Introduzione a Java
"Write Once, Run Anywhere"

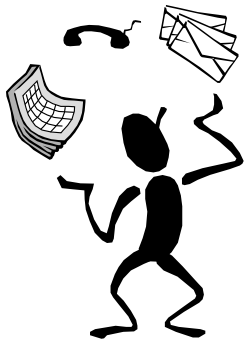
Programmazione II corso A
Corso di Laurea in ITPS
Università degli studi di Bari Aldo Moro
a.a. 2022-2023
ultima modifica: 26 ottobre 2022

Caratteristiche generali del linguaggio

i principali punti di forza di Java sono la portabilità e la sicurezza, pur tuttavia esistono altri fattori chiave nella determinazione del linguaggio

- Semplicità
- Object-Oriented
- Solidità
- Multi-threading
- Indipendenza dall'architettura
- Interpretazione con elevate prestazioni
- Possibilità di realizzare applicazioni distribuite
- Dinamicità

Semplicità



- Object-Oriented puro
  - altri linguaggi, invece, sono "ibridi"; ad esempio il C++ è procedure-oriented ed object-oriented
- libreria standard estremamente ampia e ben organizzata;
- solo tre gruppi di tipi primitivi (numerico, boolean e array)
- classi specifiche per le stringhe
- dimensioni contenute;
  - l'interprete Java occupa circa 40 kByte di memoria RAM, escludendo il supporto per il multithreading e le librerie standard, che richiedono altri 175 kByte

Orientato agli oggetti

Java è un linguaggio puramente orientato agli oggetti.

Supporta quindi le principali caratteristiche del paradigma Object-Oriented quali:

- incapsulamento delle informazioni (information hiding);
- ereditarietà;
- polimorfismo;
- binding dinamico

Java soddisfa questi requisiti perfettamente, inoltre aggiunge un considerevole supporto a run-time per rendere più facile il processo di sviluppo del software.

Solidità…

- Fortemente tipizzato (viene effettuato il controllo statico dei tipi):
  - il compilatore Java effettua particolari e severi controlli in fase di compilazione, in modo che ogni errore possa essere individuato prima che il programma sia eseguito, in Java tutte le dichiarazioni di tipo devono essere esplicite. Il linker di Java ripete tutte le operazioni di controllo di tipo per evitare incongruenze di interfaccia o di metodi
- Gestione automatica della memoria, liberando da tale responsabilità il programmatore, attraverso il meccanismo di *garbage collection*;

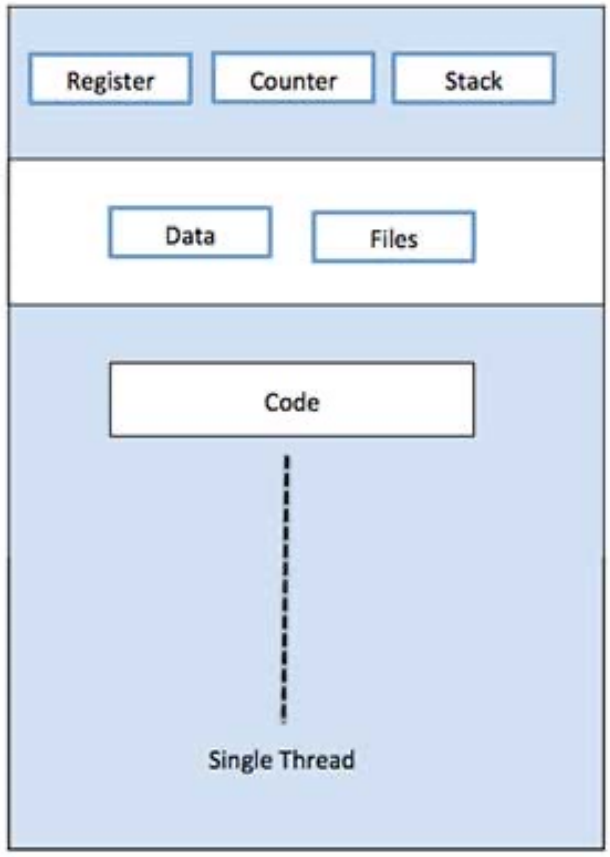
…Solidità.

- Gestione delle eccezioni orientata agli oggetti
  - in un programma Java ben scritto tutti gli errori run-time dovrebbero essere gestiti dal programma.

Processo con singolo thread

Un thread è un singolo flusso sequenziale di istruzioni all'interno di un programma che utilizza dati e files di tale programma e si serve di un insieme di *register*, di uno *stack* e di un *program counter*

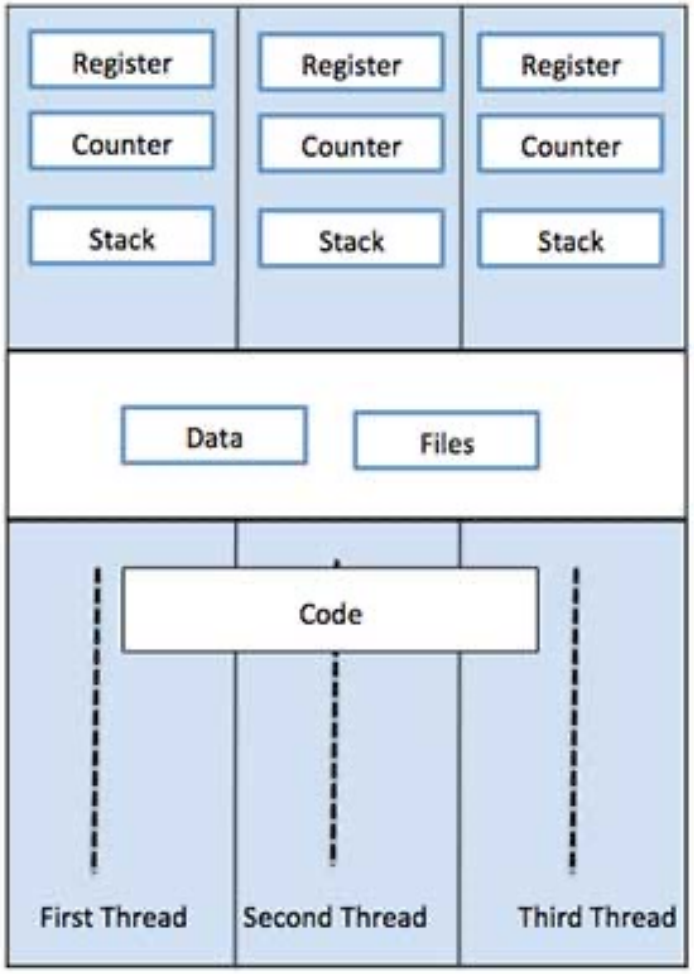
- ogni programma esegue almeno un thread;
- quando un processo esegue uno ed un solo thread è definito processo leggero (lightweight)



Single Process P with single thread

processo Multi-thread

- Quando il programma principale avvia un secondo thread tale programma diventa multi-thread
- Ogni thread può avviare un qualunque numero di thread
- I thread condividono dati e files del programma in cui sono eseguiti
- Ogni thread utilizza un proprio insieme di registri, un proprio stack ed ha un proprio program counter
- Un thread è legato ad un singolo programma



Single Process P with three threads

Multi-threading: esempio

Esempio

Il thread che tratta gli eventi del mouse dovrebbe avere un'alta priorità nel sistema operativo. In tal modo, il mouse otterrà più attenzione da parte della CPU e l'utente avrà un mouse più sensibile che non si blocca quando inizia un programma che occupa gran parte del tempo della CPU.

Java abilita il programmatore a creare un programma che stabilisce come la CPU dovrà trattare un thread. Quindi permette di creare e distruggere thread, di definire delle priorità, di sincronizzare i thread.

Multi-threading: sincronizzazione

La sincronizzazione è indispensabile per evitare i conflitti che nascono dall'accesso simultaneo, da parte di più thread, ad un' unica risorsa non condivisibile.

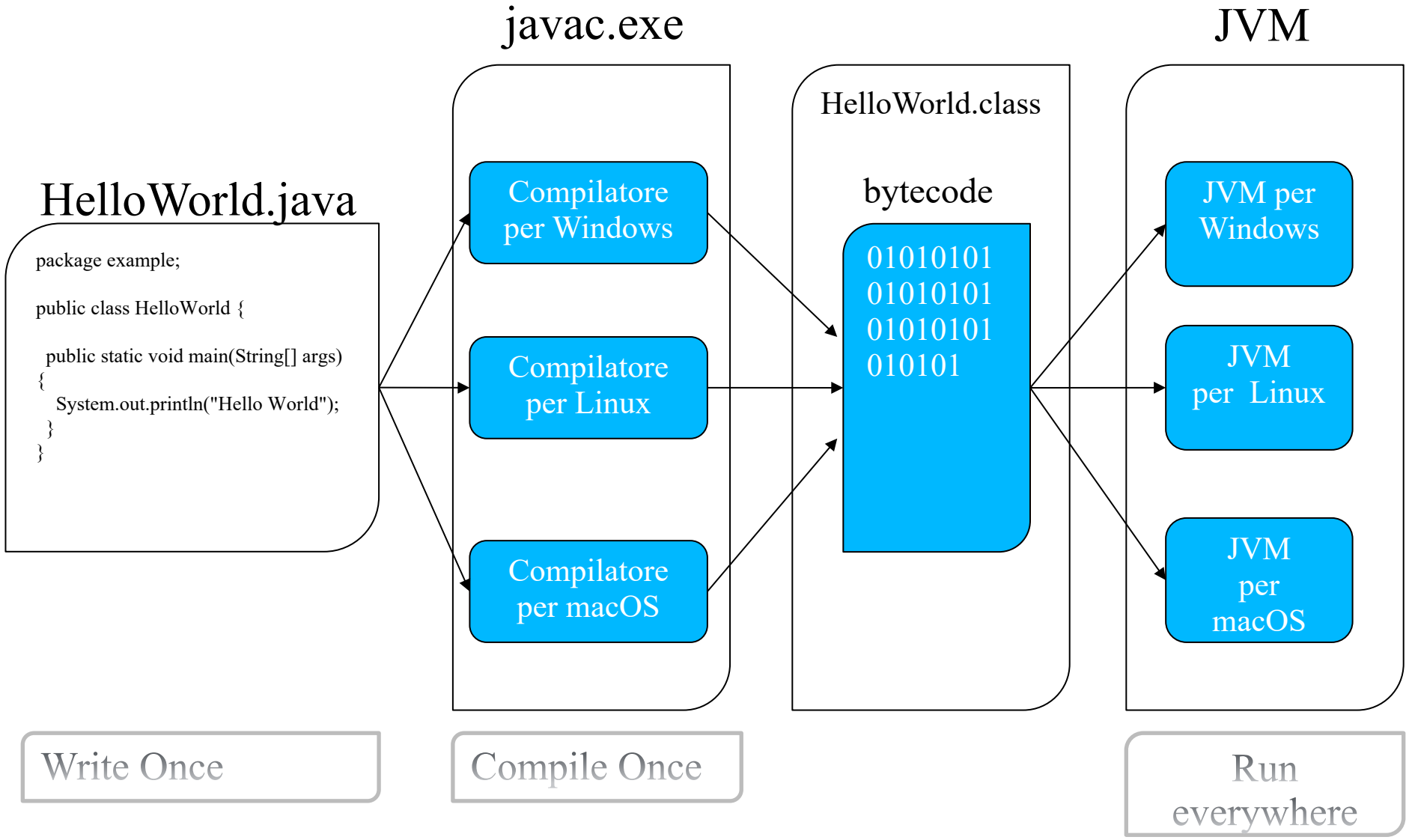
- Il vantaggio principale di tutto ciò si avverte particolarmente in alcune applicazioni multimediali.
- I sistemi operativi che supportano il multithreading sono Windows 9x/NT/2000/XP, OS2, Solaris 2 (Unix) ma non il DOS. Questo spiega l'iniziale difficoltà a far uscire una versione di Java per Windows 3.1

Indipendenza dall'architettura…

Un programma Java può essere eseguito su architetture diverse. Esiste una indipendenza a livello di:

- *codice sorgente*:
  - tipi di dati primitivi completamente specificati
  - ordine di valutazione delle espressioni rigorosamente definito
- *bytecode*
  - un programma Java viene compilato in un codice intermedio detto **bytecode**. I file di codice intermedio sono interpretati dalla *Java Virtual Machine (JVM)*.
  - È sufficiente implementare una JVM specifica per ogni piattaforma, senza per questo dover modificare il bytecode.

…Indipendenza dall'architettura



Interpretazione con elevate prestazioni…

- Il bytecode può essere interpretato su un qualsiasi sistema che offra una Java Virtual Machine.
- Pur avendo delle prestazioni inferiori ai linguaggi direttamente compilati, il bytecode di Java è stato progettato in modo tale da poter essere tradotto senza difficoltà in codice macchina nativo garantendo così delle prestazioni abbastanza elevate
  - Altri sistemi interpretati come Basic e Perl presentano deficit di prestazioni insormontabili

…Interpretazione con elevate prestazioni.

- i sistemi run-time Java che eseguono questa ottimizzazione "just in time" o "on-fly" non perdono nessuno dei vantaggi offerti dal codice indipendente da piattaforma e allo stesso tempo hanno delle buone prestazioni.

NOTA BENE

la compilazione just-in-time è oggetto di ricerca. I compilatori convenzionali sono pensati per produrre codice altamente ottimizzato senza prestare attenzione al tempo di compilazione.

Possibilità di applicazioni distribuite

- metodi per gestire l'accesso ad informazioni remote utilizzando protocolli come HTTP ed FTP;
  - package *java.net*
- possibilità di invocare metodi di oggetti remoti (appartenenti a processi diversi da quello corrente, potenzialmente su macchine diverse) mediante RMI (*remote method invocation*)
  - conferisce un ottimo livello di astrazione alla programmazione client-server
  - package *java.rmi*

Dinamicità: perdita della memoria

Uno dei problemi principali dei linguaggi di programmazione è quello della *perdita della memoria (memory leak)*.

Quando un programma parte, esso prende della memoria libera dall'ambiente per istanziare gli oggetti. In seguito durante l'esecuzione, esso continua a creare e a distruggere oggetti, occupando e liberando aree di memoria.

Al termine dell'esecuzione il programma distrugge gli oggetti rimanenti, rilasciando tutta la quantità di memoria occupata durante l'esecuzione.

Dinamicità: perdita della memoria

Nella maggior parte dei linguaggi di programmazione ciò non accade perché spesso l'ambiente utente subisce una *perdita di memoria* a causa del fatto che (vedi C++) è lasciata al programmatore la responsabilità di tenere traccia dell'allocazione degli oggetti in memoria aumentando così la possibilità di errori per programmi complessi.

In Java il programmatore è sollevato da questa compito perché la gestione della memoria è affidata ad un *garbage collector*.

Dinamicità: garbage collection…

Java ha due tipi di strumenti per il *garbage collection*:

- *compaction*: analogo al processo di deframmentazione della memoria usato per gli hard disk;
- *marking-and-sweeping*: è un processo che si compone di due passi:
  - Inizialmente si marcano gli oggetti in uso
  - Se alla fine del processo di marking un oggetto non è marcato, viene spazzato via (swept) o distrutto.

…Dinamicità: garbage collection

Il *garbage collection* non è una novità introdotta da Java (vedi linguaggi come Lisp e SmallTalk) tuttavia essendo Java concepito per un ambiente multithreading esegue il suo garbage collector su un thread separato rendendolo quasi impercettibile in elaborazioni real-time.

Tale thread riceve una bassa priorità cosicché si attiverà solo quando la CPU ha del tempo disponibile.

Java ed Internet

Java è un linguaggio ideale per Internet perché garantisce:

- sicurezza
- portabilità
- dinamicità

Java può essere utilizzato per creare differenti tipi di programmi quali:

- applicazione stand-alone;
- applet

Java ed Internet: applet…

- Un'applicazione Java è un programma equivalente ad un altro creato attraverso linguaggi di programmazione (C,C++) e viene eseguito su un computer.
- Un'*applet* è una applicazione concepita per essere trasmessa su Internet ed eseguita da un browser Web compatibile con Java
- L'applet è prelevato dinamicamente dalla rete ed è capace di reagire agli input dell'utente e di permettere lo scambio di informazioni su Internet.
- Gli applet sono sottoposti a rigidi vincoli di sicurezza e portabilità.

Java ed Internet: applicazione stand-alone

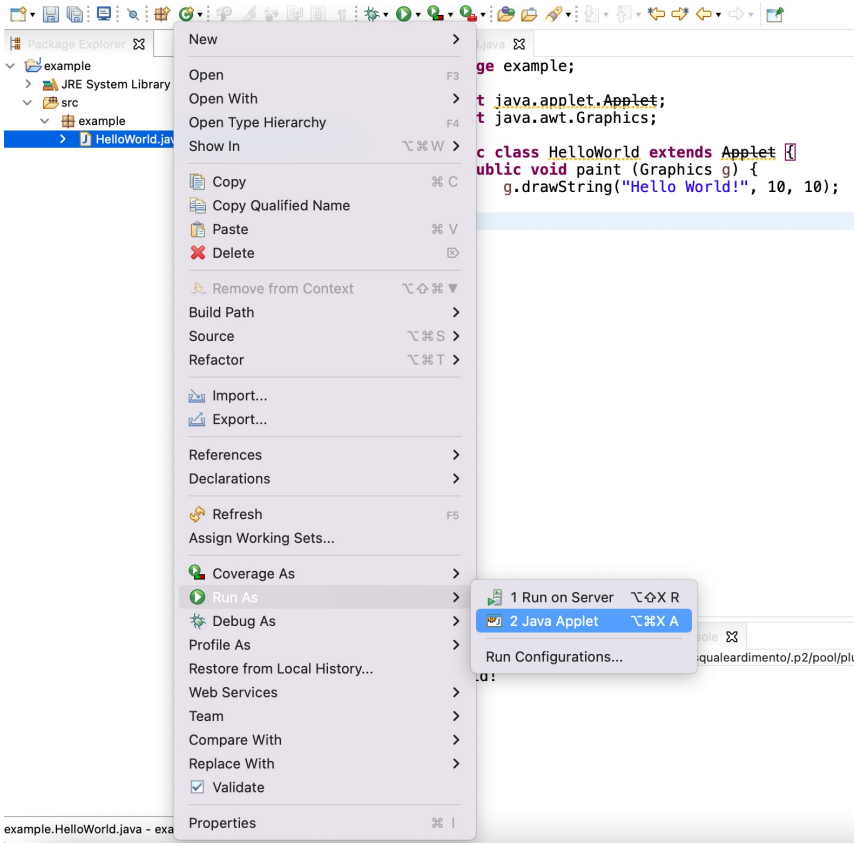
```
package example;

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

Java ed Internet: applet Java

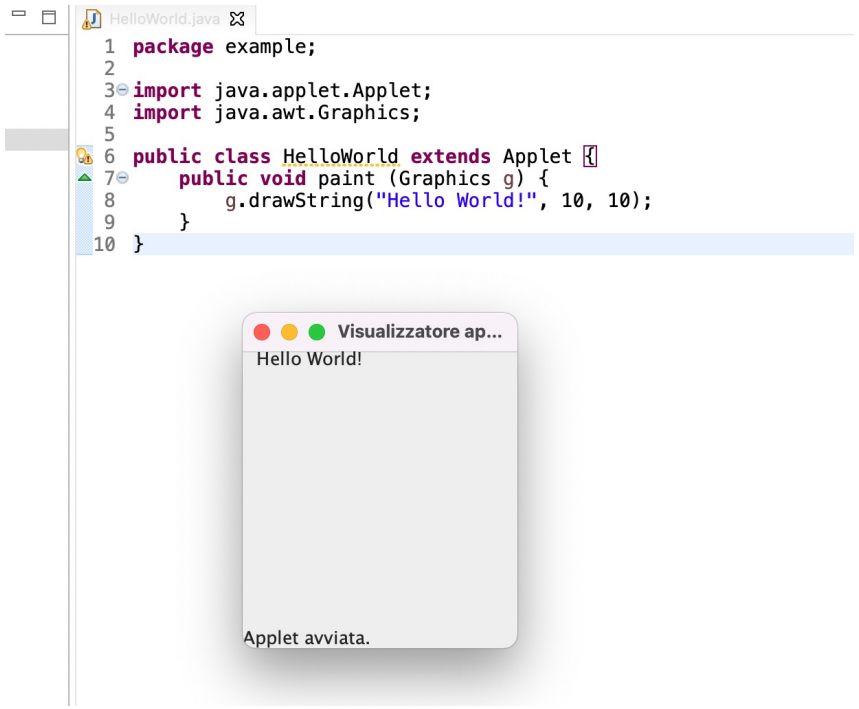
```
package example;

import java.applet.Applet;
import java.awt.Graphics;

   // N.B.: la classe Applet è disapprovata dalla JDK 9 in poi
   public class HelloWorld extends Applet {
      public void paint (Graphics g) {
          g.drawString("Hello World!", 10, 10);
   }
}
```

Esecuzione applet HelloWorld

Esecuzione

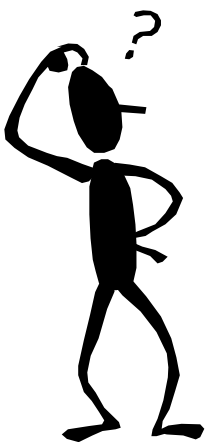


Output



Problema: la Sicurezza

Java è progettato per ambienti di rete quindi si è prestata particolare cura alle caratteristiche di sicurezza.



Se si esegue un programma Java prelevato da Internet, chi ci assicura che non sia affetto da virus, o, più in generale, che la sua esecuzione non abbia effetti collaterali sovversivi sul sistema?

I livelli di Sicurezza: allocazione della memoria

In Java sono stati introdotti i seguenti livelli di sicurezza:

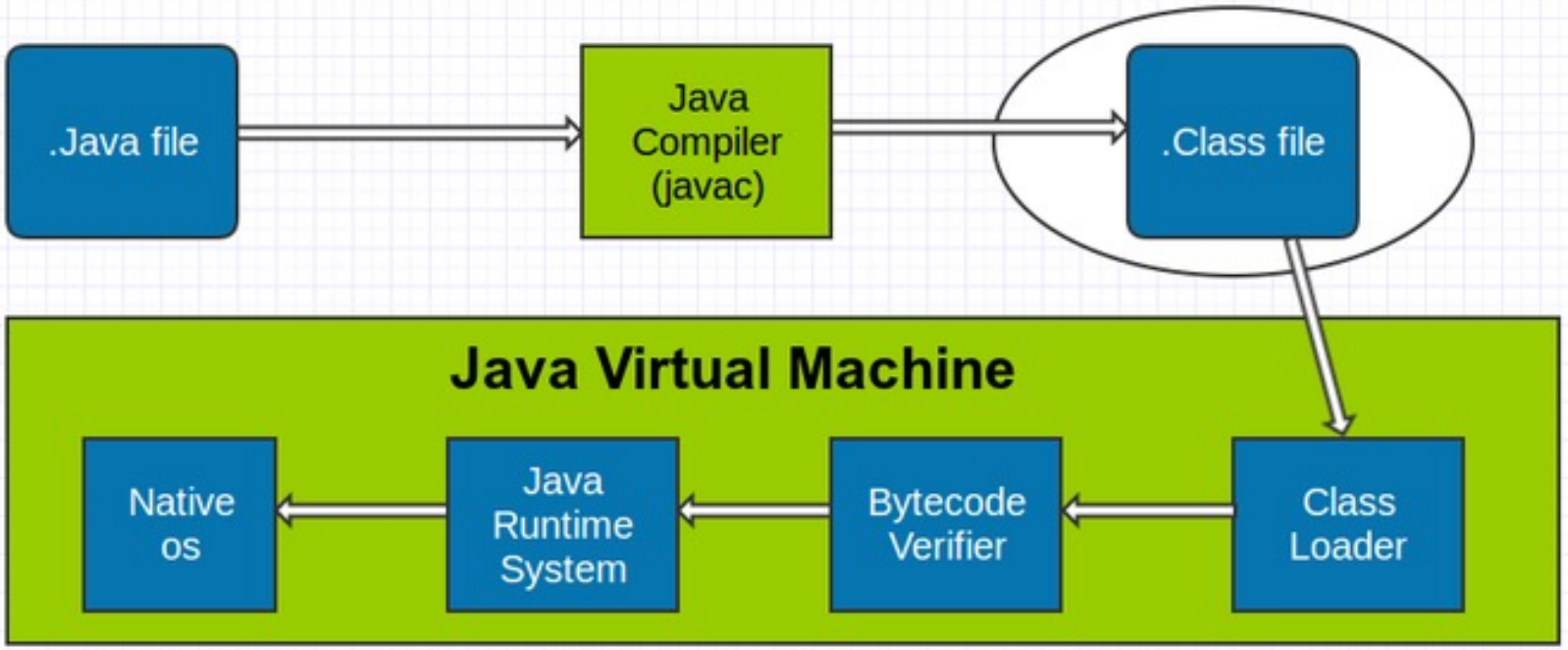
Linguaggio e compilatore

- I tradizionali compilatori C e C++ prendono le decisioni relative all'allocazione di memoria al momento della compilazione. Ne consegue che gli hacker hanno, in anticipo, molta informazione disponibile.

I livelli di Sicurezza: allocazione della memoria

- Al contrario, in Java, tali decisioni sono differite al momento dell'esecuzione, sicché l'accesso al codice sorgente Java non fornisce delle informazioni da usare in modo malizioso.
- L'inesistenza di puntatori nel linguaggio, impedisce anche la possibilità di creare danni, anche involontari, ad altri programmi in memoria.

I livelli di sicurezza: ClassLoader



- Il ClassLoader carica i file di classe sia dal programma che dalle API Java.
- Nella macchina virtuale vengono caricati solo i file di classe delle API Java che sono effettivamente necessari per il programma in esecuzione.

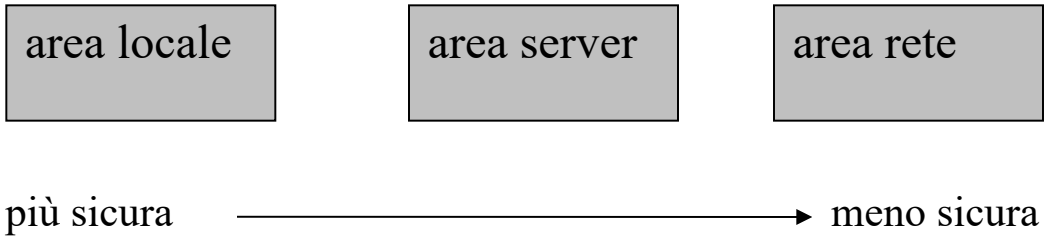
I livelli di Sicurezza: Class Loader

Un'applet/applicazione Java potrebbe chiamare molte classi differenti, come:

- classi predefinite caricate dal file-system locale;
- classi provenienti dallo stesso server di rete che ha inviato l'applet;
- classi scaricate da altri siti Web

Per ulteriore sicurezza, ciascuna classe importata è eseguita in uno spazio apposito

…I livelli di Sicurezza: la soluzione…

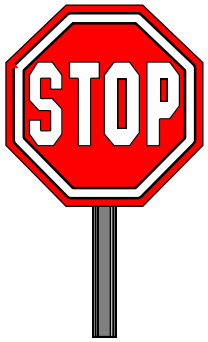


Esempio

Le primitive di I/O del sistema di gestione dei file sono definite in una classe Java locale quindi sono attive nell'area computer locale.

…I livelli di Sicurezza: la soluzione…

Quando una classe fa riferimento ad un'altra, il classloader cerca prima nell'area locale in modo tale che non ci sia modo di caricare dalla rete una classe che si potrebbe sovrapporre a quella locale

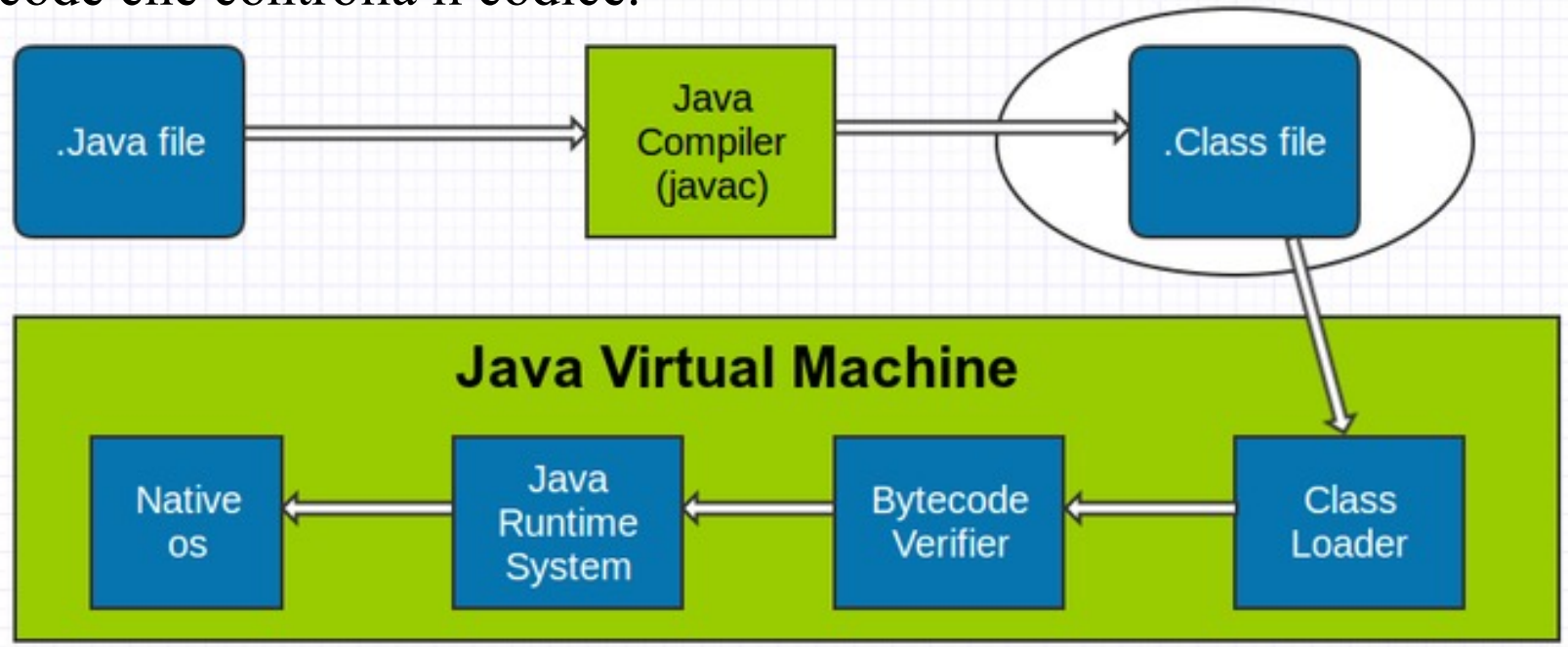


Non si consente mai ad una classe proveniente da un'area meno protetta di sostituire una classe di un'area più protetta

I livelli di sicurezza: verificatore di bytecode

Visto che il codice Java può essere caricato da un sito potenzialmente inaffidabile, l'ambiente Java deve prendere anche in considerazione la possibilità di compilatori Java truccati.

Pertanto i programmi scaricati sono passati attraverso un verificatore di bytecode che controlla il codice.



I livelli di Sicurezza: verificatore di bytecode

I controlli effettuati sono i seguenti:

- non ci devono essere puntatori a oggetti illegalmente manipolati al di fuori della JVM;
- non si deve accedere ad oggetti di tipo scorretto
  - ad esempio oggetti String sono sempre usati come String e mai in nessun altro modo
- devono essere rispettate le protezioni imposte sui campi dell'oggetto (*private, public* o *protected*)

I livelli di sicurezza: ulteriori controlli

- un applet Java non può leggere e scrivere sul disco del sistema locale;
- un applet Java non può eseguire alcun programma residente sul sistema locale;
- un applet Java può collegarsi solo con il server dal quale è trasferito;

Java vs. C++

Java è strettamente connesso al C++, ma comunque tra questi due linguaggi ci sono delle differenze sostanziali.

Java vs C++: Java non usa i puntatori

Questa modalità di accesso indiretto alle variabili, insieme ai tipi definiti dall'utente (*user defined type, UDT*), ha costituito un punto di forza del C++ ed al tempo stesso è stata la fonte di molti problemi.

I puntatori sembrano essere una delle strade preferite per l'inserimento di *bug* in un programma, sono spesso causa di improvvisi *crash* del sistema e inoltre hanno la capacità di accedere a delle informazioni a cui non sono autorizzati.

Java vs C++: Java non supporta l'ereditarietà multipla

- Il C++ supporta l'ereditarietà multipla ovvero una sottoclasse può ereditare metodi e attributi da più di una superclasse.
  - Questa soluzione pur essendo a prima vista vantaggiosa, si è rivelato troppo complicata da gestire pertanto Java è stato dotato solo di ereditarietà singola fra le classi, mentre supporta l'ereditarieà multipla fra le interfacce.

L'implementazione di più interfacce o la aggregazione tra classi da parte di una classe permette in qualche modo di sopperire alla carenza di meccanismi di ereditarietà multipla.

Java vs C++: Java ha il tipo stringa

In C++ (standard) bisogna definire il tipo stringa come array di caratteri quindi se non ci sono controlli sui limiti degli indici, nel caso di inserimento di stringhe più grandi dell'area di memoria a loro riservata, si potrebbero verificare dei comportamenti anomali.

In Java le stringhe sono oggetti veri e propri e allo stesso tempo viene fornita una sintassi semplificata per la loro gestione come se fossero tipi primitivi.

Java vs C++: Altre principali differenze

- Java ha una gestione della memoria migliore (garbage collection) rispetto al C++;
- Java supporta l'overloading delle funzioni ma non degli operatori;
- Java consente come uniche forme di coercizioni, ovvero le conversioni implicite di tipo, l'autoboxing e l'unboxing. La conversione di tipo deve essere esplicita (mediante cast);
- I file header sono rimpiazzati dalle interfacce;
- Tutte le funzioni devono essere contenute in qualche classe

Java vs C++: Conclusioni

Riassumendo possiamo concordare con Bill Joy (vicepresidente Sun) che definì Java:

"… un 'C++' - -"

Ambienti di sviluppo: JDK

- Esistono diversi ambienti di sviluppo per Java, il primo, denominato Java Development Kit (JDK) fu messo a disposizione dalla Sun, è il più diffuso.
  - L'ultima generazione del Java Development Kit (17.0.1) è la base della famiglia di piattaforme Java di Oracle.
  - Ogni altro ambiente di sviluppo per Java aggiunge funzionalità, nuovi *tool*, nuove classi, pur avendo come piattaforma base un Java Development Kit.

Ambienti di sviluppo: JDK

Tra le caratteristiche del software Java abbiamo:

- compatibilità verso l'alto;
  - il bytecode Java è compatibile in avanti ossia codice scritto con una versione di Java può essere eseguito da versione successive della JVM
- maggiore stabilità e migliori prestazioni;
- librerie di classi e tool di base per costruire applicazioni distribuite portatili per qualsiasi piattaforma Java, indipendentemente dal fatto che l'esecuzione avvenga su una smart card, un dispositivo consumer, un PC stand alone o un computer in rete;

Ambienti di sviluppo: JDK

- codice indipendente dalla piattaforma che può essere eseguito su qualsiasi JVM.
  - Attualmente, sono disponibili oltre 20 macchine virtuali Java per le piattaforme informatiche più diffuse;
- creazione di un nuovo 'look-and-feel' per la creazione di interfacce utente coerenti su diverse piattaforme;

Ambienti di sviluppo: JDK

- tecnologia Java Archive (Jar)
  - un formato file indipendente dalla piattaforma che aggrega più file in un singolo package. Nello stesso file Jar è possibile incorporare più applet (con i relativi componenti) migliorando notevolmente le velocità di download
- inclusione di un formato Jar avanzato con funzionalità addizionali per la creazione e l'aggiornamento di file Jar firmati. Inoltre, sono disponibili nuove API standard per la lettura e la scrittura di file Jar;

Ambienti di sviluppo: JDK

- sicurezza estesa con l'introduzione di tre nuovi tool di
  - *keytool*: crea coppie di chiavi pubblica/privata, visualizza, importa ed esporta certificati e genera certificati X.509 autofirmati;
  - *jarsigner*: firma i file Jar e verifica le firme dei file Jar firmati;
  - *policytool*: crea e modifica i file di configurazione che definiscono una politica di sicurezza Java.

Ambienti di sviluppo: JDK

Java SE 17.0.1 distribuito gratuitamente dalla Oracle e scaricabile a partire dal seguente indirizzo:

- http://www.oracle.com/technetwork/java/javase/downloads/index.html
- In particolare sono disponibili le versioni per Solaris, Windows, Linux, Mac OS X x64

Java SE Development Kit 8
(versione non ultima)

http://www.oracle.com/technetwork/java/javase/jdk-8-readme-2095712.html

- Modularizzazione (Java SE è scomposto in moduli più piccoli e indipendenti)
- Supporto multi-linguaggio (per esempio Ruby e Python)
- Produttività sviluppatore (concorrenza, collection, type annotation per migliorare controllo statico di correttezza, …)
- Prestazioni (G1 garbage collector, …)

Ambienti di sviluppo: JDK

Oracle struttura la piattaforma Java in due edizioni:

- Java Runtime Environment (JRE)
  - rivolta a utenti di applicazioni Java
  - permette di eseguire applicazioni Java per cui non include il compilatore e altre componenti di sviluppo
- Java Software Development Kit (Java SDK)
  - rivolta a sviluppatori di applicazioni Java
  - mette a disposizione degli sviluppatori il compilatore, le librerie, i sorgenti, le specifiche API e, in generale, tutto quanto è necessario per lo sviluppo di applicazioni Java.

Ambienti di sviluppo: tipi di JDK

Fino alla versione 8 esistevano tre tipi di JDK:

- Java Platform, Standard Edition (Java SE)
  - Applicativi e librerie per creare applicazioni desktop
- Java Platform, Enterprise Edition (Java EE)
  - Librerie per creare applicazioni di rete multi-tier ed applet
    - Enterprise JavaBeans (EJB), Servlets, JSP (Java Server Pages), JDBC, Java Persistence API, JAX-RS RESTful web services
- Java Embedded
  - permette di eseguire applicazioni Java su dispositivi che hanno capacità di memoria e potenza di calcolo ridotte:
    - Moduli wireless per M2M, Controllori industriali, Smart card , Sensori di ambiente, telefoni cellular

Dalla JDK 9 in poi esiste un'unica JDK contenente tutte le librerie sopra elencate

JDK: tool di base…

Gli strumenti di base presenti nel JDK sono:

- *javac* compilatore Java
- *java* interprete Java
- *javadoc* tool che analizza sintatticamente le dichiarazioni e i commenti presenti in un insieme di file sorgenti e produce un insieme di pagine HTML contenenti informazioni sulle classi, interfacce, costruttori e metodi utilizzati.

…JDK: tool di base…

- *appletviewer* esegue e fa il debug degli applet senza l'ausilio di un browser;
  - dalla JDK 11.x non è più presente
- *jar* gestisce i file Java Archive (JAR)
- *jdb* rappresenta il debugger Java
- *javah* generatore di file header C per il codice nativo
- *javap* disassemblatore dei file .class (bytecode)
- *extcheck* strumento di servizio (utility) per rilevare i conflitti nei file JAR

Java: ambienti di sviluppo evoluti

Tra quelli più diffusi ricordiamo:

- Forte distribuito dalla Sun;
- JBuilder distribuito dalla Borland;
- Visual Age distribuito dalla IBM
- VisualJ++ distribuito da Microsoft
- JDeveloper particolarmente adatto per lavorare su DataBase Oracle
- Eclipse IDE

Eclipse

- Ambiente di sviluppo Open-Source scritto in Java.
- Nasce da VisualAge
- Più semplice e meno pesante di VisualAge.
- Facile estendibilità dell'applicativo mediante plug-in.
- E' disponibile per numerose piattaforme.

Riferimenti Bibliografici

- Bruce Eckel, *Thinking in Java,* Prentice-Hall, 2000
- https://www.oracle.com/java/moved-by-java/timeline/ (pagina acceduta l'ultima volta il 02 dicembre 2020)