LEZIONE 2
JAVA: LINGUAGGIO && PIATTAFORMA

Ingegneria del Software e Progettazione Web

Università degli Studi di Tor Vergata - Roma

Guglielmo De Angelis
guglielmo.deangelis@isti.cnr.it

Java

- originalmente proposta e sviluppata da Sun Microsystems
- attualmente Oracle è il fornitore/sviluppatore ufficiale della tecnologia Java
  - a seguito dell'acquisizione di Sun

"Java is … … the ***global standard*** for developing and delivering mobile applications, games, Web-based content, and enterprise software."

"Java enables you to efficiently develop and deploy exciting applications and services. With ***comprehensive tooling***, a ***mature ecosystem***, and robust performance, Java delivers applications ***portability*** across even the most disparate computing environments."

[excerpts form banners at https://www.oracle.com/java/]

storia

- Nascita ’91 James Gosling e Patrick Naughton progettarono linguaggio per dispositivi consumer (switchbox per televisione via cavo)
  - Progetto Green
  - Linguaggio ***Oak***
    - JVM da idea di Niklaus Wirth
    - Basato su C++
- ’92 progetto Green rilasciò prodotto “*7”
  - Non riscosse successo
- ’94 di fatto il progetto Green chiuse i battenti
- Metà ’94 Gosling&C. si resero conto che con l’avvento del web dovevano realizzare un browser portabile e sicuro (HotJava con applet)
- 23 maggio 1995 SUNWorld ’95 presentarono la nuova tecnologia

storia

- ’96 SUN rilasciò la prima versione di Java 1.0.2
- dopo poco 1.1 che introduceva notevoli migliorie
- '98 viene rilasciata la versione 1.2
  - Java 2 Standard Edition
    Software Development Kit Version 1.2
- ...
  - 1.3
  - 1.4
  - 1.5 i.e. J2SE 5.0
- '06 Java SE 6
- '11 Java SE 7
- '14 Java SE 8 (LTS)
- ‘17 Java SE 9
- 03/18 Java SE 10
- 09/18 Java SE 11 (LTS)
- 03/19 Java SE 12

linguaggio java

<< … un linguaggio semplice e familiare, orientato agli oggetti, robusto, sicuro, architettura neutrale, portabile, ad alte prestazioni, interpretato, multithreaded e dinamico … >>

[J. Gosling e H. McGilton, “The Java Language Environment. A White Paper.” Maggio 1996]

linguaggio java

<< … un linguaggio ***semplice e familiare***, orientato agli oggetti, robusto, sicuro, architettura neutrale, portabile, ad alte prestazioni, interpretato, multithreaded e dinamico … >>

[J. Gosling e H. McGilton, "The Java Language Environment. A White Paper." Maggio 1996]

- rimozione di alcune caratteristiche di dubbia utilità, comuni ai suoi progenitori C e C++
  - a livello di sintassi
  - a livello di espressività del linguaggio
- tra gli intenti originali, l'obiettivo : "... sviluppatori in ambienti C, Objective C, C++, Eiffel, Ada, etc. acquisteranno familiarità con Java in breve tempo, dell'ordine di qualche settimana"

linguaggio java

<< … un linguaggio semplice e familiare, ***orientato agli oggetti***, robusto, sicuro, architettura neutrale, portabile, ad alte prestazioni, interpretato, multithreaded e dinamico … >>

[J. Gosling e H. McGilton, "The Java Language Environment. A White Paper." Maggio 1996]

- cerca di estrarre e fondere i migliori concetti e caratteristiche dai precedenti linguaggi orientati agli oggetti (e.g. Eiffel, SmallTalk, Objective C, C++)
- con l'eccezione dei suoi tipi di dato primitivi, ogni cosa in Java è un oggetto
  - anche i tipi di dato primitivi possono essere incapsulati in oggetti se necessario
- supporta i 4 fondamenti dei linguaggi orientati agli oggetti : *incapsulamento, ereditarietà, binding dinamico e polimorfismo*

linguaggio java

<< … un linguaggio semplice e familiare, orientato agli oggetti, ***robusto, sicuro***, architettura neutrale, portabile, ad alte prestazioni, interpretato, multithreaded e dinamico … >>

[J. Gosling e H. McGilton, "The Java Language Environment. A White Paper." Maggio 1996]

- un software si definisce robusto se si comporta "bene" in condizioni di lavoro "non previste" dallo sviluppatore
  - ***bene***: non sbaglia calcoli, non va in crash, non raggiunge situazioni di stallo (deadlock), etc.
  - ***condizioni non previste***: carico di lavoro eccessivamente elevato, input non conforme allo standard, etc.
- Java incoraggia la costruzione di un software robusto ponendo delle restrizioni riguardanti i tipi di dato e l'uso dei puntatori (es. non è possibile convertire arbitrariamente un intero in un puntatore mediante l'operatore cast)

linguaggio java

<< … un linguaggio semplice e familiare, orientato agli oggetti, ***robusto, sicuro***, architettura neutrale, portabile, ad alte prestazioni, interpretato, multithreaded e dinamico … >>

[J. Gosling e H. McGilton, "The Java Language Environment. A White Paper." Maggio 1996]

- Il compilatore e il sistema di run-time implementano strategie che mitigano la definizione di comportamenti non voluti :
  - la gestione della memoria non è delegata al compilatore (come in C o C++), ma rinviata al run-time;
  - accesso diretto HEAP impossibile (no algebra di puntatori);
  - tutto deve essere esplicito (e.g. strong typing e casting)
  - tutte le classi locali sono poste in un name space distinto da quello per le classi scaricate dalla rete;
  - le classi importate non possono "spiare" quelle locali;
  - sul codice importato viene eseguito la bytecode verification

linguaggio java

<< … un linguaggio semplice e familiare, orientato agli oggetti, robusto, sicuro, ***architettura neutrale***, portabile, ad alte prestazioni, interpretato, multithreaded e dinamico … >>

[J. Gosling e H. McGilton, "The Java Language Environment. A White Paper." Maggio 1996]

- filosofia di base : "le applicazioni devono potere essere eseguite ovunque sulla rete senza conoscere a priori la piattaforma hardware o software"
- Java adotta un codice binario indipendente dalla piattaforma hardware e dal sistema operativo.
  - Il compilatore Java non genera codice macchina ma "***bytecode***" indipendente dalla piattaforma che viene interpretato su ogni specifica architettura

linguaggio java

<< … un linguaggio semplice e familiare, orientato agli oggetti, robusto, sicuro, architettura neutrale, ***portabile***, ad alte prestazioni, interpretato, multithreaded e dinamico … >>

[J. Gosling e H. McGilton, "The Java Language Environment. A White Paper." Maggio 1996]

- esplicita considerazione di meccanismi che mitighino i rischi legati ad aspetti di portabilità :
  - interpretazione di operazioni aritmetiche (es. in C e C++, il codice sorgente può produrre risultati differenti a seconda della piattaforma hardware a causa di come vengono implementate le operazioni aritmetiche)
  - Java garantisce gli stessi risultati indipendentemente dalle piattaforme hardware

linguaggio java

<< … un linguaggio semplice e familiare, orientato agli oggetti, robusto, sicuro, architettura neutrale, portabile, ad ***alte prestazioni***, interpretato, multithreaded e dinamico … >>

[J. Gosling e H. McGilton, "The Java Language Environment. A White Paper." Maggio 1996]

- il "***bytecode***" può essere tradotto durante la fase di runtime nel linguaggio macchina del processore che viene utilizzato
- il processo di generazione del linguaggio macchina generalmente produce un codice con un buon livello di ottimizzazione, con l'allocazione automatica dei registri e un livello di prestazioni comparabili a quelle ottenute con programmi nativi in C o C++

- … in qualche caso però questo aspetto può essere critico!!!

linguaggio java

<< … un linguaggio semplice e familiare, orientato agli oggetti, robusto, sicuro, architettura neutrale, portabile, ad alte prestazioni, ***interpretato***, multithreaded e dinamico … >>

[J. Gosling e H. McGilton, "The Java Language Environment. A White Paper." Maggio 1996]

- Il "***bytecode***" prodotto dal compilatore Java questo può essere eseguito su qualsiasi macchina che abbia un interprete Java o piattaforma Java-enabled (e.g. browser).
- contribuisce alla neutralità dell'architettura
- consente di avere programmi "compilati" relativamente leggeri perchè non in codice macchina ma in bytecode da interpretare

linguaggio java

<< … un linguaggio semplice e familiare, orientato agli oggetti, robusto, sicuro, architettura neutrale, portabile, ad alte prestazioni, interpretato, ***multithreaded*** e dinamico … >>

[J. Gosling e H. McGilton, "The Java Language Environment. A White Paper." Maggio 1996]

- applicazioni concorrenti scritte in C o C++ delegano molti aspetti al sistema operativo (multi-processo VS multi-thread)
- supporto nativo di concetti di concorrenza
- il concetto di Thread di Java supporta una serie completa di metodi per avviare, eseguire, interrompere e analizzare i processi (thread)
- il supporto dei processi integrato nel linguaggio lo rende più semplice e affidabile da utilizzare

linguaggio java

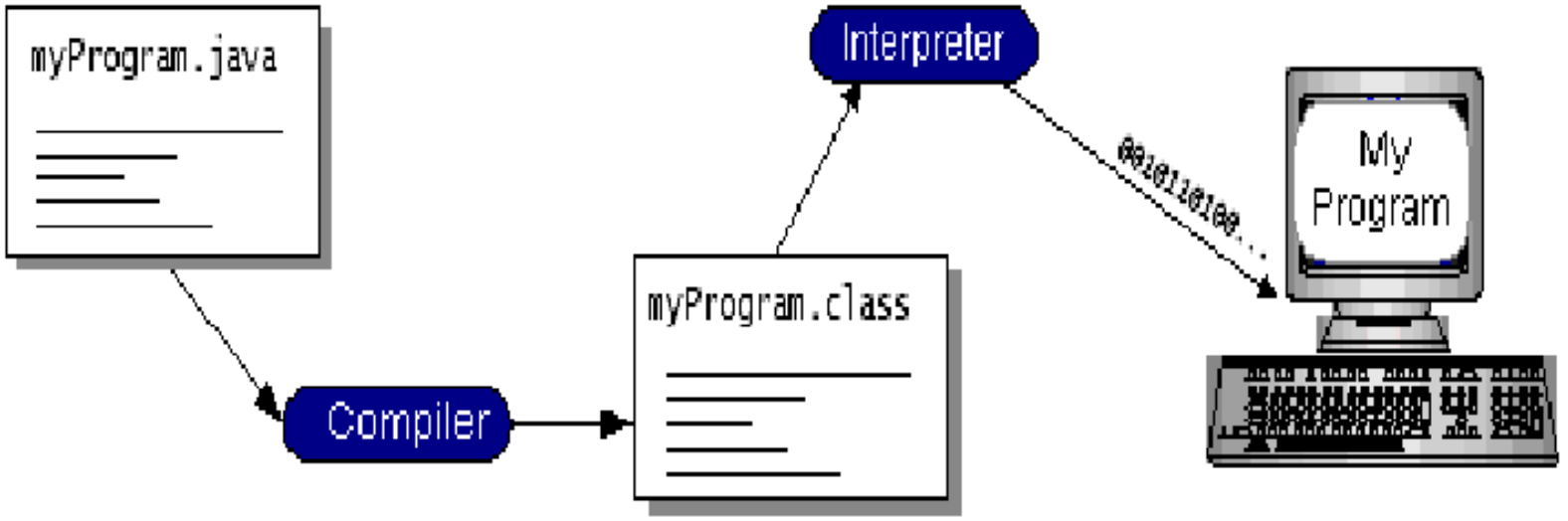
<< … un linguaggio semplice e familiare, orientato agli oggetti, robusto, sicuro, architettura neutrale, portabile, ad alte prestazioni, interpretato, multithreaded e ***dinamico*** … >>

[J. Gosling e H. McGilton, "The Java Language Environment. A White Paper." Maggio 1996]

- C++ soffre del "***constant recompilation problem***"
  - le funzioni/attributi/costanti sono riferiti (*linking*) con specifici indirizzi nei file compilati, e non da nomi simbolici
  - ricompilare anche le classi che riferiscono una classe modificata
- Il compilatore Java usa riferimenti simbolici e non riferimenti numerici
- l'interprete Java risolve i nomi quando le classi sono linkate
  - tuttavia Java (come molti linguaggi O.O.) non riesce a risolvere in modo ottimale il "***fragile base-class problem***"

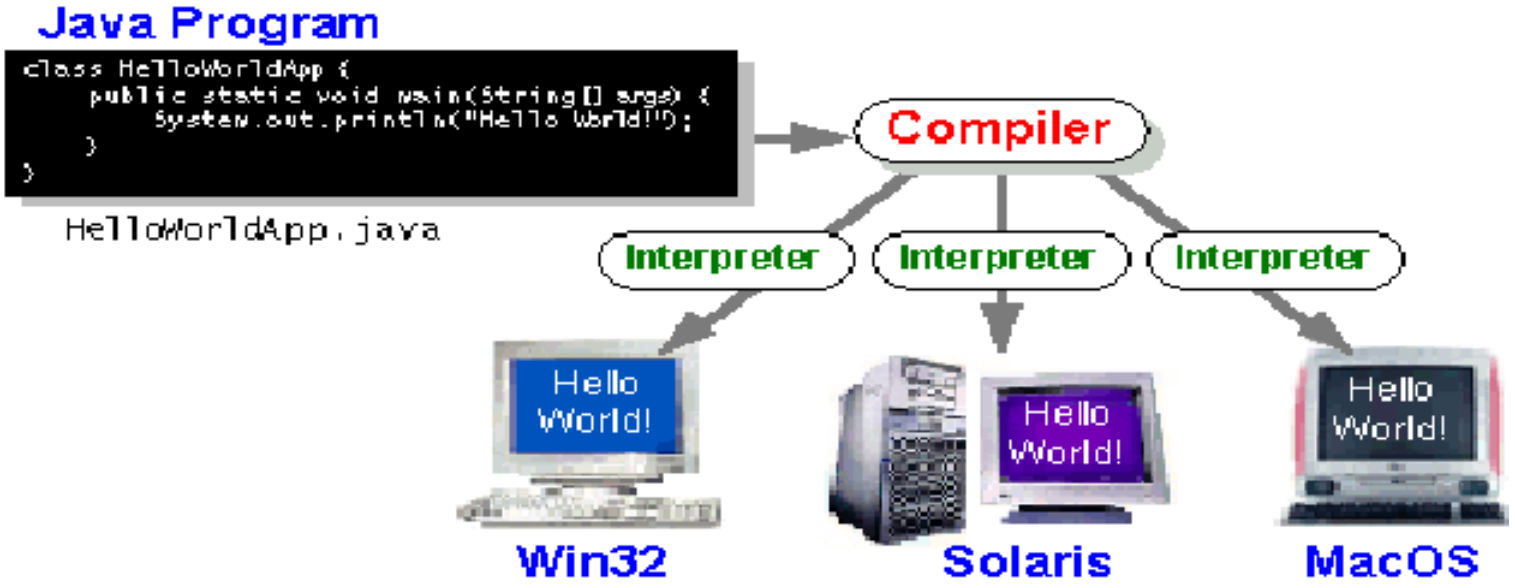
tecnologia Java – 1

- il sorgente viene compilato in un linguaggio intermedio indipendente dalla piattaforma detto "***bytecode***"
- un interprete traduce il "***bytecode***" in istruzioni macchina per la specifica architettura hardware



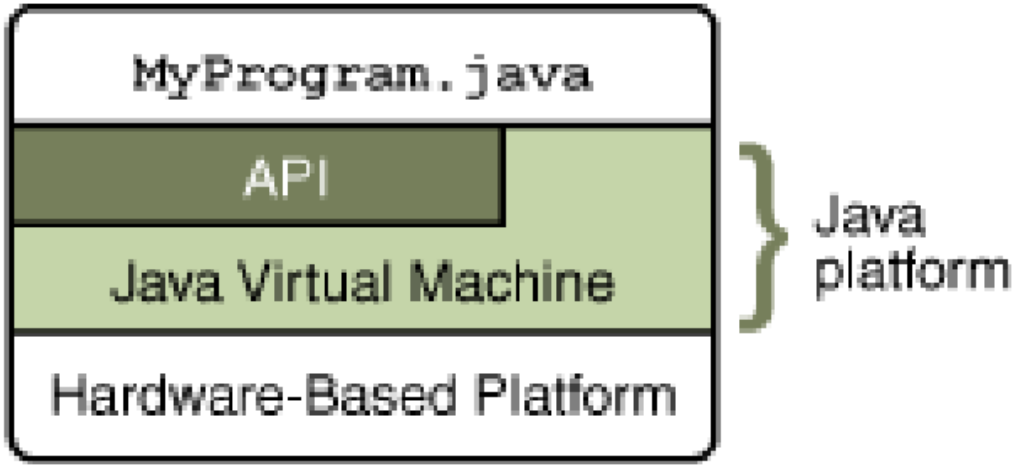
tecnologia Java – 2

- bytecode sono istruzioni macchina per la Java Virtual Machine (JVM)
- bytecode rende possibile "*write once run anywhere*"
- è possibile eseguire lo stesso codice su Windows, Solaris, Linux, iMac, qualsiasi dispositivo che ha una Java VM

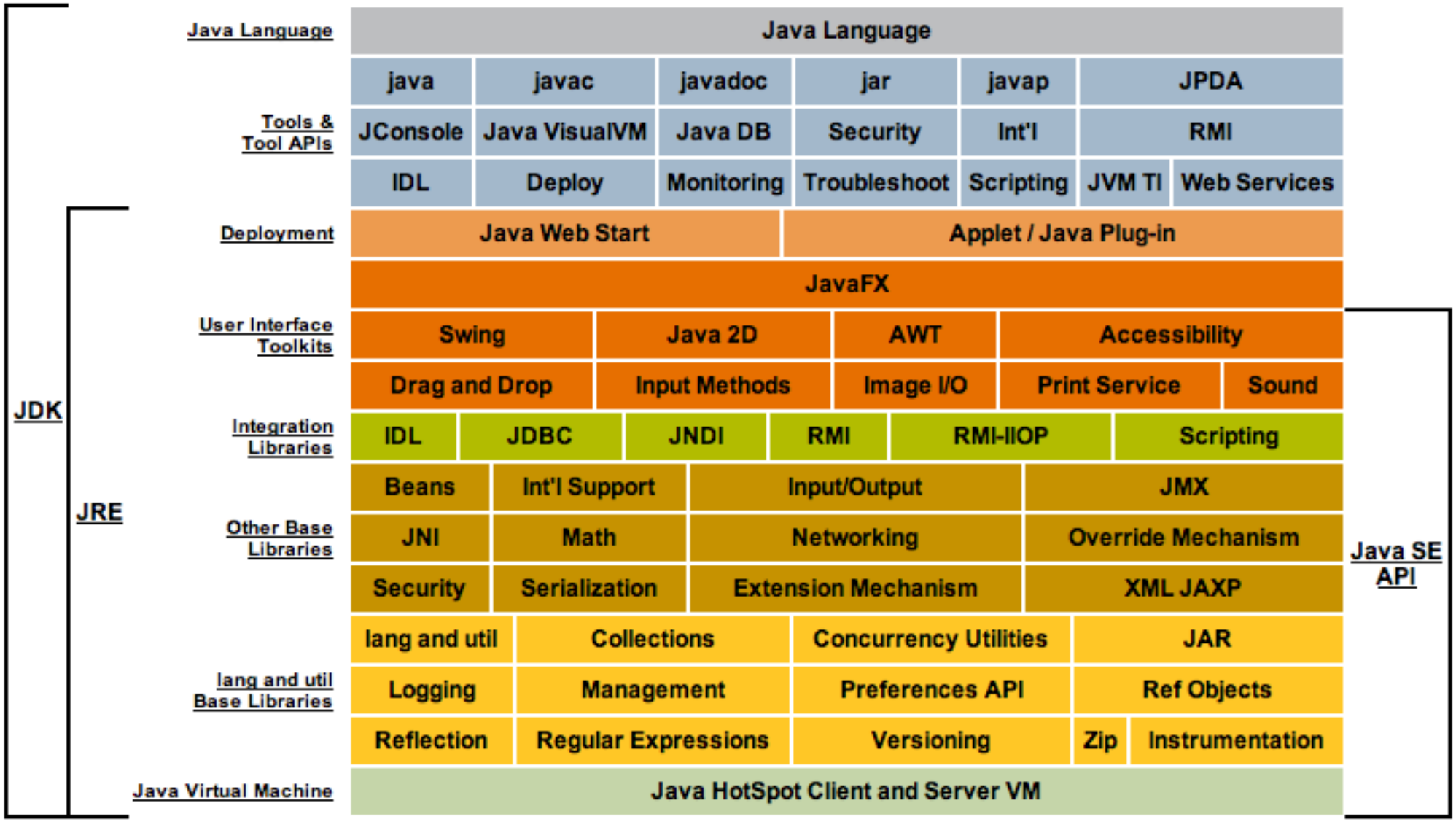


piattaforma – 1

- ambiente hardware e software all'interno del quale vengono eseguiti i programmi
  - Java Virtual Machine (JVM)
  - Java Application Programming Interface (Java API)
    - collezione di componenti software nativi concepiti supportare la risuluzione di aspetti comuni:
      - I/O, thread, networking, math, etc etc
    - raggruppata in librerie di classi ed interfacce; tali librerie sono normalmente chiamate packages



piattaforma – 2



JVM – 1

- è uno dei punti fondamentali della piattaforma Java
- è il componente che garantisce l'indipendenza dal HW e dal sistema operativo
- implementazioni "standard" della VM sono sviluppate da Oracle
  - incluse nei JAVA SDK e JAVA Runtime Environment
  - emulano la VM su Win, Solaris, Linux e Mac OS
- esistono implementazioni/distribuzioni di VM differenti da quella "standard"
  - IBM, BEA Rockit, OpenJDK, ...
- può essere implementata direttamente su HW

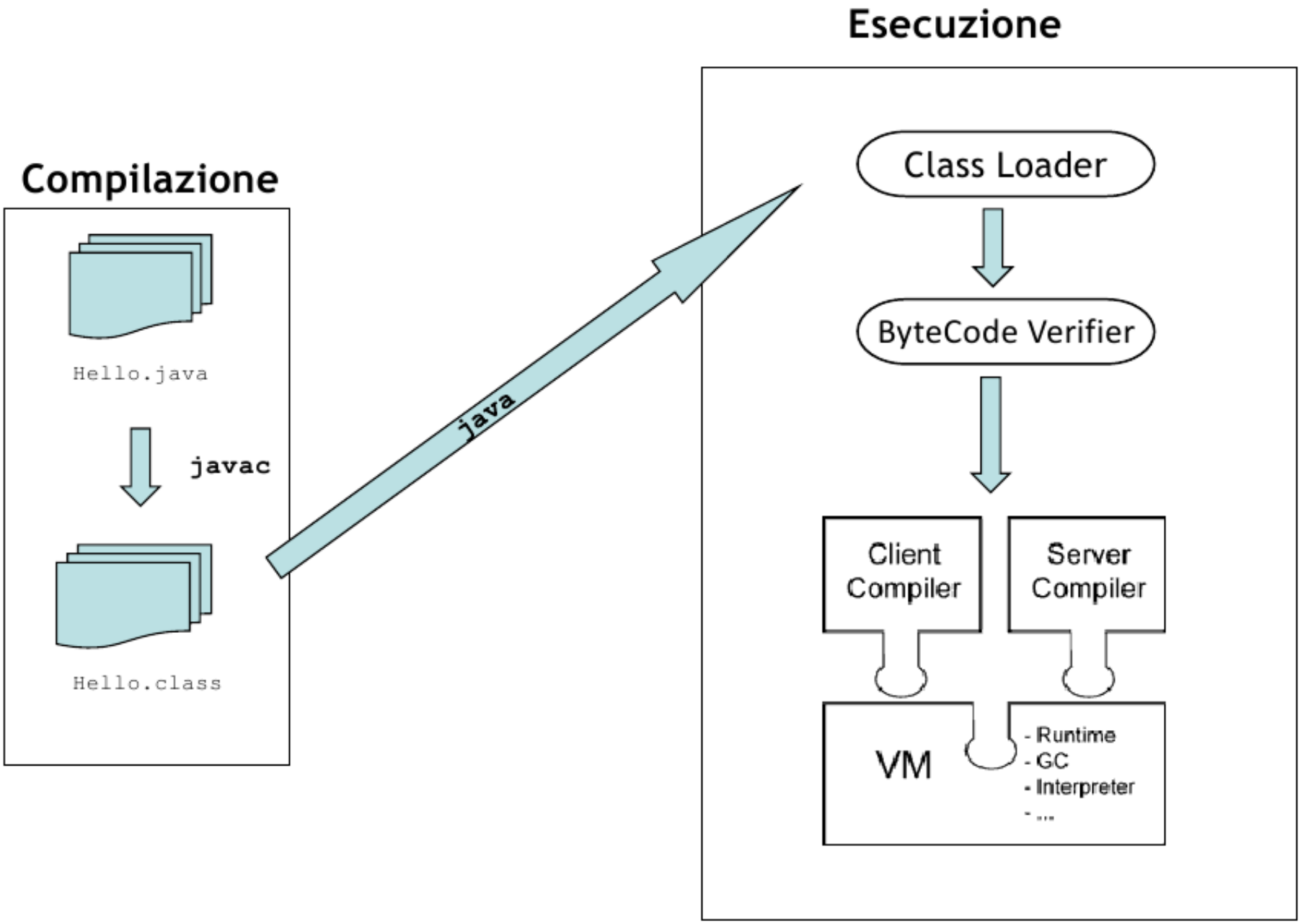
JVM – 2

- rappresenta una macchina astratta principalmente definita da
  - un insieme di istruzioni per una ipotetica CPU
  - un insieme di registri
  - stack
  - heap
  - formato di file delle classi

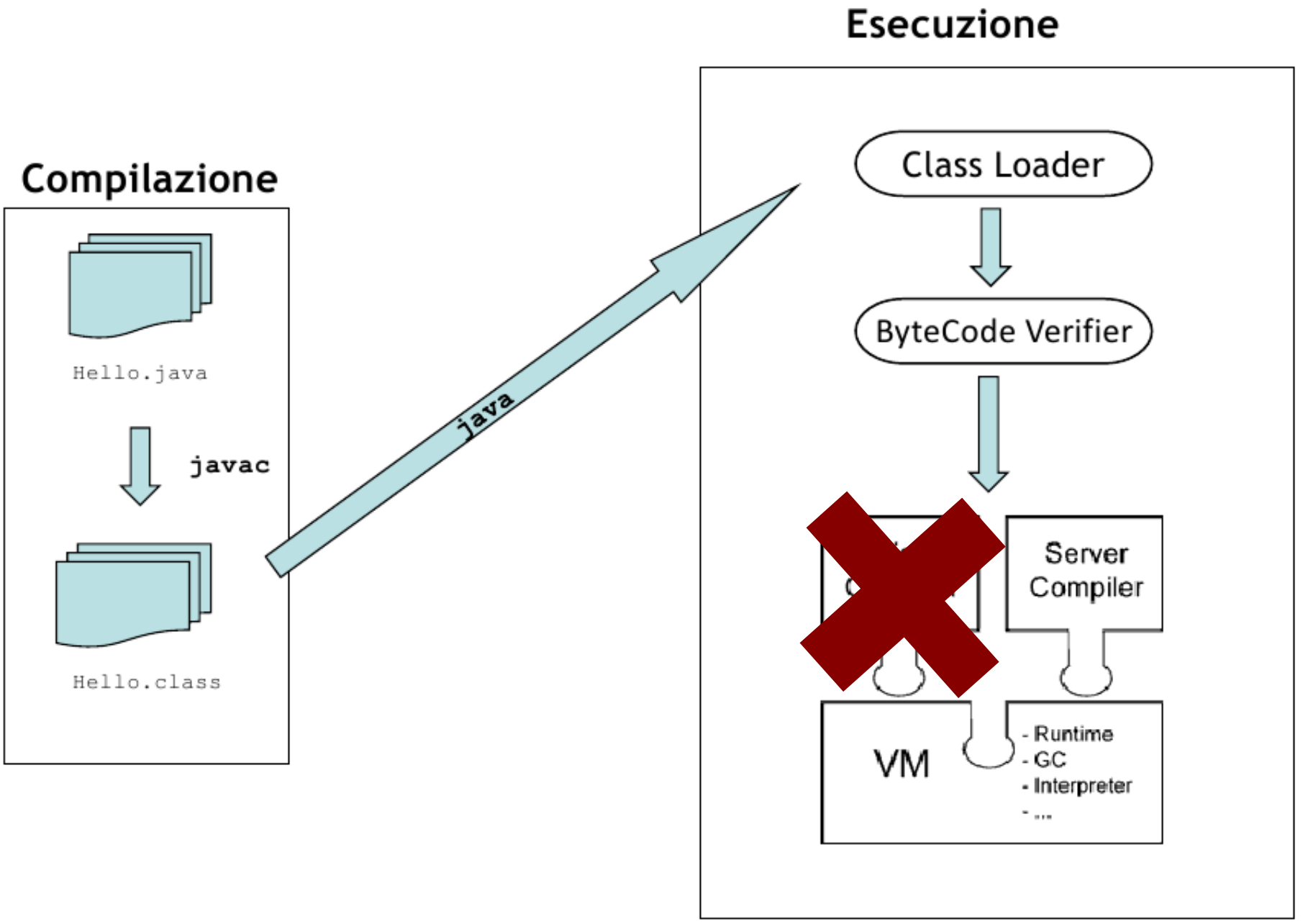
JVM – 3

- in Java (quasi) ogni cosa è un oggetto
- la creazione di oggetti viene effettuata dal programmatore utilizzando la parola chiave "***new***"
- la deallocazione viene gestita automaticamente dalla piattaforma
  - differenza importante con altri linguaggi, tipo C++
  - Garbage Collector : thread a bassa priorità che ha il compito di eliminare gli oggetti che non hanno più alcun riferimento
  - questo aspetto può avere impatti anche notevoli sulle performance delle applicazioni

JVM – 4


Compilazione
Hello.java
javac
Hello.class
java
Esecuzione
Class Loader
ByteCode Verifier
Client Compiler
Server Compiler
VM
- Runtime
- GC
- Interpreter
- ...

JVM – 4


Compilazione
Hello.java
javac
Hello.class
java
Esecuzione
Class Loader
ByteCode Verifier
Server
Compiler
VM
- Runtime
- GC
- Interpreter
- ...

struttura programma Java

- un programma Java segue la filosofia "***object oriented***"
  - è organizzato composto da (una o) più classi che interagiscono
- una classe contiene uno o più "***metodi***"
  - solo per il momento accenniamo all'*erronea* uguaglianza : "***metodo***" = "***funzione***"
- un metodo contiene le istruzioni

- il punto di accesso/lancio di un'applicazione Java è rappresentato dalla classe che contenente il metodo "***main***" che definisce, appunto, ciò che va fatto all'avvio.

```
public static void main(String[] args) {
}
```

il mio primo “`Hello World`” in Java

- scaricare il J2SE SDK (1.9.* … anche versioni più dovrebbero andare bene per gli scopi del corso)
  - http://java.com/en/download/manual.jsp
  - http://openjdk.java.net/install/index.html
- creare un file sorgente contenente l’applicazione che stampa “`Hello World`”
- compilare il file per generare l’eseguibile (bytecode)
- eseguire il bytecode con l’interprete java

- opzionale
  - scaricare la documentazione da java.sun.com (API & tutorial)
  - scaricare un IDE (Eclipse, NetBeans, IntelliJ IDEA, ...)

“HelloWorld”

```
/**
  * The HelloWorld class implements
  * an application that displays "Hello World!"
  * to the standard output.
*/

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

“HelloWorld”

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

NOTARE: non sono chiamate di funzioni ma “componenti di sistema” su cui vengono invocate operazioni ad essi pertinenti

NOTAZIONE: invia il messaggio `println(“Hello World!”)` all’attributo `out` (pubblico, e statico) che è un parte della classe predefinita `System`

struttura di una classe – 1

```
<modificatore> class <nome della classe> [extends Tipo]
                                        [implements ListaTipi] {
    [<dichiarazione di attributi>]
    [<dichiarazione dei costruttori>]
    [<dichiarazione dei metodi>]

}
```

Corpo della Classe

dichiarazione di metodi

```
<modificatore> <tipo ritorno> <identificatore> ([parametri]){
        [<istruzioni>]
}
```

struttura di una classe – esempio

```
public class ThisIsAnotherClass {
  public static void stampa(int x, int y) {
    System.out.println("x=" + x + ", y=" + y);
  }
  public static int somma(int x, int y) {
    return x + y;
  }
}
```

modificatore `static`

- indica una variabile o un metodo è “di classe”
- **SOLO PER IL MOMENTO** considerare che:
  - si può accede alla variabile/metodo mediante il nome della classe, senza ricorrere a nessun oggetto che istanzia la classe:

```
ThisIsAnotherClass.stampa(1,2);
```

  - senza questo modificatore c'è bisogno di un una istanza della classe per accedere alla variabile/metodo:

```
ThisIsAnotherClass t = new ThisIsAnotherClass();
t.stampa(1,2);
```

- il discorso sarà ripreso ed affrontato in dettaglio nelle lezioni successive

tipi generici

- come C++ e altri linguaggi C-like, Java supporta tipi generici
- i tipi generici sono tipi di dato parametrici
  - tipi di dato che possono essere usati sempre nello stesso modo, ma in contesti differenti, semplicemente variando il tipo del parametro associato
- una classe è generica se le operazioni che offre possono essere applicate ad un tipo di dato generico

```
ArrayList<Auto> garage = new ArrayList<Auto>();
for( Auto a : garage ){
  …
}
```

parametri attuali e formali

- in Java il passaggio di parametri a metodi è SEMPRE per valore
- NOTA:
  - le variabili su oggetti sono in realtà dei puntatori
  - nell'uso di un oggetto come parametro attuale, le modifiche sul parametro formale provocano effetti che "coincidono" con il passaggio di parametri per riferimento.

`vedi codice di esempio allegato alla lezione`

commenti – 1

- sono spesso detti documentazione in-line
- sono inclusi per documentare lo scopo e le funzionalità del della classe o particolari scelte implementative
- non influenzano il funzionamento del codice
- vengono trascurati dal compilatore
- possono avere tre forme:

```
1. // commenti fino alla fine della riga
2. /* commento che
      può stare su più righe */
3. /** commento che genera documentazione
      */
```

commenti – 2

- non si possono annidare
- è possibile utilizzare
  - `/*` e `*/` all'interno del marcatore `//`
  - `//` all'interno dei marcatori `/*` e `/**`
- all'interno dei marcatori `/*` e `/**`
  - e possibile utilizzare `/*`
  - non è possibile utilizzare `*/`
- esempio

```
/* this comment /* // /** ends here: */
```

convenzioni di stile – 1

- Java incentiva l'uso di alcune convenzioni per la denominazione di package, classi, interfacce, costanti, ...
- Java prevede anche delle convenzioni per l’indentazione del codice
- queste convenzioni di stile :
  - non sono obbligatorie
  - aiutano a rendere il codice più leggibile e manutenibile
  - mitigano la generazione di alcuni tipi di conflitti dei nomi
- in generale si raccomanda di utilizzare le convenzioni specificate
- Documento ufficiale:
  - Code Conventions for the Java Programming Language
    Revision April 20, 1999
  - http://www.oracle.com/technetwork/java/javase/documentation/

convenzioni di stile – 2

- nomi dei package
  - prefisso che identifica il nome del dominio internet (utilizzato al contrario) della società/ente che sviluppa l'applicazione
  - far seguire il precedente prefisso con il nome dell'applicazione o del progetto, o del dipartimento, … etc
- esempio
  - `java.util`
  - `it.uniroma2.dicii.ispw.myfirsttest`
  - `org.apache.struts`
  - `org.apache.xml.axis`
  - `org.omg.CORBA`

convenzioni di stile – 3

- nomi classi ed “interfacce”
  - si utilizzano tipicamente nomi o frasi non eccessivamente lunghe
  - se è una frase composta nomi singoli scritti con l'iniziale in maiuscolo
  - nome inizia sempre in maiuscolo
  - tipicamente le “interfacce” sono aggettivate
- esempi
  - `System`
  - `ClassLoader`
  - `SecurityManager`
  - `Thread`
  - `BufferedInputStream`
  - `Runnable`
  - `ThisIsAnotherClass`

convenzioni di stile – 4

- nomi dei metodi
  - si utilizzano tipicamente verbi o frasi
  - nome inizia sempre in minuscolo
  - se è una frase composta nomi singoli scritti con l'iniziale in maiuscolo
  - se restituisce un booleano tipicamente rispetto ad una condizione `V` si utilizza `isV`
- esempi
  - `getPriority`
  - `setPriority`
  - `length`
  - `toString`
  - `isInterrupted` nella classe `Thread`

convenzioni di stile – 5

- nomi degli attributi
  - generalmente identificano nomi o frasi oppure abbreviazioni
  - se non hanno il modificatore `final` iniziano con lettera minuscola
  - se è una frase composta nomi singoli scritti con l'iniziale in maiuscolo
  - se hanno il modificatore `final` sono scritti tutto in maiuscolo e se composti singole parole separati da “_”
- esempi
  - `buf`, `pos`, `count` in `java.io.ByteArrayInputStream`
  - `out` in `System`
  - `MIN_VALUE`, `MAX_VALUE`

convenzioni di stile – 6

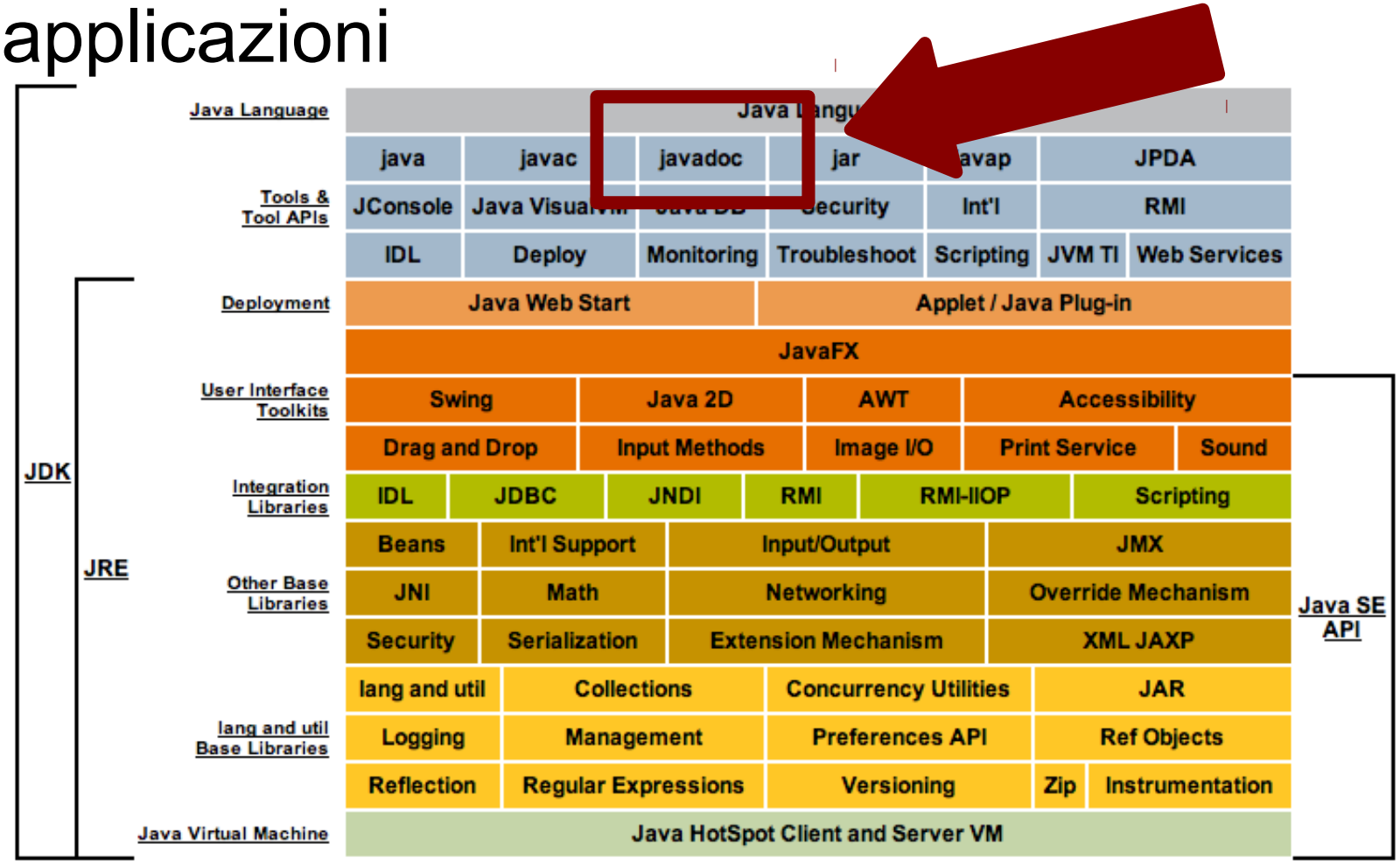
- variabili locali e nomi parametri di metodi
  - generalmente sono nomi brevi
  - possono identificare degli acronimi
    - `cp` variabile locale per classe `ColoredPoint`
  - abbreviazioni
    - `buf` per indicare un buffer
  - termini mnemonici
    - `millis` **`e`** `nano` come parametri del metodo `sleep` della classe `Thread`

convenzioni di stile – 7

- … … …
- nel caso di variabili locali si è soliti utilizzare le seguenti convenzioni
  - `b` per i tipi `byte`
  - `c` per i tipi `char`
  - `d` per i tipi `double`
  - `e` per i tipi `Exception`
  - `f` per i tipi `float`
  - `i,j,k` per i tipi `int`
  - `l` per i tipi `long`
  - `o` per i tipi `Object`
  - `s` per i tipi `String`

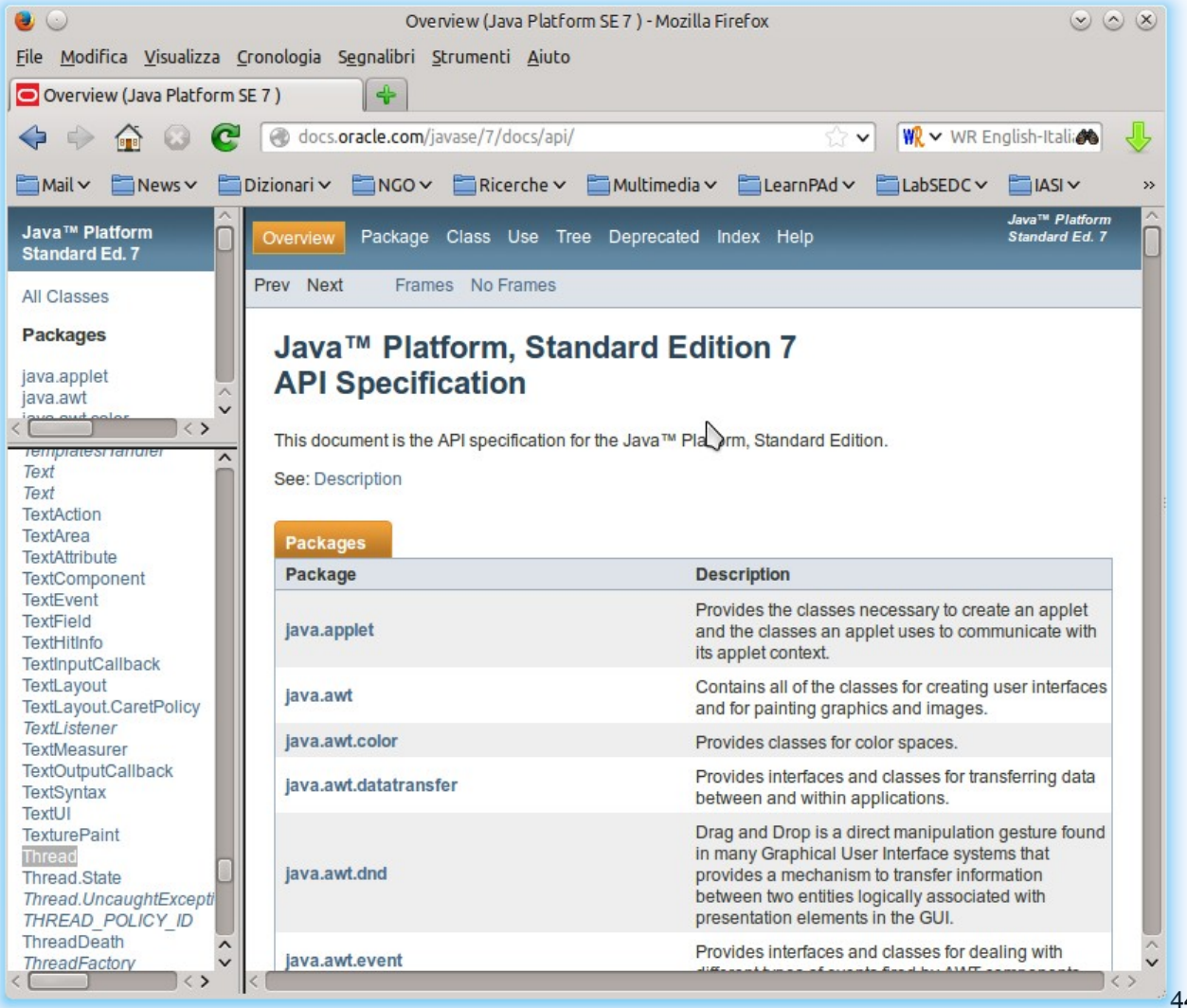
Java API documetation – 1

- la piattaforma di Java prevede meccanismi per a supporto della una documentazione di applicazioni



Java API documentation – 2

- le stesse distribuzioni Java usano questi tipi di meccanismi



Java API documentation – 3

- il comando `javadoc` prende in input sorgenti Java e genera automaticamente tale documentazione di API per le classi contenute
- `javadoc` riconosce il contenuto dei commenti formattati con "`/** **/`". Dentro tali delimitatori è possibile inserire direttamente codice HTML

```
/*************************************************
 *  Punti nel piano cartesiano                   *
 * @author Laurent Théry                         *
 * @version 1.0                                  *
 *************************************************/
public class Point {
  // campi privati
  private int x,y;
  // Costruttore completo
  Point (int x1, int y1) {
    x=x1;
    y=y1;
  }
  /**  test d'uguaglianza
    @param o l'altro oggetto
    @return true se i due oggetti sono gli stessi
  **/
  public boolean equals(Object o) {
    if (o instanceof Point)  {
      return (((Point)o).x == x) && (((Point)o).y == y);
    } else {
      return false;
    }
  }
}
```

Java API documentation – 4

Class Point

```
java.lang.Object
  |
  +--Point
```

public class **Point**
extends java.lang.Object

Punti nel piano cartesiano *

Method Summary

boolean	equals(java.lang.Object o) test d'uguaglianza

Methods inherited from class java.lang.Object

clone, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

equals

```
public boolean equals(java.lang.Object o)
```

test d'uguaglianze

Overrides:
equals in class java.lang.Object
Parameters:
o - l'altro oggetto
Returns:
true se i due oggetti sono gli stessi

credits

parte dei contenuti di queste slide sono stati elaborati a partire dalla presentazione di Davide Di Ruscio :

“Linguaggio e Piattaforma Java”