampio ecosistema di plugin	Configurazione iniziale complessa, interfaccia pesante, consumo di risorse	**Molto alto**	Aziende, ambienti enterprise consolidati
VS Code	Avvio rapido rispetto agli IDE tradizionali, leggero, grande ecosistema di estensioni	Necessita plugin per supporto completo a linguaggi complessi, rischio frammentazione	**Medio-alto** (ibrido)	Principianti, progetti leggeri, sviluppatori con risorse limitate
Neovim	Estremamente leggero, avvio istantaneo, personalizzazione totale, vicino al sistema	Richiede configurazione avanzata, curva di apprendimento ripida, poche feature pronte	**Basso** (vicino al codice)	Sviluppatori esperti, power-user che privilegiano controllo e performance

> N.B.: si consiglia l'utilizzo di VS Code perché richiede relativamente poche risorse ed è configurabile in 5 minuti.

2.2.1 Approfondimenti

- Setup Eclipse — dev.java;
- Setup VS Code — code.visualstudio.com;

2.3 Introduzione e Installazione di Git

Git è un sistema di controllo di versione (VCS) distribuito, essenziale per la gestione della cronologia del codice e la collaborazione. L'installazione di Git è un prerequisito fondamentale per qualsiasi sviluppatore moderno. Le istruzioni per l'installazione variano a seconda del sistema operativo, ma in genere consistono nel download di un installer per Windows o macOS, o nell'utilizzo di un gestore di pacchetti su Linux.

Per approfondire l'installazione, si rimanda alla documentazione ufficiale di Git: https://git-scm.com.

3 Fondamenti del Linguaggio Java

Prima di utilizzare framework moderni come Spring Boot, è fondamentale avere una comprensione minima della sintassi e delle convenzioni del linguaggio Java. Questo capitolo fornisce un'introduzione pratica e veloce ai concetti essenziali.

3.1 La Struttura di un Programma Java

Un programma Java è organizzato in **classi**. Ogni applicazione ha un punto di ingresso definito dal metodo `main`.

Esempio di programma minimale:

```java
public class HelloWorld {

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

- `public class HelloWorld`: definisce una classe chiamata *HelloWorld*;
- `public static void main(String[] args)`: metodo di ingresso dell'applicazione, eseguito all'avvio;
- `System.out.println(...)`: stampa un messaggio sulla console;

Per compilare il codice in bytecode è possibile utilizzare `javac HelloWorld.java`. Per eseguire il metodo main di una classe, invece, può essere utilizzato `java HelloWorld`.

3.2 Tipi di Dato e Variabili

Java è un linguaggio **fortemente tipizzato**: ogni variabile ha un tipo preciso.

```java
int number = 42;
String name = "Mario";
boolean isActive = true;
```

- Tipi primitivi più comuni: `int`, `double`, `boolean`, `char`.
- Tipi complessi (oggetti): `String`, wrapper quali `Integer` o `Double`, classi personalizzate.

3.3 Strutture di Controllo

Come in altri linguaggi, Java offre costrutti condizionali e cicli:

```java
if (numero > 0) {
    System.out.println("Positivo");
} else {
    System.out.println("Negativo o zero");
}

for (int i = 0; i < 5; i++) {
```

```
        System.out.println("Iterazione: " + i);
    }
```

3.4 Classi e Oggetti

Java è un linguaggio orientato agli oggetti. Le classi rappresentano modelli, mentre gli oggetti sono istanze di queste classi.

```
public class Person {
    private String firstName;
    private String lastName;

    /**
     * Constructor method of the Person class.
     *
     * @param firstName First name of the person instantiated.
     * @param lastName  Last name of the person instantiated.
     * @return The instantiated object.
     */
    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    /**
     * Method executable on instances of objects of type Person.
     *
     * @return A String defining greetings from the Person.
     */
    public String greet() {
        return "Hi, my name is " + nome + " " + cognome;
    }

    public static void main(String[] args) {
        Person p = new Person("Mario", "Rossi");
        System.out.println(p.greet());
    }
}
```

Concetti chiave:

- **Attributi** (`firstName` , `lastName`): descrivono lo stato.
- **Metodi** (`greet()`): definiscono il comportamento.
- **Costruttori**: inizializzano nuovi oggetti.

3.5 Package e Import

Il codice Java è organizzato in **package**. Ad esempio, se la classe `Person` si trova nel pacchetto `it.ibm.demo` , all'inizio del file avremo: `package it.ibm.demo;` .

Per usare classi di altri pacchetti, si usa `import java.util.List;` .

3.6 Approfondimento Java

Per approfondire i concetti introdotti, consultare il videocorso Udemy o risorse alternative quali www.baeldung.com:

- Le basi di Java in forma scritta (www.baeldung.com/get-started-with-java-series);
- Videocorso Udemy;

4 Fondamenti di Git: Controllo di Versione per lo Sviluppo Moderno

Il vero valore di Git non risiede solo nella sua funzione di "salvataggio" del codice, ma nella sua capacità di gestire l'intera evoluzione di un progetto e di facilitare la collaborazione in modo efficiente.

4.1 Concetti Chiave di Git

Per comprendere Git, è necessario familiarizzare con tre concetti fondamentali che definiscono il ciclo di vita di un file all'interno di un progetto:

- Working Directory (Directory di lavoro): è la cartella locale del progetto dove si trovano i file e dove si apportano le modifiche.
- Staging Area (Area di Staging): è un'area intermedia dove si preparano le modifiche prima di registrarle nel repository. Un file viene spostato in quest'area con il comando `git add`.
- Local Repository (Repository locale): è la copia del database di Git sul proprio computer. Le modifiche preparate nell'area di staging vengono salvate in questo repository con il comando `git commit`.
- Remote Repository (Repository remoto): è la versione del software memorizzata online ed accessibile a tutti quelli che ne hanno accesso.

È cruciale distinguere tra un repository locale, che risiede sulla macchina dello sviluppatore, e un repository remoto, che è una copia centrale del progetto ospitata su un servizio esterno come GitHub o GitLab. I comandi `git push` e `git pull` sono i meccanismi che sincronizzano i repository locali e remoti, permettendo al team di condividere il lavoro.

4.2 Comandi Essenziali e Flussi di Lavoro Semplici

Ecco una panoramica dei comandi Git più comuni, essenziali per un principiante.

Comando	Funzione/Descrizione	Esempio d'uso
`git init`	Inizializza un nuovo repository Git in una cartella locale	`git init`
`git clone`	Crea una copia di un repository remoto esistente	`git clone <url_repository>`
`git status`	Mostra lo stato dei file nella working directory e nell'area di staging	`git status`
`git add`	Sposta le modifiche dalla working directory all'area di staging	`git add file.java` o `git add .`
`git commit`	Salva le modifiche dall'area di staging al repository locale	`git commit -m "Messaggio"`
`git push`	Invia le modifiche del repository locale a un repository remoto	`git push origin main`
`git pull`	Scarica e integra le modifiche da un repository remoto	`git pull origin main`

La padronanza di questi comandi elementari permette di gestire progetti individuali, ma il vero valore di Git si manifesta nei flussi di lavoro collaborativi. Un flusso di lavoro comune per i principianti, o per team di piccole dimensioni, è il flusso di lavoro centralizzato, che si avvicina al modello di sistemi più datati come SVN. In questo modello, gli sviluppatori clonano un unico repository centrale, apportano modifiche e le inviano (`push`).

Tuttavia, possono sorgere conflitti se un altro sviluppatore ha inviato delle modifiche nel frattempo. In questi casi, Git si rifiuta di eseguire la push per evitare di sovrascrivere il lavoro altrui. La soluzione consiste nel recuperare le modifiche altrui (`git pull`) e risolvere i conflitti prima di procedere.

A un livello più avanzato, esistono flussi di lavoro più complessi, come il Gitflow o il GitHub Flow, che impongono regole più specifiche sull'interazione tra i branch e offrono maggiore sicurezza e robustezza per progetti di grandi dimensioni. L'esistenza di questi flussi di lavoro dimostra che Git non è solo un insieme di comandi, ma uno strumento strategico per la gestione del ciclo di vita del codice. La possibilità di annullare facilmente gli errori, ad esempio con `git rebase --abort`, è un'ulteriore conferma della filosofia di sicurezza e controllo che sta alla base del sistema.

4.3 Approfondimento Git

Di seguito alcune risorse utili:

- Documentazione completa ufficiale (git-scm.com);
- Videocorso Udemy;
- Esempio setup repository — Atlassian

5 Gestione del Progetto e delle Dipendenze con Apache Maven

In un progetto Java, la gestione del build e delle dipendenze può diventare complessa. Apache Maven è uno strumento che si propone di risolvere questi problemi, fungendo non solo da "build tool" ma da vero e proprio "Project Management Tool".

5.1 Cos'è Maven e il suo Ruolo Cruciale

Apache Maven semplifica e standardizza il processo di sviluppo e deployment. I suoi principali vantaggi includono:

- Gestione delle dipendenze: Permette agli sviluppatori di elencare le dipendenze esterne in un semplice file XML. Maven si occupa di risolvere, scaricare e includere le librerie necessarie durante il processo di build, eliminando manualmente la dependency hell;
- Struttura del progetto standardizzata: Impone una "convenzione sulla configurazione", il che significa che i progetti Maven seguono una struttura di directory predefinita (src/main/java, src/test/java, ecc.). Questo approccio facilita la collaborazione e l'ambientamento di nuovi membri in un progetto;
- Automazione dei task: Fornisce una serie di comandi standardizzati (`mvn clean` , `mvn test` , `mvn package`) per eseguire le varie operazioni del ciclo di vita del build.

5.2 Il Cuore di Maven: il file pom.xml

L'unità di lavoro fondamentale di Maven è il Project Object Model (POM), un file XML chiamato pom.xml. Questo file contiene tutte le informazioni e i dettagli di configurazione del progetto. Al suo interno, tre elementi sono obbligatori e formano le "coordinate" che identificano univocamente il progetto nel vasto ecosistema di Maven: groupId, artifactId e version.

La gestione delle dipendenze è uno degli aspetti più potenti del POM. La sezione `<dependencies>` elenca tutte le librerie di cui un progetto ha bisogno. Maven gestisce anche le dipendenze transitive, ovvero le librerie di cui le dipendenze del progetto stesso hanno bisogno. In alcuni casi, è possibile escludere manualmente dipendenze indesiderate per evitare conflitti. Il POM supporta anche il concetto di ereditarietà, dove un POM "figlio" può ereditare configurazioni e dipendenze da un POM "genitore", un'architettura utile per progetti multi-modulo.

5.3 Il Ciclo di Vita del Build e le Fasi di Maven

Maven definisce un ciclo di vita del build come una sequenza di fasi (phases). L'esecuzione di una fase attiva automaticamente tutte le fasi precedenti, garantendo un ordine logico e coerente.

Fase	Obiettivo	Esempio di Utilizzo
`validate`	Convalida la correttezza del progetto e verifica che tutte le informazioni necessarie siano disponibili	`mvn validate`
`compile`	Compila il codice sorgente del progetto	`mvn compile`
`test`	Esegue i test unitari sul codice compilato	`mvn test`
`package`	Impacchetta il codice compilato in un formato distribuibile (es. JAR, WAR)	`mvn package`
`install`	Installa il pacchetto finale nel repository Maven locale (.m2), rendendolo disponibile per altri progetti sulla stessa macchina	`mvn install`

Sebbene *Git* e *Maven* siano strumenti distinti, la loro integrazione è un elemento cruciale in un flusso di lavoro di sviluppo moderno. Questa integrazione non è una magia, ma è resa possibile da una ricca architettura di plugin. Ad esempio, è possibile utilizzare un plugin Maven come il git-commit-id-maven-plugin per integrare automaticamente informazioni sul commit Git (come l'ID del commit) all'interno dell'applicazione al momento del build.

Queste informazioni sono preziose per il debugging e la tracciabilità delle versioni in ambienti di produzione. Inoltre, Maven può essere configurato per pubblicare pacchetti su repository remoti ospitati da servizi come GitHub Packages, creando un ciclo di sviluppo e distribuzione continuo che unisce la gestione del codice e la gestione del build in un unico ecosistema.

5.4 Approfondimento Maven

Di seguito alcune risorse utili:

- Maven in 5 minuti — documentazione ufficiale;
- Videocorso Udemy;