IL LINGUAGGIO DI PROGRAMMAZIONE JAVA

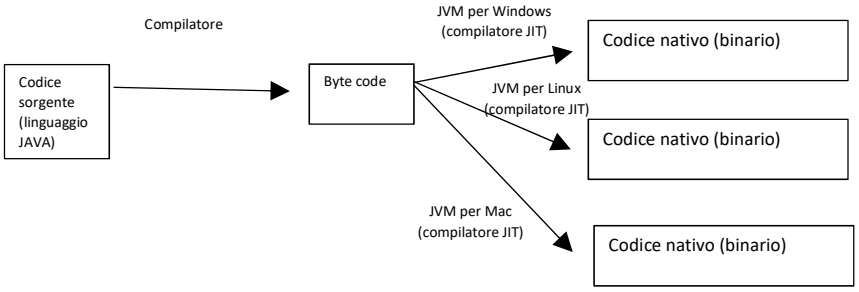
1. Soluzione Java fra compilatore e interprete

Come già visto l'anno scorso, Java ha ideato una soluzione per la traduzione in binario del codice sorgente, soluzione che rende estremamente portabili i software scritti con questo linguaggio di programmazione. Questa "soluzione java" è una "via di mezzo" fra l'approccio compilato e quello interpretato. Il compilatore java non genera un **codice binario** specifico per una determinata piattaforma HW/SW ma un codice "intermedio" chiamato **byte-code**. Il byte-code viene interpretato a runtime (ossia "durante l'esecuzione") da una "macchina virtuale" chiamata **Java Virtual Machine** (**JVM**). Per ogni piattaforma SW (ossia per ogni sistema operativo) esiste una JVM. La JVM va installata sul PC poichè essa è il software che interpreta il byte-code. La JVM rappresenta quindi una **astrazione** di un computer per qualunque piattaforma HW/SW. Il vantaggio di questo approccio è che, quando il programmatore scrive del codice in Java e poi genera il byte code, questo bytecode sarà eseguibile da ciascun computer reale, indipendentemente dal sistema operativo installato, a condizione che la macchina abbia installato la JVM specifica per quel sistema operativo. Per questo motivo il motto di Java è: ***"write once, run everywhere".***

Quando con un doppio click si manda in esecuzione un programma scritto in java, ciò che accade è che il byte-code viene convertito in codice binario dalla JVM. Questa traduzione non coinvolge tutto il bytecode ma riguarda solamente alcune parti, ossia le "parti" che vengono effettivamente eseguite dal processore, e tale traduzione avviene solamente nel momento in cui le funzioni vengono invocate (per questo motivo la traduzione in binario del bytecode è chiamata anche compilazione **Just In Time**, **JIT**, appena in tempo) analogamente a quanto avviene con un software interpretato. Le funzioni invocate, dopo esser state tradotte in binario ed eseguite, rimangono nella memoria centrale per eventuali successive invocazioni durante l'esecuzione, ma se alcune funzioni non vengono mai invocate durante l'esecuzione, esse non vengono mai tradotte da bytecode in binario. Questa modalità di traduzione consente di non dover tradurre in binario parti di codice non necessarie, velocizzando l'esecuzione. La velocità di esecuzione comunque non potrà mai raggiungere i livelli prestazionali di un programma eseguibile compilato **nativamente** (ossia "direttamente in binario" ad esempio in C) a causa dell'ulteriore livello di astrazione introdotto dalla JVM.

L'approccio JAVA è considerato una via di mezzo fra l'approccio compilato e quello interpretato perché:

1. Il codice sorgente viene tradotto in byte code (come nel compilato) ma ciò che si ottiene dalla traduzione non è il codice binario eseguito (a differenza dell'approccio compilato).
2. Il byte code viene tradotto in binario a runtime (come nell'approccio interpretato) ma una sola volta (a differenza dell'approccio interpretato in cui la traduzione istruzione per istruzione avviene ogni volta che l'istruzione viene eseguita dall' esecutore)



Quando nasce il linguaggio Java e perché ha avuto tanto successo?

Il linguaggio Java nasce nella prima metà degli anni '90 del secolo scorso, da un progetto della SUN Mycrosystem (poi acquisita nel 2009 dalla società Oracle) ad opera di un gruppo di lavoro guidato da **James Gosling**, ed è un linguaggio orientato agli oggetti.

Nasce come linguaggio di programmazione **orientato agli oggetti** per dispositivi embedded (ossia processori integrati a bordo di macchinari, ad esempio elettrodomestici, caldaie, automobili ecc.), con una sintassi simile al C/C++ (per facilitarne la diffusione fra i programmatori) ma semplificando alcuni meccanismi che risultavano di non semplice gestione in C++, ad esempio l'uso dei puntatori.

Java deve il proprio successo al fatto che nel 1995, nel periodo di grande incremento di utilizzo della rete internet, il browser Netscape (all'epoca il più diffuso), integrò la tecnologia per eseguire su pc in locale degli applicativi il cui codice era inviato da remoto, le **applet java.** Fino ad allora, le pagine web erano realizzate solamente in html e quindi erano statiche, ossia costituite solamente da testi e link. Per inserire nelle pagine web delle animazioni che le rendessero dinamiche, si trovò la soluzione di inviare, insieme al codice HTML, del codice che potesse poi essere eseguito sul pc client.
Il ragionamento era il seguente: sappiamo che ogni piattaforma hardware/software può eseguire del codice binario solamente se questo è stato compilato per tale specifica piattaforma. Quando un web server invia una pagina web ad un client, per poter inviare anche del codice eseguibile, sarebbe necessario che il web server riconoscesse la piattaforma hw/sw del client che ha richiesto la pagine, e che inviasse del codice binario diverso in base alla piattaforma installata sul client. Questa soluzione è molto complessa e di difficile manutenzione poiché renderebbe necessario modificare i file binari inviati insieme alle pagine web ogni volta che avvenisse una qualsiasi modifica su un qualsiasi sistema operativo. Si scelse allora di dotare i client di un ambiente di esecuzione (la JVM) che, una volta installato, fosse in grado di tradurre nel codice binario opportuno il bytecode inviato attraverso la rete insieme alla pagina web. Questo approccio, oltre ad aumentare estremamente la **portabilità** di un codice (*write once, run everywere*), garantiva un certo livello di **sicurezza** perché la JVM poteva, oltre a tradurre il codice binario, rilevare eventuale codice maligno (ad esempio rilevando istruzioni per la creazione e cancellazione di file) ed eseguirlo solamente se autorizzate dall'utente. I browser più recenti (Chrome e Edge) non supportano più le applet java, ma l'idea di installare la JVM sul PC client si è dimostrata vincente nella realizzazione di applicazioni desktop, tanto da essere poi utilizzata anche da Microsoft nel proprio linguaggio di programmazione ad oggetti C#). L'equivalente della **JVM** di Microsoft si chiama **CLR** (Common Language Runtime).

Quindi, riassumendo, gli elementi che hanno favorito la diffusione del linguaggio java sono:

1. **il fatto che è nato per la OOP e quindi dispone nativamente di numerose classi "già pronte" realizzate per determinate operazioni, ad esempio per la comunicazione in rete.**

2. **Portabilità (indipendenza dalla piattaforma).**

3. **Sicurezza**

2. Cosa sono JRE (Java Runtime Enviroment) e JDK (Java Developement Kit)

(Link al video su questa lezione: https://www.youtube.com/watch?v=odOxIdSguac&list=PLNrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=2)

Che cosa ci serve per eseguire un programma JAVA?

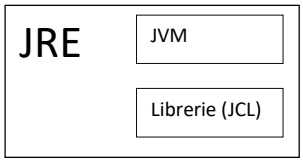
Per **l'esecuzione** di un programma java (ossia di codice in bytecode) è necessario che sia installato sul pc il **Java Runtime Enviroment** (**JRE**, ambiente di esecuzione java), esso comprende:

- la JVM
- JCL (Java Class Library): alcune librerie di classi caricabili dinamicamente che possono essere richiamate a runtime.

Se il nostro pc in passato ha già eseguito dei programmi in java, il JRE è già installato. Il JRE è ciò che installiamo quando ci viene chiesto, per eseguire un software, di "scaricare JAVA".
Per verificare se il JRE è già installato sul nostro PC:

menu cerca/configura Java → si apre il pannello di java, nella scheda java ci sono le informazioni del JRE

La versione più recente del JRE nel 2023 è la versione 8.
Se sul nostro PC non è installato il JRE non è un problema perché tale piattaforma è inclusa nel JDK, che ci serve per **programmare** in java, e che installeremo dopo.



Che cosa ci serve per scrivere programmi in java?

Ci serve un compilatore java, ossia il software che traduce il codice sorgente, in bytecode. In realtà, oltre al compilatore si scarica dal sito della Oracle un **JDK** (Java Developement Kit), che contiene diversi software di utilità per la programmazione:

- Compilatore (**Javac**)
- **Javadoc**: software per la documentazione del codice (vedremo in TPS)
- **Jar**: Java archivier, software che prepara un file compresso contenente una serie di librerie fra loro collegate

Qual è la differenza tra JRE e JDK ?

JRE **(Java Runtime environment)**	**JDK** **(Java Development Kit)**
È un'implementazione della Macchina virtuale Java* che esegue effettivamente i programmi Java.	È un insieme di software che potete utilizzare per sviluppare le applicazioni basate su Java.
Java Runtime Environment è un plug-in necessario per l'esecuzione dei programmi Java.	Java Development Kit è necessario per sviluppare nuove applicazioni Java.
JRE è più piccolo di JDK, quindi necessita di meno spazio sul disco.	JDK richiede più spazio sul disco poiché contiene JRE tra i diversi strumenti di sviluppo.
JRE può essere scaricato/supportato gratuitamente da java.com	JDK può essere scaricato/supportato gratuitamente da oracle.com/technetwork/java/javase/downloads/
Include JVM, le librerie di base e altri componenti aggiuntivi per eseguire le applicazioni e le applet scritte in Java.	Include JRE, l'insieme delle classi API, il compilatore Java, Web Start e i file aggiuntivi necessari per scrivere le applet e le applicazioni Java.

Approfondimento: quale è la società che fornisce l'ambiente di sviluppo JDK?
Come detto, il progetto Java è stato inizialmente sviluppato dalla SUN Mycrosystem. Tale società è stata poi acquisita dalla società Oracle. Quindi Oracle distribuisce il JDK. La versione che utilizziamo noi (chiamata JDK SE che sta per Standard Edition) è distribuita gratuitamente con licenza open source. La versione "professionale" utilizzata per sviluppare grandi applicazioni è chiamata JDK EE (Enterprise Edition). Di questa versione esistono sia release distribuite con licenza open source sia release con particolari funzionalità che vengono distribuite a con licenza proprietaria, quindi a pagamento. Link al download di Oracle: https://www.oracle.com/it/java/technologies/downloads/

Oltre ad Oracle esiste un'altra società che distribuisce con licenza open source sia il JDK SE sia il JDK EE. Tale società si chiama RedHat e la sua versione del JDK si chiama OpenJDK.
Il JDK di Oracle e OpenJDK vengono sviluppati parallelamente e non hanno differenze significative nel loro utilizzo. Link al download delle versioni di OpenJDK: https://developers.redhat.com/products/openjdk/download#assembly-field-downloads-page-content-82031
Esistono poi altre società che rilasciano le proprie distribuzioni dei JDK basate su OpenJDK.

Citazione dal sito di RedHat:
"La più grande differenza tra OpenJDK e Oracle JDK è che OpenJDK è un progetto open source gestito da **Oracle, da Red Hat e dalla comunità**, mentre Oracle JDK è closed source, richiede una licenza a pagamento ed è gestito da **Oracle**. Con questa differenza, ci sono alcune funzionalità che non sono disponibili con OpenJDK poiché le funzionalità sono closed source o limitate dalla licenza."
https://www.redhat.com/en/topics/application-modernization/openjdk-vs-oracle-jdk#:~:text=The%20biggest%20difference%20between%20OpenJDK,and%20is%20maintained%20by%20Oracle.

Le versioni del JDK di Oracle sono aggiornate ogni 6 mesi. Alcune versioni sono più importanti di altre poiché per tali versioni viene garantito il supporto (aggiornamenti, correzione bachi ecc.) a lungo tempo.