Linguaggio e Piattaforma Java

Davide Di Ruscio
Dipartimento di Informatica
Università degli Studi dell'Aquila

davide.diruscio@univaq.it

Sommario

- Tecnologia Java: Linguaggio e Piattaforma
- Java Virtual Machine
- Storia
- Struttura di un programma Java
- First Cup of Java
- Commenti e Spazi Bianchi
- JavaDoc
- Coding convention

Tecnologia Java (1)

» Linguaggio

– Sun Microsystems definisce Java come
<<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.” Ottobre 1995]

Tecnologia Java (1)

» Linguaggio

– Sun Microsystems definisce Java come
*<<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." Ottobre 1995]

- La rimozione di molte caratteristiche di dubbia utilità dai suoi "progenitori" C e C++, mantiene Java relativamente piccolo e permette ai programmatori la produzione di applicazioni più affidabili

- I programmatori che utilizzano il C, Objective C, C++, Eiffel, Ada e linguaggi affini acquisteranno familiarità con Java in breve tempo, dell'ordine di qualche settimana

Tecnologia Java (1)

» Linguaggio

– Sun Microsystems definisce Java come
*<<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.” Ottobre 1995]

- Java estrae i migliori concetti e caratteristiche dai precedenti linguaggi orientati agli oggetti, come Eiffel, SmallTalk, Objective C e C++
- Con l’eccezione dei suoi tipi di dato primitivi, ogni cosa in Java è un oggetto ed anche i tipi di dato primitivi possono essere incapsulati all’interno di oggetti se necessario
- Java supporta le quattro caratteristiche chiave di un linguaggio orientato agli oggetti i*ncapsulamento*, *polimorfismo*, *ereditarietà e binding dinamico*

Tecnologia Java (1)

» Linguaggio

– Sun Microsystems definisce Java come
<<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." Ottobre 1995]

- Un software si definisce robusto se si comporta "bene" in condizioni di lavoro non previste dallo sviluppatore
  - per *bene* si intende non va in crash, non sbaglia calcoli, etc.
  - per *condizioni non previste* si intende 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*)

Tecnologia Java (1)

» Linguaggio

– Sun Microsystems definisce Java come
<<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." Ottobre 1995]

- Il compilatore e il sistema di runtime implementano molti strati di difesa contro del codice potenzialmente non corretto:
  - la gestione della memoria non è delegata al compilatore (come in C o C++), ma rinviata al run-time;
  - i puntatori non esistono (accesso diretto impossibile);
  - tutto deve essere esplicito (ad esempio il casting), niente avviene "nell'ombra";
  - 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;
  - Java non si fida del codice proveniente dalla rete, per il quale viene eseguito la bytecode verification

Tecnologia Java (1)

» Linguaggio

– Sun Microsystems definisce Java come
*<<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." Ottobre 1995]

- 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*: un codice ad alto livello, indipendente dalla macchina per un computer astratto che viene implementato dall'interprete Java

Tecnologia Java (1)

» Linguaggio

– Sun Microsystems definisce Java come
*<<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." Ottobre 1995]

- Un aspetto relativo alla portabilità è come l'hardware interpreta le 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)
- In Java questo è stato semplificato e si hanno sempre gli stessi risultati indipendentemente dalle piattaforme hardware

Tecnologia Java (1)

» Linguaggio

– Sun Microsystems definisce Java come
<<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." Ottobre 1995]

- Per applicazioni che richiedono elaborazioni pesanti e quindi prestazioni maggiori il formato bytecode può essere tradotto durante la fase di runtime nel linguaggio macchina del processore che viene utilizzato
- Il processo di generazione del linguaggio macchina è generalmente semplice e produce un codice con un buon livello di ottimizzazione, con l'allocazione automatica dei registri e un livello di prestazioni quasi uguale a quello ottenuto con linguaggi C o C++ nativi

Tecnologia Java (1)

» Linguaggio

– Sun Microsystems definisce Java come
<<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.” Ottobre 1995]

- Il bytecode prodotto dal compilatore Java questo può essere eseguito su qualsiasi macchina che ha un interprete Java o un browser Java-enabled.
- Questo permette al codice Java di essere scritto indipendentemente dalla piattaforma dell’utente

Tecnologia Java (1)

» Linguaggio

– Sun Microsystems definisce Java come
<<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." Ottobre 1995]

- Scrivere in C o C++ applicazioni che eseguono più processi simultaneamente è molto difficile: non si è mai totalmente sicuri di aver impostato oppoerunamente i blocchi necessari o di averli liberati
- La classe *Thread* di Java supporta una serie completa di metodi per avviare, eseguire, interrompere e analizzare i processi
- Il supporto dei processi integrato nel linguaggio lo rende più semplice e affidabile da utilizzare

Tecnologia Java (1)

» Linguaggio

– Sun Microsystems definisce Java come
<<un linguaggio semplice e familiare, orientato agli oggetti, robusto, sicuro, architettura neutrale, portabile, ad alte prestazioni, interpretato, multithreaded e ***dinamico****>>*

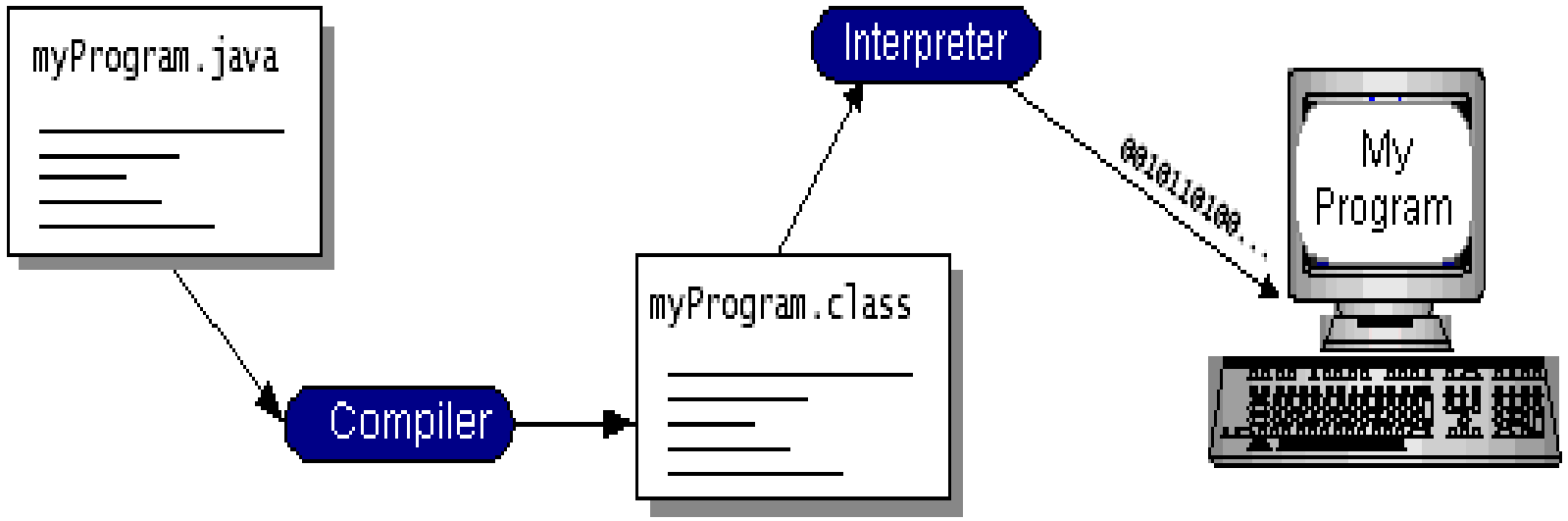
[J. Gosling e H. McGilton, "The Java Language Environment. A White Paper." Ottobre 1995]

- Il C++ soffre del *fragile superclass problem*, dovuto alla mancanza di uno standard riguardo al layout degli oggetti, al passaggio dei parametri e così via. Ogni volta che si aggiunge un metodo o una variabile d'istanza ad una classe, qualsiasi classe che si riferisce a quella modificata dovrà essere ricompilata (*constant recompilation problem*)
- Il compilatore Java non trasforma i riferimenti in valori numerici, ma li converte in riferimenti simbolici. L'interprete Java esegue la risoluzione finale dei nomi quando le classi sono linkate

Tecnologia Java (2)

» Sorgente viene compilato in un linguaggio intermedio indipendente dalla piattaforma detto *bytecode*

» Interprete traduce il bytecode in istruzioni macchina del computer

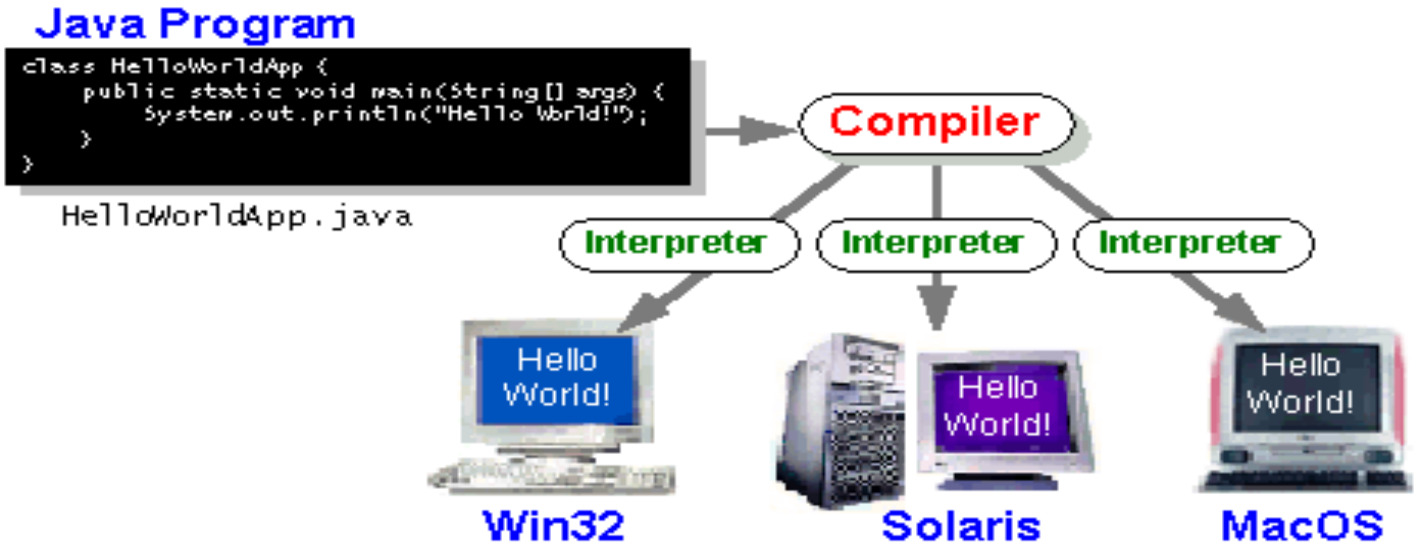


Tecnologia Java (3)

» Bytecode sono istruzioni macchina per la Java Virtual Machine (JVM)

» Bytecode rende possibile "*write once run anywhere*"

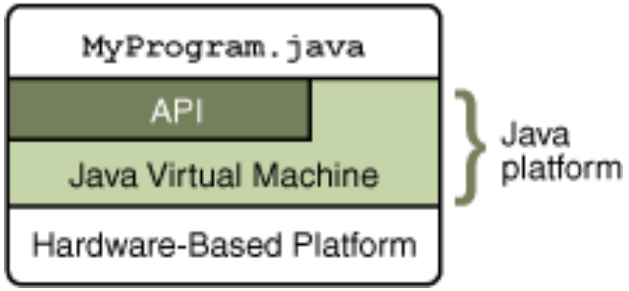
» E' possibile eseguire lo stesso codice su Windows, Solaris, Linux, iMac, qualsiasi dispositivo che ha una Java VM



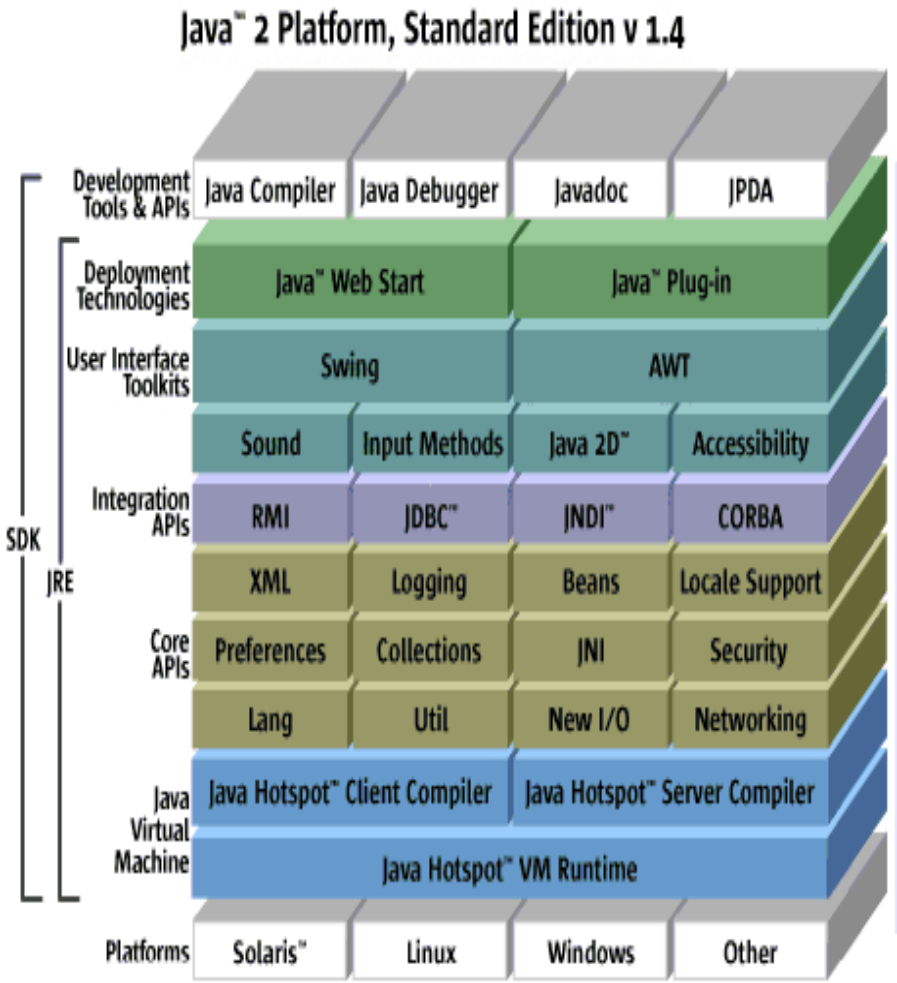
Tecnologia Java (4)

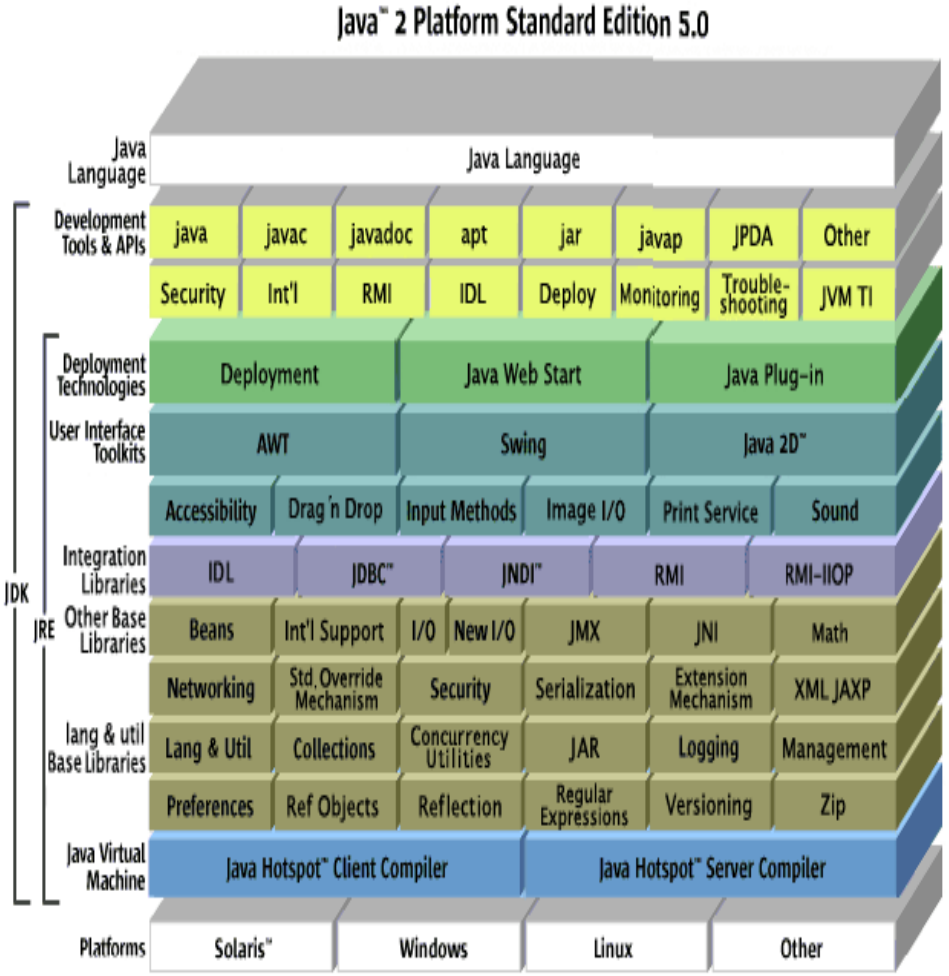
» Piattaforma

- Ambiente hardware e software all'interno del quale vengono eseguiti i programmi
- E' composta dalla
  - Java Virtual Machine (JVM)
  - Java Application Programming Interface (Java API)
    - Collezione di componenti software pronti per essere utilizzati concepiti per diversi scopi
    - Raggruppata in librerie di classi ed interfacce; tali librerie sono normalmente chiamate packages

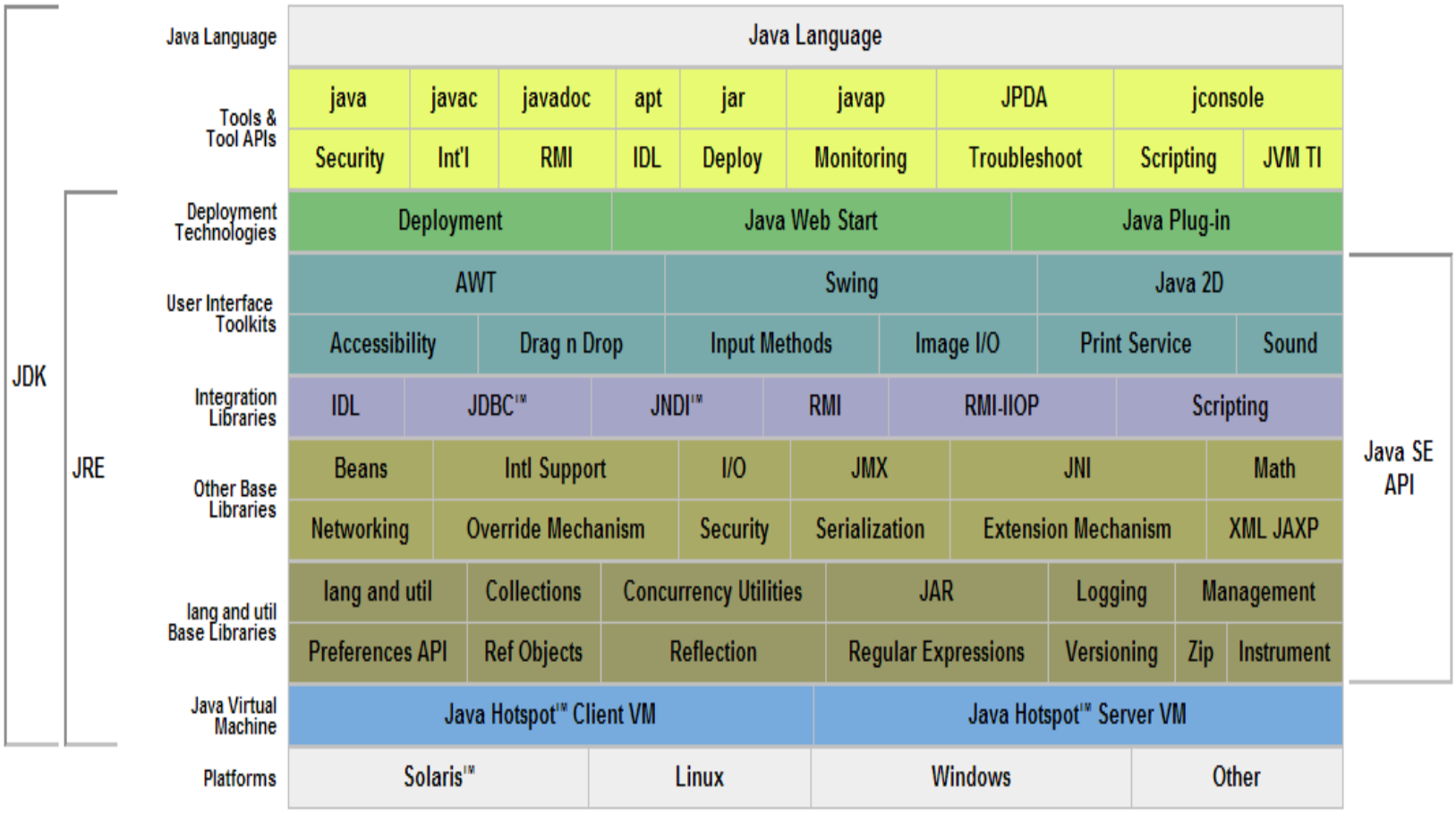


Tecnologia Java (5)





Tecnologia Java (6): Java 6.0



Storia (1)

» Nascita ’91 Patrick Naughton e James Gosling 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 progetto First Person (i.e. 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 (2)

- Inizi ’96 SUN rilasciò la prima versione di Java 1.0.2
- Dopo poco 1.1 che introduceva notevoli migliorie
- 1998 alla conferenza JavaOne rilasciarono versione 1.2 ovvero Java 2 Standard Edition Software Development Kit Version 1.2
- …
- 1.3
- 1.4
- JavaOne 2004 presentata 1.5 ovvero Java 5.0

Evoluzione del linguaggio

Vers.	Nuove Funzionalità del linguaggio	Numero classi e interfacce
1.0	Linguaggio di partenza	211
1.1	Classi interne	477
1.2	Nessuna	1524
1.3	Nessuna	1840
1.4	Asserzioni	2723
5.0	Classi generiche, “for each”, vararg, autoboxing, metadati, enumerazioni, importazione statica	3270
6.0	Aggiunte librerie Web services	≈7000

Java Virtual Machine (1)

- Costituisce la pietra angolare della piattaforma Java
- E' il componente responsabile dell'indipendenza dall'Hardware e dal sistema operativo
- Implementazioni correnti della VM della SUN (facenti parti del JAVA SDK e JAVA Runtime Environment) **emulano** la VM su Win32, Solaris, Linux e Mac OS
- Esistono altre implementazioni della VM come ad esempio IBM, BEA Rockit, ….
- Può essere implementata direttamente su HW

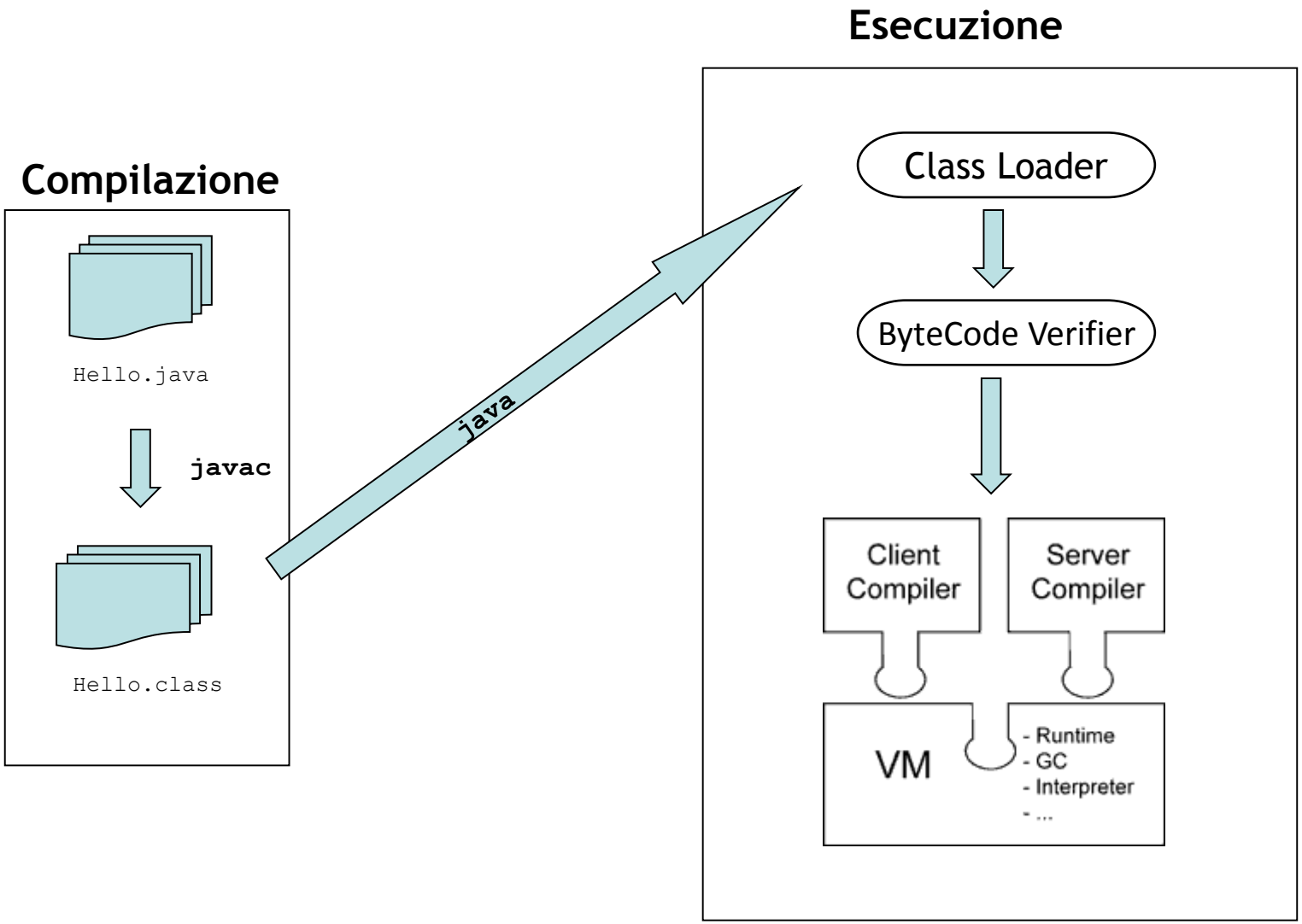
Java Virtual Machine (2)

- E’ fondamentalmente un computer astratto che ha un
  - Insieme di istruzioni per una ipotetica CPU
  - Insieme di registri
  - Stack
  - Heap
  - Formato di file delle classi

Java Virtual Machine (3)

- In Java ogni cosa è un oggetto
- La creazione di tali oggetti viene effettuata dal programmatore utilizzando la parola chiave `new`
- A differenza di altri linguaggi (C++) la deallocazione viene fatta automaticamente tramite il cosiddetto *Garbage Collector*
  - Thread a bassa priorità che ha il compito di eliminare gli oggetti che non hanno più alcun riferimento
  - Ha un impatto notevole sulle performance dell'applicazione

Java Virtual Machine (4)

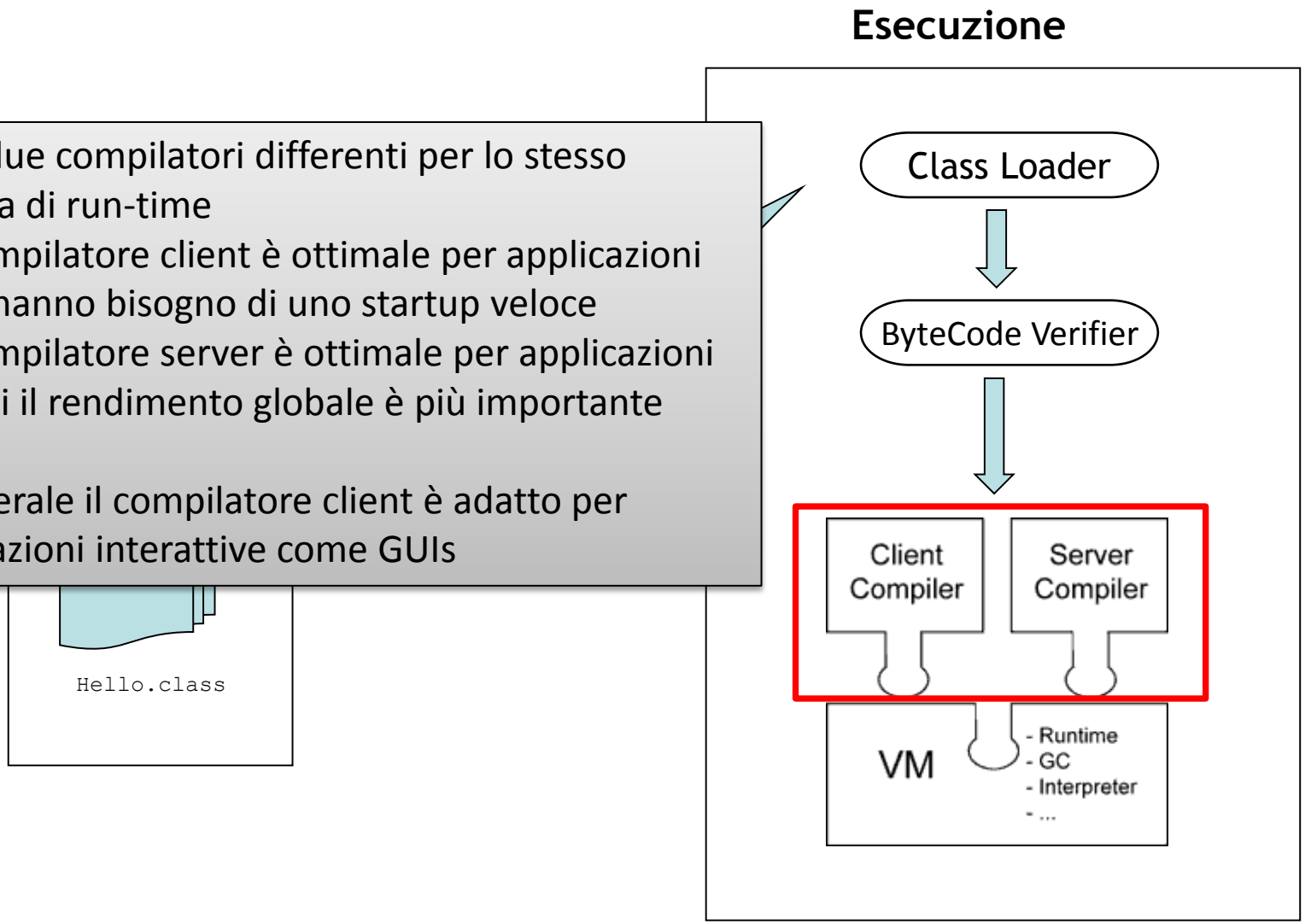


Java Virtual Machine (4)

Sono due compilatori differenti per lo stesso sistema di run-time

- Il compilatore client è ottimale per applicazioni che hanno bisogno di uno startup veloce
- Il compilatore server è ottimale per applicazioni in cui il rendimento globale è più importante

In generale il compilatore client è adatto per applicazioni interattive come GUIs



Struttura programma Java

- Ogni programma Java e composto da una o più classi
- Una classe contiene uno o più metodi
- Un metodo contiene le istruzioni
- Punto di partenza di un'applicazione Java classe contenente un metodo `main`
  - Sintassi

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

First Cup of Java (1)

» Scaricare il J2SE SDK (1.5.0 o 1.6.0)
  – http://java.sun.com/j2se/downloads/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, IDEA, …)

First Cup of Java (2)

Commento

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

Classe

Main

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

}
```

First Cup of Java (2)


```
/**
   * The HelloWorldApp cla
  that displays "Hello Wo
   * output.
*/

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

}
```

Non chiamate di funzioni con parametri che rappresentano i dati su cui operare ma "componenti" su cui vengono invocate operazioni a essi pertinenti

Notazione puntata:
Il messaggio println("Hello World!") è inviato all'oggetto out che è un membro (statico) della classe predefinita System

Struttura di una classe (1)

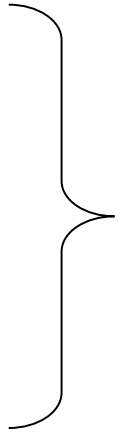

```
<modificatore> class <nome della classe>
                        [extends Tipo]
                        [implements ListaTipi] {


[<dichiarazione di attributi>]

[<dichiarazione dei costruttori>]

[<dichiarazione dei metodi>]


}
```



Corpo della classe

Struttura di una classe (2)

» **Per ora definizione di un metodo statico (o classe)**

» Metodo

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

   [<istruzioni>]

}
```

» Esempio

```
public class Test {

   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;
   }
}
```

Struttura di una classe (2)

- `static`
  - Indica una variabile di classe
  - Vi è un unico valore associato alla classe indipendente da quante istanze della classe sono create
  - Si accede alla variabile mediante il nome della classe (`Test.stampa(1,2)`)
- Senza `static`
  - Indica una variabile di istanza
  - Vi sono diversi valori per ogni oggetto della classe

Struttura di una classe (2)


```
class StaticTest{
      static int i = 47;
}

StaticTest st1 = new StaticTest();
StaticTest st2 = new StaticTest();
```

- Sia `st1.i` che `st2.i` hanno lo stesso valore 47 perché si riferiscono allo stesso segmento di memoria
- Si può fare riferimento alla variabile static anche direttamente il suo nome di classe:

```
StaticTest.i++;
```

A questo punto sia `st1.i` che `st2.i` assumeranno il valore 48

Struttura di una classe (2)

» Una logica simile vale per i metodi statici

```
class StaticFun{
      static void incr() { StaticTest.i++; }
}

StaticFun sf = new StaticFun();
sf.incr();
```

oppure

```
StaticFun.incr();
```

» Un uso importante di static per i metodi è quello che permette di chiamare il metodo senza creare un oggetto. Questo è essenziale per il metodo main() che è il punto di ingresso per eseguire un'applicazione

Commenti (1)

- Sono spesso detti documentazione in-line
- Sono inclusi per documentare lo scopo e le funzionalità del programma
- Non ne influenzano il funzionamento
- Vengono trascurati dal compilatore
- Possono avere tre forme

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

Commenti (2)

- Commenti non si possono annidare
- Si possono utilizzare `/*` e `*/` all'interno di `//`
- Si può utilizzare `//` all'interno dei commenti `/*` e `/**`
- All'interno di `/*` e `/**` e possibile utilizzare `/*` ma non `*/` contemporaneamente
- Esempio

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

Spazi bianchi

- Spazi, righe vuote e le tabulazioni sono considerati spazi bianchi
- Sono usati per separare le parole e i simboli di un programma
- Vengono ignorati dal compilatore
- Vengono utilizzati per formattare un programma Java migliorando la leggibilità

Code Convention (1)

» Java ha una convenzione per la definizione dei package, classi, interfacce, costanti, …

» Vi sono anche delle convenzioni per l'indentazione del codice

» Tali convenzioni aiutano a rendere il codice più leggibile e ad eliminare alcuni conflitti di nomi

» Si raccomanda di utilizzare le convenzioni adottate

» Link ufficiale

> http://java.sun.com/docs/codeconv/

Code Convention (2)

» Nomi dei package

> Tipicamente si utilizza un prefisso che identifica il nome del dominio internet (utilizzato al contrario) della società/ente che sviluppa l'applicazione

> Generalmente si fa seguire tale prefisso con il nome dell'applicazione o del progetto o del dipartimento, …

> Esempio

> com.sun.javaproject

> it.univaq.tlp.j2se

> org.apache.struts

> org.apache.xml.axis

> org.omg.CORBA

Code Convention (3)

» Nomi classi ed interfacce

> Si utilizzano tipicamente nomi o frasi non eccessivamente lunghe

> Se è una frase composta nomi singoli scritti in maiuscolo

> Nome inizia sempre in maiuscolo

> Tipicamente le interfacce sono aggettivate

> Esempi

> ClassLoader

> SecurityManager

> Thread

> BufferedInputStream

> Runnable

> Cloneable

Code Convention (4)

» Nomi dei metodi

> Si utilizzano tipicamente verbi o frasi

> Nome inizia sempre in minuscolo

> Se è una frase composta nomi singoli scritti in maiuscolo

> Esempi

> getPriority, setPriority

> length

> toString

> Se restituisce un booleano tipicamente rispetto ad una condizione V si utilizza isV

> isInterrupted classe Thread

Code Convention (5)

» Nomi dei campi

> Generalmente identificano nomi o frasi oppure abbreviazioni

> Se non sono final iniziano con lettera minuscola

> Se sono composti le altre parole lettera maiuscola

> Esempi

> buf, pos, count di java.io.ByteArrayInputStream

> out classe System

> Nel caso di costanti (ovvero final e campi nelle interfacce) i nomi sono scritti in maiuscolo e se composti singole parole separati da _

> MIN_VALUE, MAX_VALUE

> interface ProcessStates {

> int PS_RUNNING = 0;

> int PS_SUSPENDED = 1;

> }

Code Convention (6)

- Variabili locali e nomi parametri metodi
  - Generalmente sono nomi brevi
  - Possono identificare
    - Degli acronimi
      - cp variabile locale per classe ColoredPoint
    - Abbreviazioni
      - buf per un buffer di qualche tipo
    - Termini mnemonici
      - off e len oppure parametri read e write nei nomi dei metodi dell'interfaccia DataInput e DataOutput

Code Convention (7)

» .....

> Nel caso di variabili locali si possono usare 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 Documentation

» La distribuzione di Java contiene una documentazione della base di Java sotto forma HTML. Tale documentazione è sotto la directory `docs/api/`

Java Documentation

JDK 5 Documentation - Mozilla Firefox

File Modifica Visualizza Vai Segnalibri Strumenti ?

file:///C:/Java/jdk1.5.0/docs/index.html

Firefox Help Firefox Support Plug-in FAQ

JDK™ 5.0 Documentation

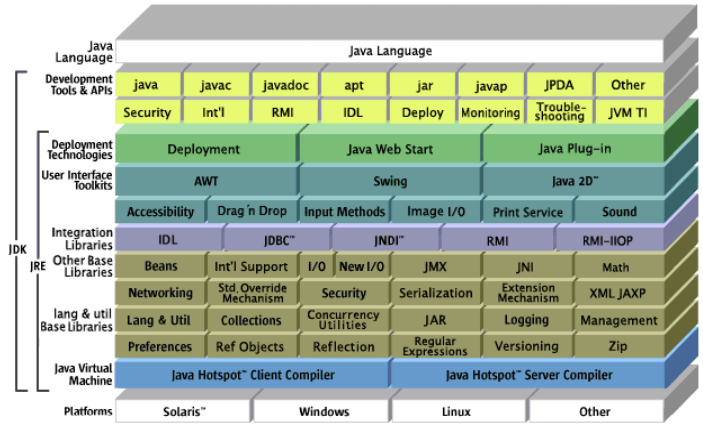
Download this Documentation

Search General Info API & Language Guide to Features Tool Docs Demos/Tutorials

J2SE™ Platform at a Glance

This document covers the Java™ 2 Platform Standard Edition 5.0 Development Kit (JDK 5.0). Its external version number is 5.0 and internal version number is 1.5.0. For information on a feature of the JDK, click on its component in the diagram below.

Java™ 2 Platform Standard Edition 5.0



Search the Documentation

Location

Search the online documentation — website

New to the Java Platform?

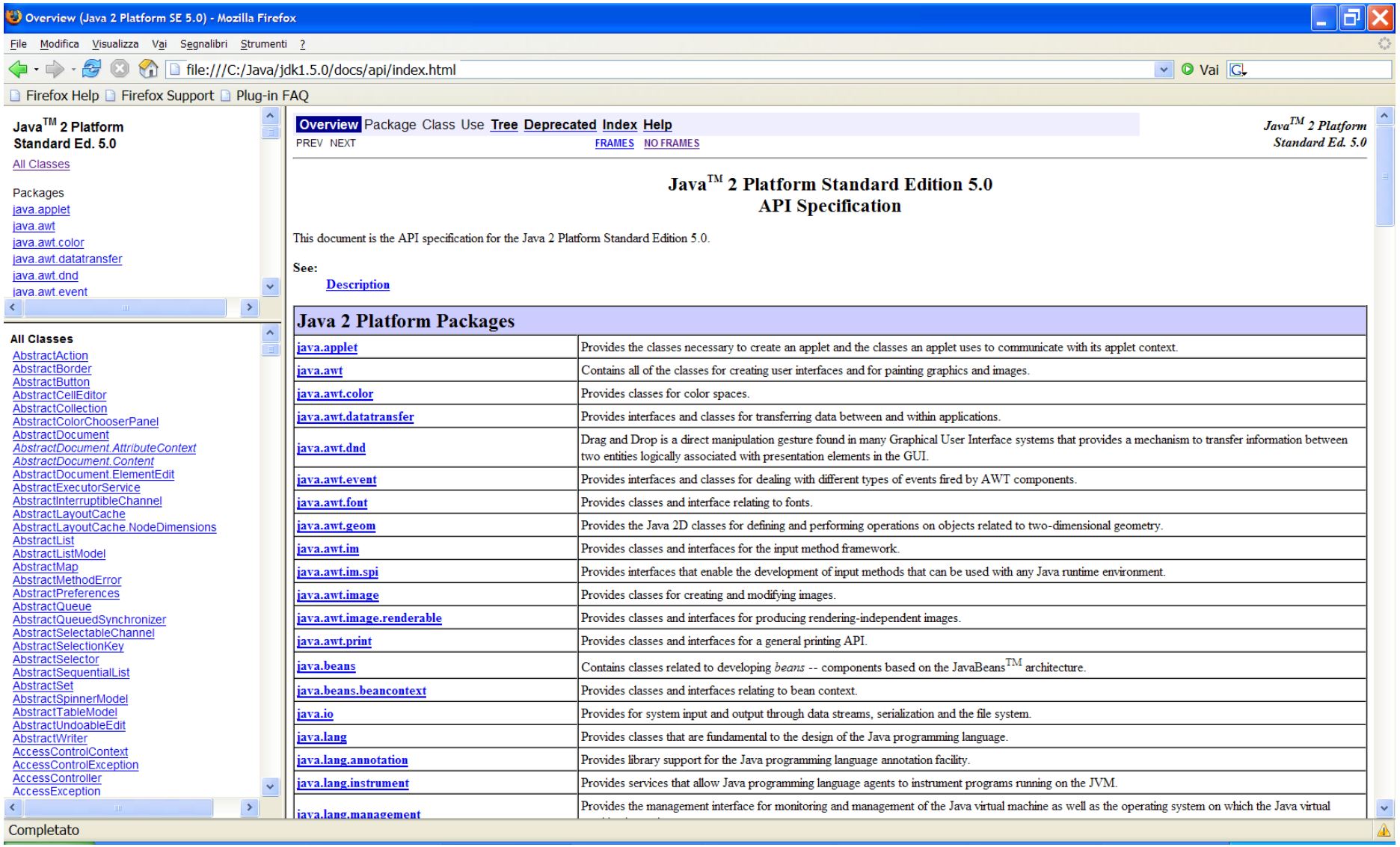
See the New-to-Java™ Programming Center. — website

General Information

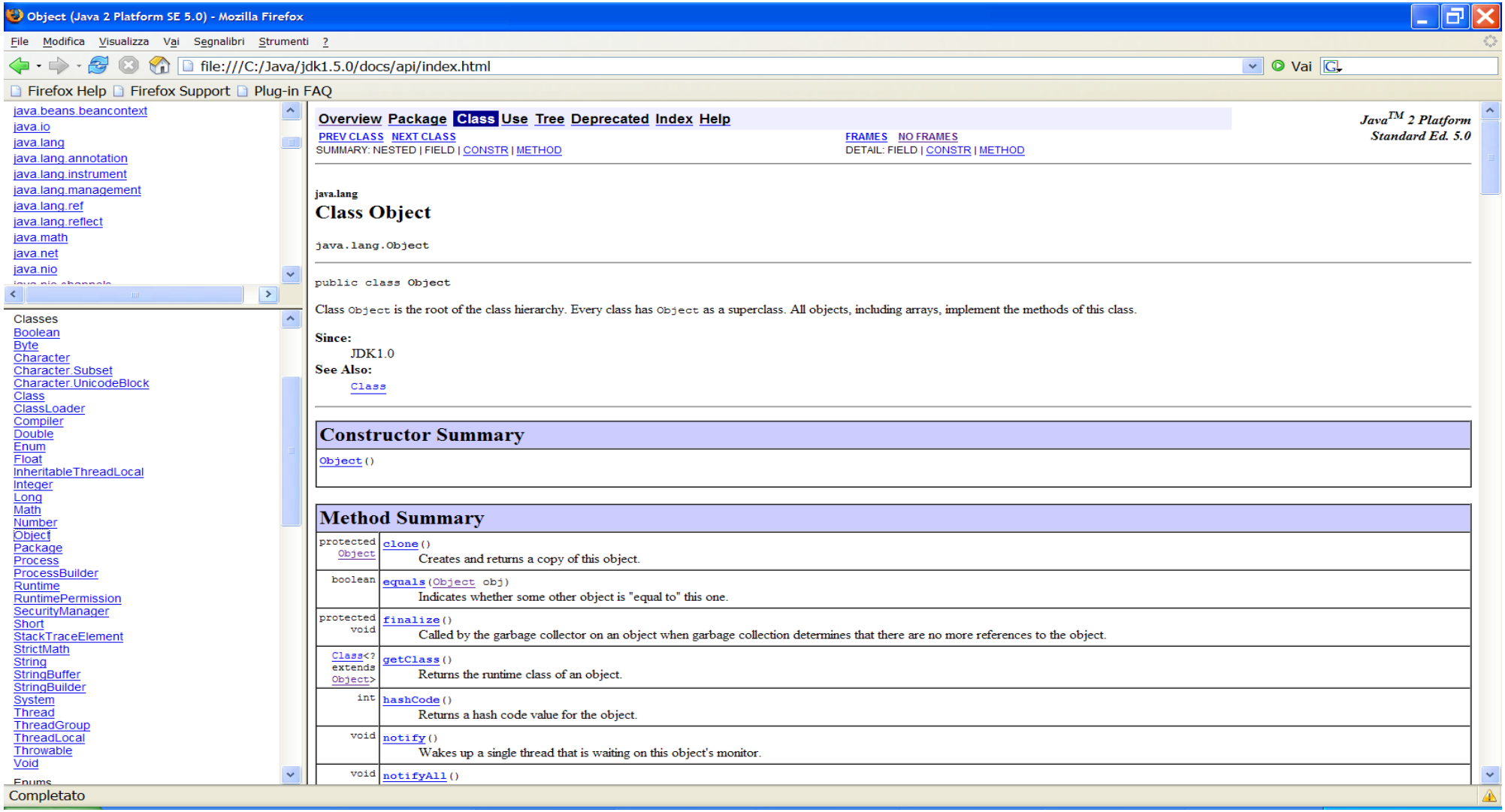
Readme, Overview, Changes

Completato

Java API Documentation



Java API Documentation



Java API Documentation

» Si può usare il comando `javadoc` per generare automaticamente tale documentazione di API per nuove classi

» `javadoc` utilizza una convenzione speciale dei commenti per consentire di mettere informazioni pertinenti API. Questi commenti sono della forma `/** **/`. Dentro tale commenti si può mettere qualunque 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

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