Lezione 1 — 26 Settembre 2017

## Introduzione al linguaggio Java

Il linguaggio Java nasce all'inizio degli anni '90 sotto forma di un progetto che si chiamava OAK, presso la Sun Microsystem (gigante informatico del secolo scorso, caduta in declino negli anni 2000 e poi acquisita dalla Oracle nel 2009-2010)

La Sun Microsystem produceva hardware (server, workstation), aveva la sua versione di UNIX che si chiamava SOLARIS; all'interno di questa azienda Gosling iniziò il progetto OAK che aveva lo scopo di risolvere il problema di poter eseguire una applicazione su dispositivi di elettronica di consumo in modo indipendente dalle caratteristiche hardware e software della macchina.

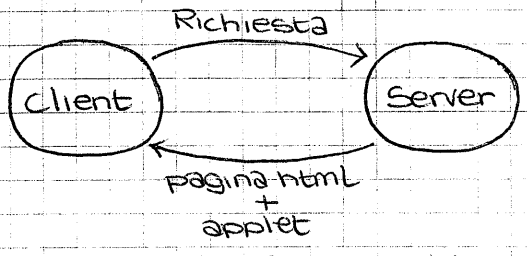
L'idea era di poter scrivere applicazioni per dispositivi tipo lettori dvd, apparecchi per la ricezione digitale terrestre, forni, frigoriferi, ecc..
Il problema era che ogni produttore di questi apparati sviluppava il proprio hardware e i propri software; di volta in volta, sopra ogni dispositivo, c'era una specie di mini sistema operativo diverso e questo complicava lo sviluppo di applicazioni che dovevano essere customizzate per il singolo prodotto.
Non era possibile scrivere una applicazione valida per tutti i dispositivi di uno stesso tipo, ma doveva essere customizzata per il singolo dispositivo, perché ogni dispositivo aveva un microprocessore e un sistema operativo differente.

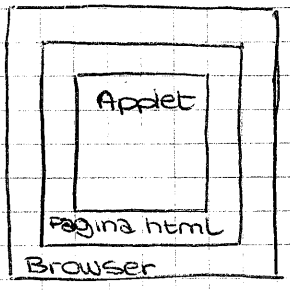
L'idea quindi era sviluppare una piattaforma software che rendesse le applicazioni portabili, cioè data una applicazione anche in formato binario, questa possa essere eseguita su macchine diverse in termini di architettura e di software (s.o)

Questo era il proposito della piattaforma fino agli anni '90, fino a quando poi la cosa si combinò con l'esplosione di Internet: allora il progetto cambiò direzione e l'idea fu quella di sviluppare una piattaforma software che consentisse di scrivere applicazioni portabili per il web.

In particolar modo divenne molto popolare per via del meccanismo delle applet (adesso praticamente scomparse)

Le applet funzionavano così:




Il client (web) fa una richiesta e il server (web) risponde con una pagina html che poteva incorporare delle applet; nella pagina html veniva riservata una porzione all'interno della quale veniva eseguita l'applet, tutto a sua volta eseguito all'interno del browser

Questo meccanismo consentiva di eseguire del codice all'interno del browser e potevano essere realizzate animazioni, interfacce grafiche, elaborare localmente input da parte dell'utente senza coinvolgere ogni volta il server.

Era fondamentale la portabilità del linguaggio, perché il browser poteva essere eseguito sia su macchine Windows che UNIX o processori sia INTEL che non; l'applet poteva essere eseguita su entrambi i client perché indipendente da S.O e hardware sottostante.

Il linguaggio include inoltre delle caratteristiche di robustezza e sicurezza che rendevano abbastanza sicuro eseguire del codice prelevato dal web all'interno della macchina client; in particolare le applet venivano eseguite all'interno di una server box (ambiente di esecuzione che vietava ad esempio di accedere al file system, di aprire connessioni se non con il server da cui erano state scaricate, ...)

Ora questo meccanismo è stato sostituito da tecnologie più recenti e Java viene maggiormente usato in ambito server dove c'è abbastanza differenza di architetture e S.O.; mentre le macchine desktop sono principalmente Windows, quelle server sono UNIX.

Avere applicazioni lato server che lavorino su macchine Windows o UNIX senza preoccuparmi dell'hardware sottostante è un vantaggio per lo sviluppatore e per le aziende.

Java viene utilizzato molto anche per la programmazione di dispositivi mobile, in particolare tutte le applicazioni android sono scritte in Java.

## Design Goals (obbiettivi di progettazione)

- Simple → rimozione aspetti complessi del C++ (overloading operatori, puntatori, ereditarietà multipla)
  Object Oriented → il codice esiste solo dentro le classi
  Familiar → molti costrutti simili o uguali al C/C++

- Robust → situazioni di errore vanno gestite con le eccezioni
  Secure → rimozione di aspetti complessi del C++ rende le applicazioni più sicure (non c'è rischio di accedere a zone di memoria per errore)
- Architecture neutral → non si fanno assunzioni su hardware e S.O. nella definizione del linguaggio
  Portable → portabilità delle applicazioni a livello binario

- High Performance → Java è un linguaggio interpretato e allo stesso tempo compilato, e ciò consente di aver prestazioni più elevate di linguaggi puramente interpretati.
- Interpreted →
  Threaded → il linguaggio prevede librerie per la programmazione concorrente
  Dinamic → non si ha la necessità di produrre un eseguibile che contiene tutte le classi dell'applicazione; si può iniziare ad eseguire il programma caricando una sola classe e se si incontrano definizioni di altre classi, vengono caricate dinamicamente (non esiste una fase di collegamento per creare un unico eseguibile; il collegamento è dinamico e avviene a tempo di esecuzione)

## Versioni

| | | | |
|---|---|---|---|
| 1996 | JDK 1.0 | 2004 | J2SE 5.0 |
| 1997 | JDK 1.1 | 2006 | Java SE 6 |
| 1998 | J2SE 1.2 | 2011 | Java SE 7 |
| 2000 | J2SE 1.3 | 2014 | Java SE 8 |
| 2002 | J2SE 1.4 | 2017 | Java SE 9 |

JDK = Java Development Kit
↳ ambiente di sviluppo: le versioni del linguaggio venivano definite dalla versione dell'ambiente

Con la versione 1.2 sono state introdotte differenze in termini di sintassi del linguaggio (nuove parole chiave) e la piattaforma Java è stata suddivisa in 3 versioni:
- Java Standard Edition (SE) → per macchine desktop
- Java Micro Edition → applicazioni per palmari e cellulari
- Java Enterprise Edition → sovrainsieme di JavaSE per macchine Server

Micro Edition è scomparsa dal mercato

Nelle ultimi 15 anni sono state aggiunte solo nuove librerie, la sintassi e gli aspetti fondamentali del linguaggio sono rimasti sostanzialmente uguali; è cambiato solo qualcosa nella gestione delle eccezioni

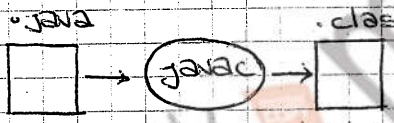
Sul sito della Oracle possiamo scaricare JDK, JRE e NetBeans.

JRE = Java Runtime Environment
↳ ambiente di esecuzione per applicazioni Java

NetBeans = IDE Java
↳ ambiente di sviluppo integrato che include JDK utile per lo sviluppo di interfacce Grafiche. include anche tutta la documentazione Java in formato html.

Il linguaggio Java è interpretato e compilato, in particolar modo le cose funzionano così:

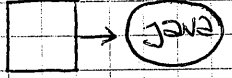
- ie programmatore scrive il codice sorgente con estensione .java
- ie codice sorgente viene compilato usando javac (compilatore java)
- se non ci sono errori sintattici ie compilatore produce un file con estensione .class che contiene il bytecode



file sorgente bytecode

Il bytecode è una sorta di linguaggio macchina per una macchina astratta, che è la Java Virtual Machine; in pratica l'ideatore del linguaggio ha definito le specifiche della JVM che poi viene normalmente realizzata in software (il così detto interprete)

- ie file .class può essere eseguito dandolo in ingresso alla JVM, attivandola con il comando java



Esecuzione

| Applicazione Java |
|---|
| JVM |
| S.O. |
| H.W. |

La nostra applicazione Java viene eseguita sopra lo strato software della JVM, che a sua volta si appoggia sopra il SO eseguito sopra l'hardware del calcolatore.

La JVM è normalmente realizzata via software ed è di fatto un interprete che esegue il bytecode ottenuto dalla compilazione del file sorgente java.

Per lo sviluppatore Java, conoscere come è fatto il bytecode non è normalmente importante; contiene delle istruzioni che di solito lavorano su uno stack

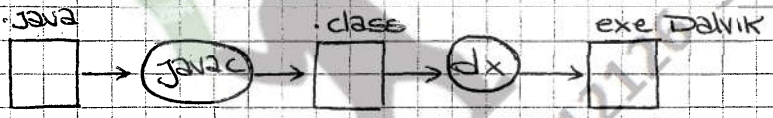
(la JVM è basata su stack) e quindi le istruzioni macchina riguardano inserimenti in stack e operazioni fra elementi dello stack.

Sapere i dettagli del bytecode è necessario solo a coloro che vogliono implementare una nuova JVM o un nuovo compilatore.

Quindi data una applicazione, la possiamo eseguire su una JVM che si appoggia su un S.O UNIX o Windows; è la JVM che colma le differenze hardware e software.

Una caratteristica particolare e poco sfruttata è che la JVM e il linguaggio sorgente sono due cose separate: la JVM è in grado di eseguire codice binario prodotto dalla compilazione di applicazioni scritte in altri linguaggi (alcuni linguaggi possono essere compilati producendo bytecode java conforme con le specifiche della JVM).

Una applicazione scritta con codice sorgente Java non è detto che debba essere eseguita su una JVM; ad esempio quando si scrive una applicazione Android, le programmatore scrive file sorgenti java sfruttando librerie tipiche di Android e attraverso javac otteniamo i file .class. Nel caso di Android i file .class vengono dati in ingresso al compilatore dx che produce un eseguibile per una macchina virtuale chiamata Dalvik (basata su registri).




+ Librerie Android (scritte in Java)

Adesso Android usa un meccanismo leggermente diverso: non si produce più un eseguibile per la macchina virtuale Dalvik ma c'è un altro meccanismo chè si chiama ART (Advanced Runtime) in cui nel momento in cui si scarica l'applicazione sullo smartphone viene tradotta direttamente in codice nativo.

JRE = JVM + librerie standard Java

Essendo le applicazioni interpretate, sono generalmente più lente rispetto ad una applicazione che è stata compilata in codice nativo; questo era particolarmente vero per le prime versioni di macchine virtuali, mentre le macchine virtuali recenti sono piuttosto efficienti.