ALTRI PARADIGMI DI PROGRAMMAZIONE

A.A. 2020/2021

Laurea triennale in Informatica

1: Introduzione

CHI SONO IO

Michele Mauro

Passionate Developer

@michelemauro

https://dev.to/michelemauro

REPERIBILITÀ

michele.mauro@unipd.it

subject: [app2020]

PARADIGMA

Dal greco *paradigma*: esempio, esemplare.

Modello di riferimento, termine di paragone.

In filosofia della scienza: "visione globale del mondo da parte degli scienziati di una certa disciplina"
(Kuhn, 1962)

PARADIGMA DI PROGRAMMAZIONE

Insieme delle tecniche e dei metodi considerati adeguati ad affrontare una classe determinata, anche se ampia, di problemi.

ESEMPIO: MODELLO DI VON NEUMANN

Architettura composta da CPU, Memoria centrale, organi di I/O

Singola linea di esecuzione, sincrona, su risorse completamente sotto controllo.

ALTRI PARADIGMI

CONCORRENZA

Più linee di esecuzione contemporanee, asincrone, che condividono l'uso di un insieme di risorse in modo (possibilmente) coordinato.

PARALLELISMO

Più linee di esecuzione contemporanee, asincrone, che eseguono in modo coordinato lo stesso calcolo su di una partizione dei dati di ingresso.

IN RETE

Collaborazione con altri sistemi attraverso la comunicazione asincrona su di una interfaccia di rete.

DISTRIBUZIONE

Un sistema è costituito da nodi indipendenti che, attraverso una rete non affidabile, devono coordinare l'esecuzione dello stesso lavoro, condividendo il consenso sullo stato complessivo del sistema.

REATTIVITÀ

Un sistema distribuito costruito sulle basi della comunicazione asincrona tramite messaggi da cui ottiene caratteristiche di flessibilità, resilienza, scalabilità, reattività.

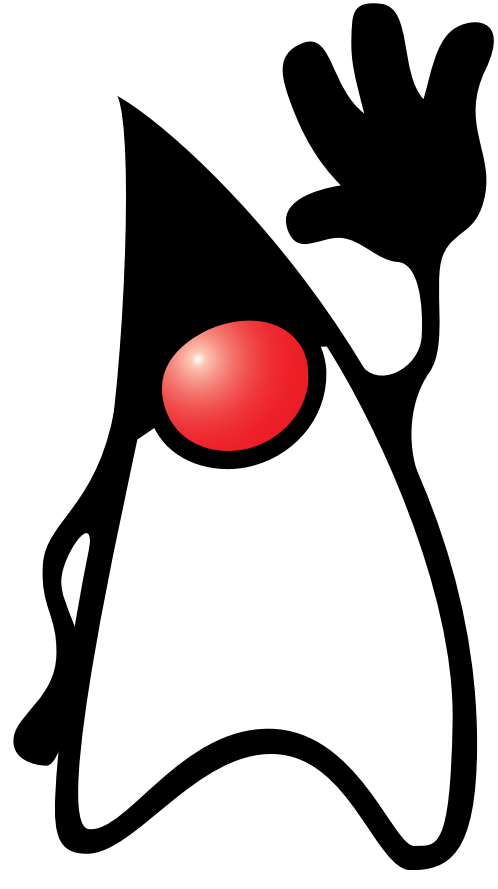
Definito dal Reactive Manifesto.

LINGUAGGIO

Per studiare questi paradigmi di programmazione useremo il linguaggio Java.

JAVA

Java è un linguaggio
Object-Oriented,
a memoria gestita,
con controllo statico dei
tipi, basato su Classi ad
ereditarietà singola,
con una sintassi simile a C
e C++.



Viene compilato in un linguaggio intermedio, detto *bytecode*, interpretato da una piattaforma, la Java Virtual Machine a sua volta implementata per più sistemi operativi.

L'obiettivo della piattaforma è WORA:
Write **O**nce **R**un **A**nywhere.

La portabilità della piattaforma garantisce la portabilità del codice.

JAVA DESIGN GOALS

- Simple, Object Oriented, and Familiar
- Robust and Secure
- Architecture Neutral and Portable
- High Performance
- Interpreted, Threaded, Dynamic

https://www.oracle.com/java/technologies/introduction-to-Java.html

Java ben si presta come supporto per lavorare nei paradigmi di nostro interesse perché:

- la programmazione concorrente e parallela è uno dei suoi scopi di design
- l'impiego in sistemi distribuiti è stato un caso d'uso fin dall'inizio
- nella sua evoluzione la JVM è diventata una piattaforma di ricerca molto apprezzata

LA PIATTAFORMA

La prima peculiarità di Java è in realtà la sua piattaforma di esecuzione, la JVM.

Ad oggi è una specifica aperta e standardizzata, disponibile per tutti i sistemi operativi.

Nella seconda metà della decade 2000 la JVM diventa uno degli ambienti in cui è più semplice sperimentare e fare ricerca sui linguaggi di programmazione.

Alcune di queste ricerche diventano poi feature della piattaforma (per es. l'istruzione *invokedynamic*) o del linguaggio Java (Generics, Lambda).

Molti altri linguaggi di successo oltre a Java hanno come ambiente di esecuzione la JVM.

Alcuni dei più rilevanti sono:
Scala, Groovy, Clojure
JRuby, Jython

Inizialmente la JVM interpretava il *bytecode* direttamente, con forti penalità di performance.

Nei primi anni 2000 entra in produzione la tecnica detta Compilazione Just-In-Time (*JIT*) che progressivamente porta Java ad avere prestazioni oggi comparabili con linguaggi di più basso livello.

Oggi la JVM implementa alcune delle tecniche più avanzate di compilazione e gestione del codice:

- Gestione del mix fra codice nativo e interpretato a seconda della statistica di esecuzione
- Compilazione in anticipo in eseguibile nativo

- Implementazioni specifiche per applicazioni particolari
- Esecuzione su silicio (in passato) e su GPU/FPGA e architetture eterogenee

Fino al 2017, fra una versione e l'altra di Java (e della JVM) passavano diversi anni.

Da Java 9, uscito il 21/9/2017, viene pubblicata una major release ogni 6 mesi.
L'ultima è Java 15, uscito il 15/9/2020.

Java 11 è l'ultima LTS; la prossima sarà Java 17, che uscirà a Settembre 2021.

INSTALLAZIONE

JDK

Principali distribuzioni

https://jdk.java.net/15/ https://adoptopenjdk.net/
https://www.azul.com/downloads/zulu-community/
https://aws.amazon.com/corretto/

Metodi di installazione

Windows: (`chocolatey`)

```
choco install adoptopenjdk15
```

MacOs: (`homebrew`)

```
$ brew tap AdoptOpenJDK/openjdk
$ brew cask install adoptopenjdk
```

Ubuntu/Debian (`apt-get`):

```
sudo apt-get install openjdk-15-jdk
```

Alpine (`apk`):

```
apk add openjdk14
```

ESECUZIONE

La JVM è predisposta per il linking dinamico del codice: i nomi di classi e metodi vengono controllati al momento del caricamento

Il compilatore Java produce un file con estensione `.class` per ogni classe ottenuta dal codice.

Secondo la convenzione seguita dalla JVM, le classi sono organizzate in package che corrispondono a directory nel filesystem.

Il codice può essere caricato in modo generico da molte fonti; in pratica, per questioni di sicurezza si usa solo il filesystem locale.

Un parametro molto importante per la JVM è il `CLASSPATH`, ovvero l'elenco dei posti dove cercare una classe.

Il caricamento del codice è demandato ad una gerarchia di `ClassLoader`. Questo consente:

- dividere chiaramente il caricamento delle classi di libreria da quelle delle classi "di estensione" (globali per una installazione) e da quello delle classi dell'applicazione

- separare la visibilità di classi particolari in modo che solo codice di un certo tipo o di una certa provenienza possa raggiungerlo.
- caricare il codice da origini differenti, per es. da una URL, da un archivio, da una classe sul filesystem.

Il fatto che una classe venga caricata solo quando viene richiesta ha profonde implicazioni e conseguenze.

- rende possibili alcune tecnologie che sono diventate peculiari dell'ecosistema Java
- può essere (è stato) un problema di sicurezza profondamente dibattuto
- può provocare comportamenti inaspettati nel codice (se lo si dimentica)

COMPILATORE

Il compilatore Java è il comando `javac`, ed è scritto anch'esso in Java.

Il suo compito è trasformare un file sorgente (`.java`) in un file bytecode (`.class`).

I file sorgenti *devono* essere organizzati in questo modo:

- Ogni file deve chiamarsi come l'oggetto che contiene: `NomeClasse.java`
- Il percorso delle directory deve corrispondere al package in cui l'oggetto si trova
- Nei sorgenti, o nel CLASSPATH devono esserci tutti i tipi nominati dai sorgenti

L'ordine di compilazione non è importante: il compilatore deduce ed organizza le dipendenze fra le classi.

Come in altre piattaforme, comunque, difficilmente il compilatore viene usato direttamente. Di solito viene pilotato da uno strumento di livello più elevato.

ESECUTORE

Il comando `java` avvia la JVM ed esegue il bytecode contenuto nel CLASSPATH.

Il bytecode può trovarsi in file `.class` o in altre forme.

Il codice della classe principale viene caricato e la JVM inizia ad eseguire a partire dal punto di ingresso.

La JVM può essere ispezionata durante l'esecuzione per fornire statistiche, informazioni diagnostiche, o interfacce di gestione.

ARCHIVIAZIONE

Il comando `jar` gestisce archivi di codice java che possono essere utilizzati all'interno del CLASSPATH.

Il formato `.jar` è il più comune formato di distribuzione del codice nella piattaforma Java.

Si tratta di un archivio compresso zip contenente alcuni file specifici che descrivono il suo contenuto e come usarlo.

Alcune varianti sono legate a specifiche tecnologie:

- `war` - Web Archive (applicazione web con codice e risorse statiche)
- `ear` - Enterprise Archive (codice organizzato secondo lo standard JakartaEE)

Un file `jar` può essere firmato digitalmente per garantirne l'integrità e l'autenticità.

Una parte importante del successo della piattaforma JVM è la facilità con cui una libreria può essere individuata, reperita e procurata come file `jar` a partire da un repository pubblico o privato.

ALTRI COMANDI

- `javadoc`: Java Documentation Compiler
- `javap`: Java disassembler
- `jdb`: Java debugger
- `jps`: Java Processes
- `jstat`: JVM statistics monitor
- `jaotc`: Java Ahead Of Time Compiler (Java 14)

ECOSISTEMA

Attorno alla JVM è cresciuta una intera industria ed un vero ecosistema in cui l'OpenSource è forza quasi predominante.

Lo sviluppo del linguaggio è finanziato da Oracle, che nel 2009 ha acquisito Sun Microsystem, ideatore originale di Java.

Tuttavia, il modo in cui il software OpenSource ha fornito soluzioni di alta qualità alla piattaforma è stato un fattore di successo cruciale.

Ad oggi, le principali entità che producono software OpenSource in Java sono:

- Apache Foundation
- Eclipse Foundation

Ci sono anche entità commerciali che vivono vendendo formazione e supporto per il software OpenSource che producono:

- Spring
- Red Hat (ora di IBM)

COSTRUIRE, MEGLIO

Il compilatore java, come abbiamo detto, non è molto pratico da usare direttamente.

Molto presto (fine 1990) il nascere dei primi progetti di una certa dimensione fa emergere la necessità di strumenti che gestiscano la fase di costruzione e preparazione del software.

Il primo di questi strumenti è **Apache Ant**: attraverso una specifica in XML vengono dettagliati i passi necessari alla costruzione del software.

Ant rende questo processo ripetibile, facile da comunicare, semplice da estendere.

Il grosso passo avanti avviene nel 2004, con il primo rilascio di **Apache Maven**.

L'approccio di Maven è **opinionated**: lo strumento ha un'idea molto precisa di come un progetto deve essere organizzato.

Adeguarsi a tale idea porta moltissimi vantaggi; deviarne è possibile, ma complesso e nella maggior parte dei casi inutile.

Per Maven il progetto è descritto da un `POM`

Project Object Model

Dichiarativamente, vengono elencati:

- nome e metadati del progetto
- dipendenze
- plugin e loro configurazioni

La grande maggioranza dei progetti può essere correttamente costruita semplicemente seguendo le convenzioni e configurando pochi plugin di base.

La ricca libreria di plugin forma un effetto rete per cui ogni nuova tecnologia ha come requisito, per la sua diffusione, essere correttamente integrata con Maven in modo da essere facilmente adottabile.

Il modello di gestione delle dipendenze proposto da Maven è probabilmente uno dei fattori di successo dell'ecosistema Java.

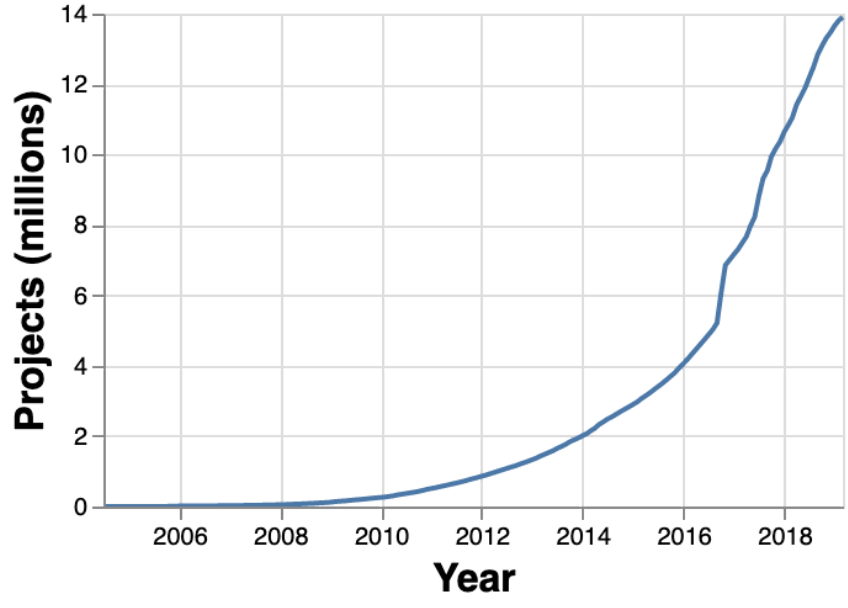
Ogni libreria o componente può essere aggiunto ad un progetto indicandolo secondo delle coordinate:

`gruppo:artefatto:versione`

Attraverso le coordinate, Maven è in grado di contattare un repository remoto (privato o pubblico), procurarsi il `jar` del componente, e gestire tutte le configurazioni degli strumenti della JVM per renderlo disponibile durante la programmazione.

Questo metodo di distribuzione rende semplicissimo collaborare o mettere a disposizione di altri (o del pubblico) una libreria di software.

Oggi con questo sistema si possono raggiungere oltre 18 milioni di artefatti open source; la pubblicazione del software in Java è diventato sinonimo di pubblicazione in un repository Maven.



E lo stesso procedimento è ormai standard (e sottointeso come indispensabile) per qualsiasi nuovo ambiente di programmazione.

Nel 2007 viene rilasciata la prima versione di **Gradle** una alternativa a Maven che punta a migliorare alcuni suoi limiti.

In particolare:

- non più XML, ma un linguaggio di programmazione (Groovy) per definire il progetto
- non più una struttura fissa, ma task che dipendono l'uno dall'altro a formare un DAG

Una parte della motivazione di Gradle è, sicuramente, il fatto che XML sia passato di moda e che invece i linguaggi con un sistema di tipi dinamico vadano invece per la maggiore.

Ma anche l'esperienza che non tutti i progetti possano incastrarsi nella rigida struttura del `POM` di Maven.

Gradle non ha soppiantato Maven: i due strumenti si spartiscono il mercato.

Maven è in molti progetti la soluzione "classica". Gradle ha come punti di forza l'essere il sistema di build ufficiale per le applicazioni Android e la capacità di innovare, per esempio, aggiungendo Kotlin come linguaggio di descrizione della build.

Il fatto che nel corso sarà adottato Gradle è, in realtà, ininfluente: la scrittura di una build non è argomento del corso.

IDE

Un Integrated Development Environment è un programma che mette a disposizione degli strumenti specifici per rendere più efficace l'attività dello sviluppo software:

- editor specializzati per linguaggio
- integrazione con strumenti di build
- integrazione con debugger ed esecutori di test

Nell'ecosistema Java i più popolari sono:

- Eclipse
- IntelliJ Idea
- Microsoft VSCode

Nota: è consigliato sceglierne uno, ma utile provare gli altri.

DEMO

LINK INTERESSANTI

I CONTENUTI DI JAVA 15

https://www.infoq.com/news/2020/09/java15-released/

I 25 ANNI DI JAVA

https://www.infoq.com/news/2020/05/java-at-25/

WOMEN IN TECH

Una conferenza gratuita organizzata da Manning Publications

Martedì 13/10 1800-2300 CEST

https://freecontent.manning.com/livemanning-conferences-women-in-tech/