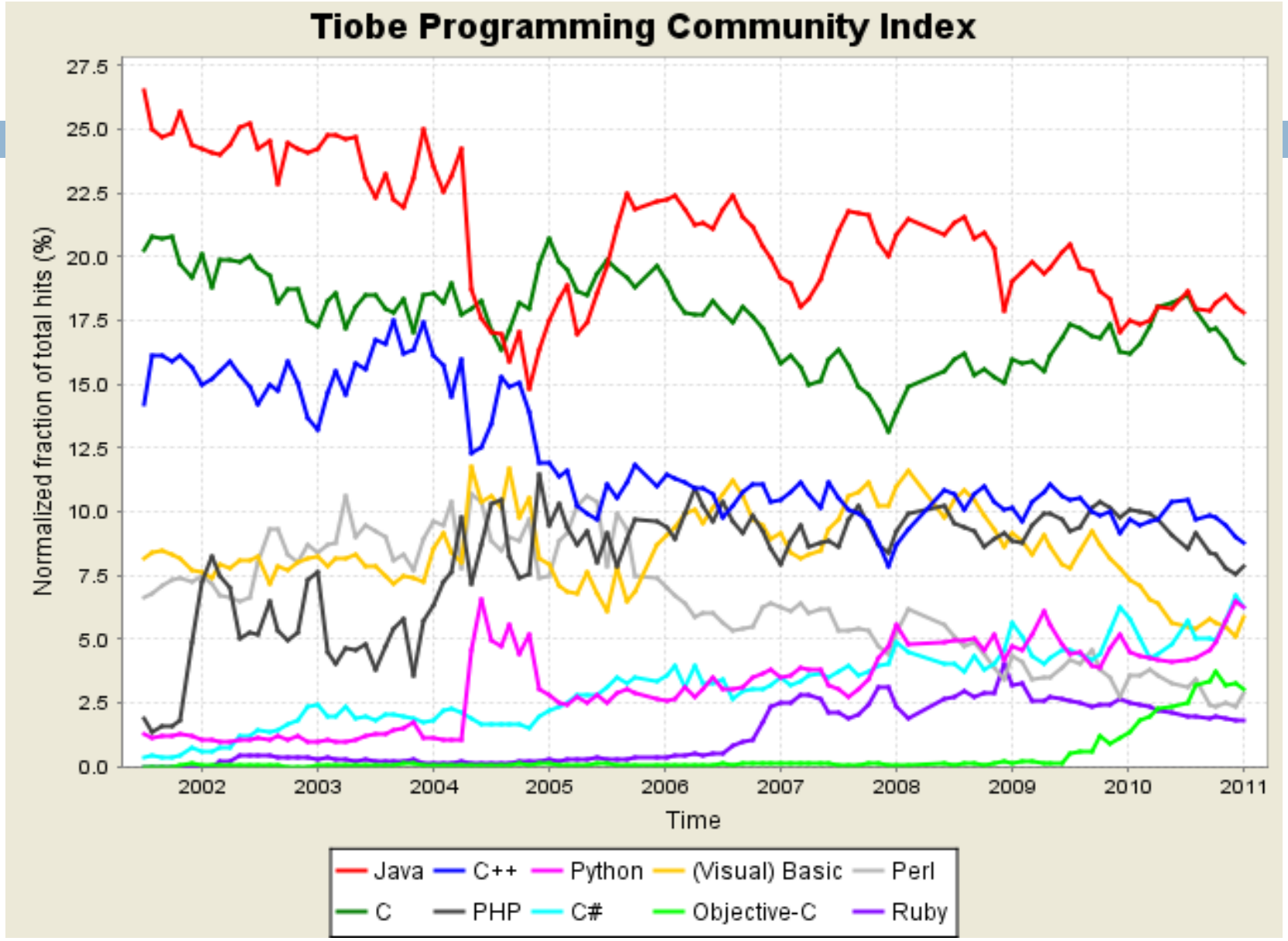
# Linguaggi 2 e laboratorio

- Docente: Leonardo Badia
- Contatti: **leonardo.badia@gmail.com**
- Pagina web del corso:
  **www.unife.it/scienze/informatica/insegnamenti/linguaggi-2-laboratorio**

# Argomenti del corso: Java

- Java è un linguaggio di programmazione a oggetti, di grande diffusione e versatilità
- Consente di realizzare interfacce grafiche e applicazioni multimediali per i browser web.
- Viene supportato da tutte le piattaforme, grazie alla presenza della macchina virtuale Java (Java Virtual Machine), ed è quindi estremamente portabile.


Tiobe Programming Community Index
Normalized fraction of total hits (%)
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
Time
Java
C++
Python
(Visual) Basic
Perl
C
PHP
C#
Objective-C
Ruby

# Materiale didattico

- **R. Bruni, A. Corradini, V. Gervasi, "Programmazione in Java", ed. Apogeo, 2009**
  - K. Arnold, J. Gosling, D. Holmes, "JAVA. Manuale ufficiale", Addison-Wesley, 2001.
  - C.S. Horstmann, "Concetti di informatica e fondamenti di Java 2", Apogeo, 2005.
  - J. Cohoon, J. Davidson, "Java guida alla programmazione", McGraw-Hill, 2004.
  - G. Pizzighini, M. Ferrari, "Dai fondamenti agli oggetti – corso di programmazione JAVA", Addison-Wesley, 2005.
  - L. Cabibbo, "Fondamenti di informatica, oggetti e Java", McGraw-Hill, 2004.

# Modalità d'esame

- L'esame standard comprende 3 fasi, da sostenersi in questo preciso ordine:

- Compito scritto (sostenibile anche suddiviso in due parti – c.d. "compitini")
- Progetto di programmazione (da svolgere in proprio, tipicamente in gruppi, e che viene discusso all'inizio dell'esame orale)
- Esame orale vero e proprio

# Java

- Java è un linguaggio di programmazione sviluppato da Sun Microsystems nel 1995.
  - Nel gennaio 2010, Sun è stata rilevata, e con essa Java, da Oracle, diventando Oracle America, Inc.
- È un linguaggio **imperativo** di tipo **strutturato** e **orientato agli oggetti**. Deriva molta della sua sintassi da C e C++, pur essendo più semplice.
- Funziona su piattaforme molto differenti tra loro, secondo il motto “write once, run anywhere” (**WORA**): scrivi una volta, esegui ovunque.

# Proprietà di Java

- Rispetto ad altri linguaggi orientati agli oggetti, Java possiede i seguenti vantaggi:
  - i programmi sono più facili da leggere
  - il codice è più portabile (WORA)
  - il codice è robusto e con meno errori
  - esistono ampie collezioni di librerie (chiamate *package*) che forniscono svariate funzionalità
- Tutto questo rende Java estremamente potente nonostante la sua relativa semplicità

# Programmazione a oggetti

- Programmare serve generalmente a risolvere un problema o far eseguire un compito al calcolatore.
- Fasi della programmazione:
  - Analisi del problema e sua comprensione
  - Sviluppo di un **algoritmo** per risolverlo
  - Traduzione dell'algoritmo in un programma scritto mediante un opportuno linguaggio

# Programmazione a oggetti

- In un linguaggio orientato agli oggetti come Java, la programmazione consiste nel definire oggetti che interagiscono tra di loro.
- Questo è opposto a quanto accade nei linguaggi di tipo **procedurale** dove ci si focalizza sulla definizione di procedure (routine) che si invocano a vicenda.

# Programmazione a oggetti

- In Java implementare un algoritmo risolutivo corrisponde alla definizione di una o più tipologie di oggetti che interagiscono tra loro.
  - Progettare gli oggetti, anziché elencare le procedure da compiere.
- Un progetto di tipologia di oggetto è detto **classe**.
- Programmare in Java significa quindi **definire una o più classi**.

# Programmazione a oggetti

- La programmazione Java si sviluppa in:
- EDITING del programma, tipicamente partendo da più file, ciascuno contenente la descrizione di una classe.
  - ciò favorisce la portabilità (in senso WORA).
- COMPILAZIONE del programma che ne consente l'ESECUZIONE da parte del calcolatore

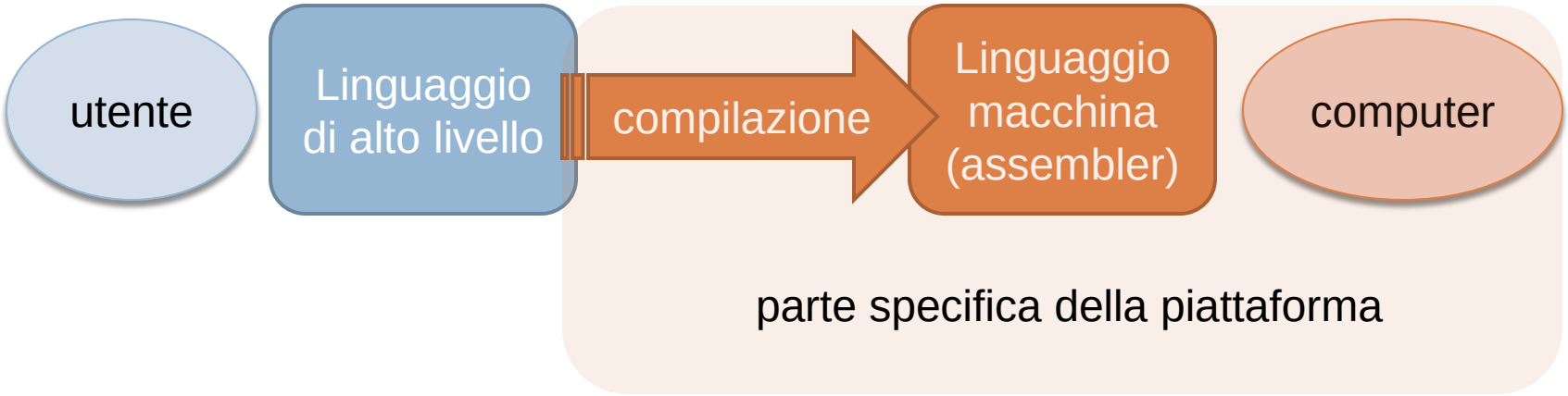
# Editing

- L’editing è la produzione del programma (una o più classi) secondo la sintassi del linguaggio ottenuta anche con un semplice editor di testo.
  - Esistono tuttavia ambienti di sviluppo dove i successivi passaggi (compilazione, esecuzione) avvengono in modo integrato
- Tuttavia tale programma non è direttamente eseguibile dall’elaboratore. Questo avviene poiché il linguaggio astrae dalla piattaforma.

# Compilazione

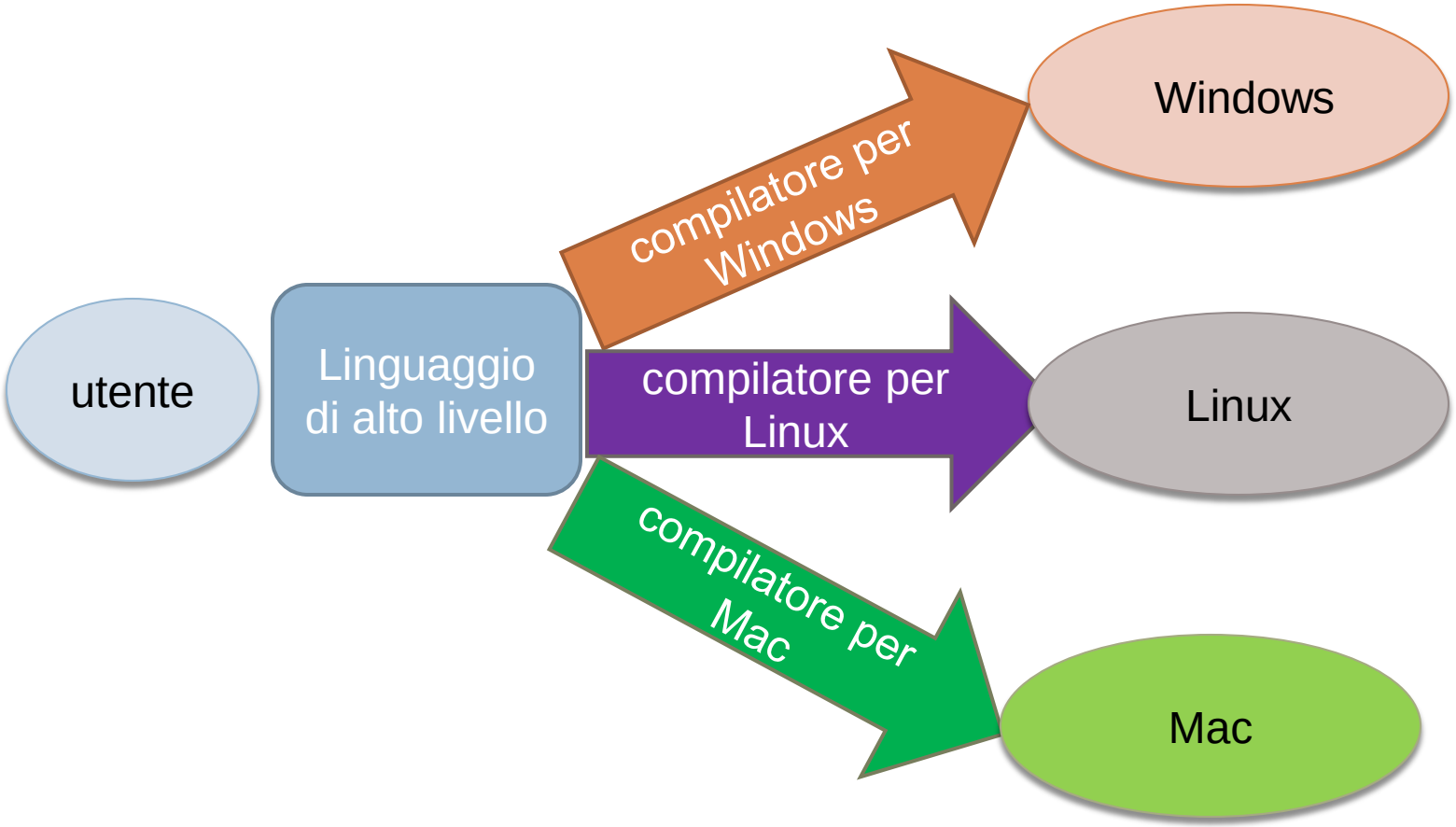
- La compilazione traduce il **codice** del programma dal linguaggio di programmazione al **linguaggio macchina** della piattaforma.
- Il codice originario (sorgente) è scritto in maniera comprensibile per gli utenti umani, il codice compilato (eseguibile) è invece utilizzabile dall'elaboratore, e può riferire aspetti specifici della piattaforma sconosciuti o troppo complessi per l'utente umano.

# Compilazione




- La parte “portabile” della programmazione è quella precedente alla compilazione.

# Compilazione



# Java Virtual Machine

- Questo meccanismo di compilazione consente di ottenere un codice sorgente portabile; tuttavia, il codice compilato non lo è (dipende dalla piattaforma).
- Per essere interamente WORA, Java utilizza un approccio di compilazione misto, con un ulteriore livello di processazione del codice.
- Tale livello, detto di INTERPRETAZIONE, ha lo scopo di rendere la compilazione indipendente dalla piattaforma.

# Java Virtual Machine

**In generale**

- Editing
- Compilazione
- Esecuzione

**Per Java**

- Editing
- Compilazione
- **Interpretazione**
- Esecuzione

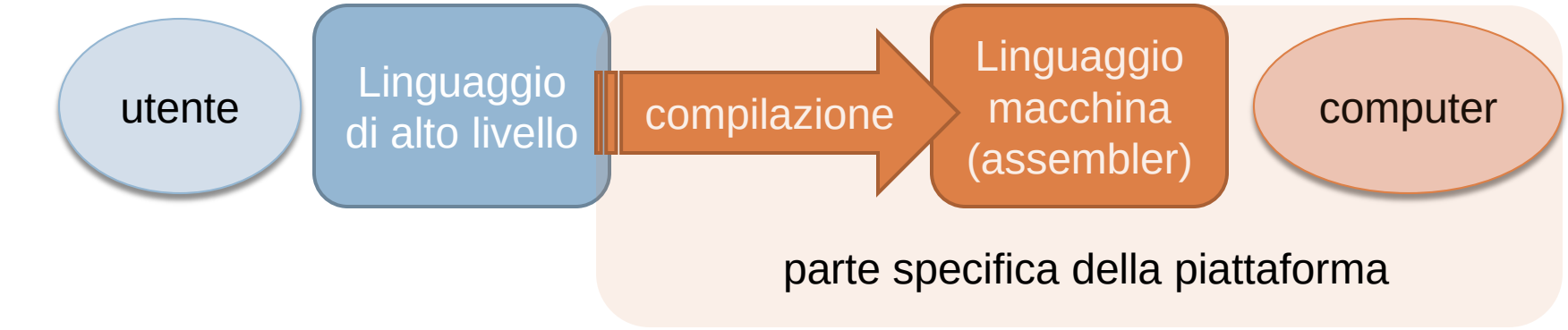
# Java Virtual Machine

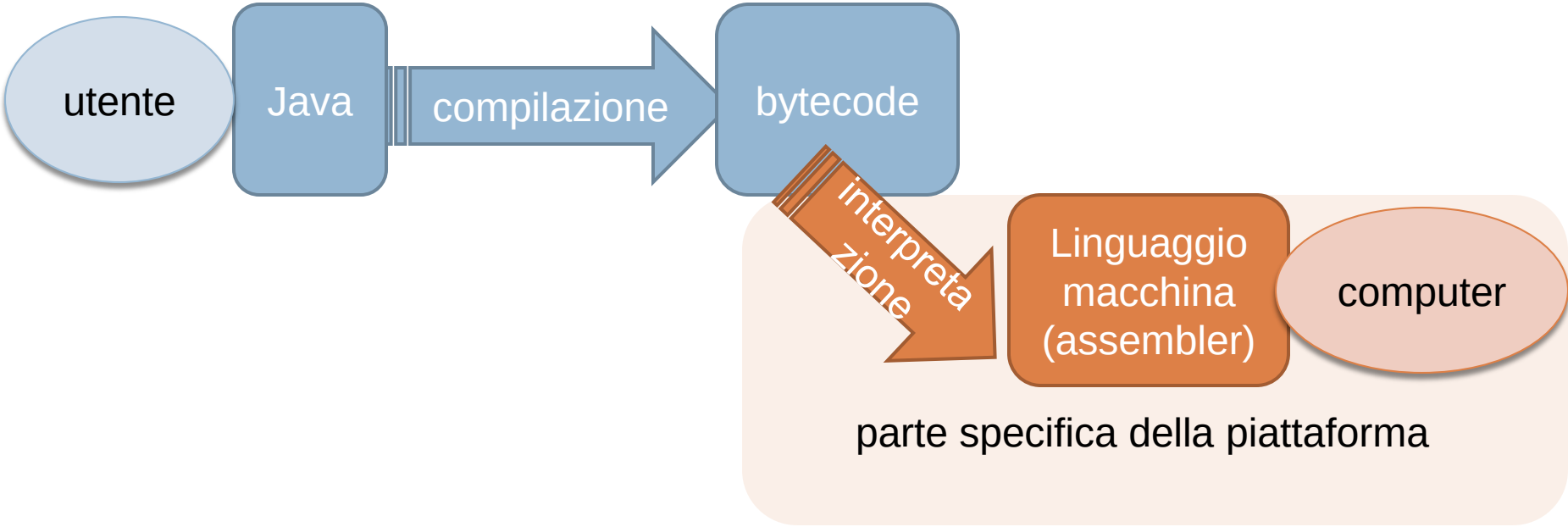
- Il fatto che il codice compilato (e il compilatore stesso) siano dipendenti dalla piattaforma impedisce la portabilità del programma compilato su ogni macchina.
- Ciò contrasta con l'obiettivo primario di Java, e giustifica la necessità di inserire un ulteriore livello nel processo di traduzione del codice.

# Java Virtual Machine

- Nella fase intermedia (tra compilazione e interpretazione) si avrà un formato diverso sia da Java che dall'assembler, detto **bytecode**.
- Quindi:
  - il sorgente viene compilato ottenendo il bytecode
  - il programma espresso come bytecode può essere eseguito da un opportuno interprete chiamato **Java Virtual Machine** (JVM)

# Java Virtual Machine



# Java Virtual Machine

- Una JVM è un interprete in grado di tradurre il bytecode, e quindi eseguire i programmi Java. Questi possono essere di tipo
  - *applicazione*: intesi come programma a sé stante
  - *applet*: se vanno eseguiti dalla JVM di un browser all'interno di una pagina web
  - *servet*: se vanno eseguiti dalla JVM di un server web così da generare pagine dinamiche
- Applicazioni, applet, servlet differiscono per aspetti di sintassi e di ambiente d'esecuzione.

# Java Virtual Machine

- La JVM è a sua volta un programma scritto in un linguaggio **dipendente** dalla piattaforma.
- Quindi il codice compilato è sì portabile, ma a patto che sulle diverse piattaforme sia installata una JVM.
- Tuttavia, questo garantisce di **non dover ricompilare** il codice per ogni piattaforma.

# Java Virtual Machine

- Vantaggi dell'approccio Java
  - il bytecode non è l'assembler, ma un linguaggio più espressivo (di più alto livello) e indipendente dalla piattaforma; un programma in formato bytecode può essere eseguito su ogni JVM
  - Java consente di astrarre dalle caratteristiche della piattaforma ed elimina i costrutti dipendenti (formati numerici, stringhe… sono sempre uguali)
  - è la JVM a fare il lavoro di traduzione adattandosi alle caratteristiche della piattaforma

# Applet

- Gran parte del successo di Java deriva anche dall'utilizzo che ne viene fatto sul web, in particolare per la creazione di interfacce grafiche (Graphic User Interface, GUI).
- Praticamente ogni browser contiene una JVM in grado di eseguire applet Java contenute (sottoforma di bytecode) in pagine remote.
- Dato che quello che serve per l'esecuzione è il bytecode, non c'è bisogno di adattare il programma alla piattaforma che lo eseguirà.

# Gestione della memoria

- Java semplifica la gestione della memoria con un meccanismo detto **garbage collector**.
- Molti linguaggi richiedono allocazioni e de-allocazioni esplicite di memoria (malloc, free).
- Java invece alloca memoria al bisogno (al primo riferimento di un oggetto). Il garbage collector tiene traccia dei riferimenti agli oggetti, liberando la memoria di quelli che non sono più referenziati.

# Multi-thread

- Java è un linguaggio **multi-threaded**. In altri termini, può sfruttare logiche concorrenti.
- Questo significa che parti diverse del programma possono essere eseguite come se *(dal punto di vista di un utente/programmatore umano)* fossero processate in parallelo.

# Ambiente di sviluppo

- Inizialmente **Java Development Kit** (JDK): ambiente di sviluppo contenente gli strumenti necessari alla compilazione ed esecuzione, in particolare la JVM.
- Disponibile per diverse piattaforme hardware/software, rilasciato in maniera aderente allo standard “de facto.”
- Sun adotta una filosofia “free software” (sia nel senso di gratuito, sia nel senso di open source secondo licenza GNU).

# Ambiente di sviluppo

- Con le versioni successive, è stata sviluppata una piattaforma più avanzata: Java 2.
- Java 2 è completamente retro-compatibile ma offre migliori prestazioni di stabilità e gestione della memoria.
- Inoltre fornisce diverse “varianti” al JDK, estendendolo a un più generale Software Development Kit (SDK).

# Ambiente di sviluppo

- Sun distingue tra SDK/JDK e Java Runtime Environment (JRE).
  - JRE è un sottoinsieme del JDK che serve solo ad eseguire, ma non a sviluppare. È quindi privo del compilatore, di alcuni software di utilità, e della documentazione.
  - Per eseguire applicazioni Java è sufficiente JRE.

# Ambiente di sviluppo

- Il JDK viene fornito inoltre in 3 versioni, leggermente dipendenti dalla piattaforma:
  - Java EE (Enterprise Edition) per applicazioni di tipo industriale;
  - Java SE (Standard Edition), versione standard;
  - Java ME (Mobile Edition), con caratteristiche molto ridotte, orientata a dispositivi portatili (per esempio, cellulari o palmari).

# API

- Una **Application Programming Interface** (API) è un software che serve a interfacciare un programma con altro software.
  - Svolge cioè la stessa funzione di un'interfaccia utente, solo che l'utente in questo caso è un altro programma.
- In particolare le tre versioni del JDK di Java sono distinte per quanto riguarda il set delle API che viene fornito.

# API

- In Java, vengono infatti fornite diverse API con funzioni di libreria. Esse sono **già compilate** sotto forma di bytecode.
- Il programmatore che volesse utilizzarle deve specificare quali vuole inserire nel suo programma.
- Il programma compilato contiene solo il bytecode relativo al sorgente scritto dal programmatore, non quello delle API.

# API

- Le API fornite con il JDK contemplano molte funzioni di uso comune: fare riferimento alla documentazione fornita.
- Le classi di un programma possono quindi essere:
  - Classi di libreria già fornite dal JDK (API)
  - Classi scritte dal programmatore in precedenza che vengono riusate
  - Classi nuove scritte all'occorrenza.

# Dynamic loading and linking

- Java memorizza ogni classe in un singolo file.
- All'atto dell'esecuzione, le classi vengono caricate in memoria.
- Java supporta la metodologia di **dynamic loading and linking**, che significa che le classi vengono caricate in memoria solo quando necessarie.
- In questo modo, ad esempio, solo le API di libreria usate vengono caricate in memoria.

# Caratteristiche base

- Java è case-sensitive, come il C. Distingue quindi tra maiuscole e minuscole.
  - ovvero: **`prova`**, **`Prova`**, **`PROVA`** sono tre identificativi differenti.
- Java inoltre non distingue intrinsecamente spazi e interruzioni di linea. Quindi è possibile un po' di libertà (nei limiti del buon gusto per la leggibilità) nell'organizzare i programmi.

# Installazione di Java

- I file sono reperibili sul web navigando sul sito:

  **`www.oracle.com/technetwork/java/`**

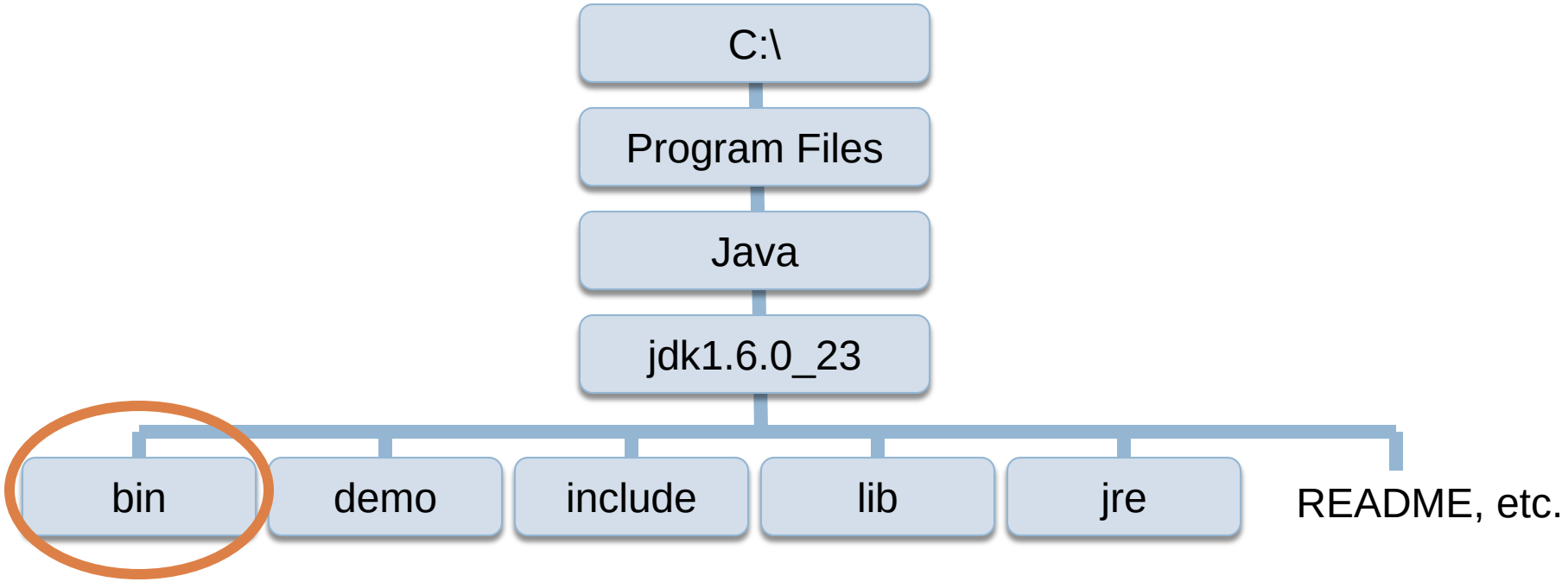
  che ha rimpiazzato (l'ancora funzionante)

  **`java.sun.com`**

- La versione più recente da installare del JDK (ad oggi) è la release 1.6 update 23.
- Supponiamo di scaricarla e scompattarne i file di installazione sotto il path

  **`C:\Program Files\Java`**

# Direttorio di Java




- Ci interessa in particolare la sottodirectory **`bin`**, comprendente, tra l'altro, il compilatore **`javac`**, la JVM **`java`** e le API.

# Programmazione

- La fase di EDITING richiede un qualsiasi editor. È consigliabile usare un ambiente integrato (Integrated Development Environment, IDE), ad es.: eclipse
- La fase di COMPILAZIONE richiede di utilizzare il compilatore javac.
- La fase di INTERPRETAZIONE ed ESECUZIONE usa direttamente la JVM java.

# Editing

- Ci sono diversi editor integrati, molti di questi supportano lo sviluppo mediante **progetti**.
- Un progetto è un insieme di file (corrispondente quindi a un insieme di classi) pensati per interagire.
- Uno di essi contiene il punto di partenza **main**.
- Risulta possibile quindi operare su un progetto ed eseguirlo tutto direttamente dall'IDE.

# Editing

- Nella produzione del file di testo contenente il progetto di una classe andrà usata l'estensione **`.java`** .
- È buona norma di programmazione chiamare il file con lo stesso nome della classe contenuta in esso.
- Quindi, un file = una classe, e i nomi coincidono.

# Compilatore

- javac funziona con un argomento da linea di comando dato dal nome che si vuole compilare, che deve avere l'estensione **`.java`**
- Ad esempio **`javac Prova.java`** produce la compilazione del file **`Prova.java`** (che contiene la descrizione della classe **`Prova`**).
- Questa compilazione può essere svolta a riga di comando o da IDE.

# Compilatore

- Al compilatore possono essere passati inoltre diversi parametri, tra cui:

  -O (**`javac -o Prova.java`**) attiva l'ottimizzazione del codice prodotto

  -g (**`javac -g Prova.java`**) attiva il supporto debug

  -classpath *path* specifica il percorso dove cercare le funzioni di libreria; può anche essere specificato tramite la variabile di ambiente **`CLASSPATH`**

# Compilatore

- Se quindi si vuole dire al compilatore che il path di riferimento per le librerie è <PATH1>, si può tanto usare:

  **`javac -classpath`** <PATH1> **`Prova.java`**

- quanto:

  **`set CLASSPATH =`** <PATH1>

  **`javac Prova.java`**

# Esecuzione

- Il compilatore produce in output un file con lo stesso nome e l'estensione **`.class`**, nell'esempio è **`Prova.class`**
- Solitamente, file sorgente e bytecode sono tenuti nella stessa cartella
- Questo file è scritto in bytecode e può essere eseguito dalla JVM, invocando **(senza indicare l'estensione `.class`):**

```
java Prova
```

# Package

- Prima di scrivere una classe da zero, conviene vedere se non esiste già una classe di libreria che fa allo scopo.
- Java considera già aggiunto ad ogni programma il package **`java.lang`**, che è già fornito con l'ambiente di sviluppo.
- In particolare, questo package contiene la classe **`System`**, che contiene le funzioni più comuni di I/O (scrittura a video, input, etc.)

# Caricamento delle classi

- Se in un programma Java vogliamo utilizzare una classe che **non** appartiene al package java.lang, dobbiamo dirlo esplicitamente al compilatore, usando la direttiva

  ```
  import <nomedellaclasse>;
  ```

  dove <nomedellaclasse> comprende anche il percorso della directory di appartenenza.
- Anche qui, o usiamo percorsi assoluti, o ci riferiamo a **`CLASSPATH`**.

# Caricamento delle classi

- Nell'importare una classe di un altro package va usato il carattere . (punto) come separatore:

  **`import prog.test.daImportare`**

  significa che si vuole importare la classe (da percorso relativo): prog/test/daImportare.class
- Ai percorsi assoluti di **`CLASSPATH`** il programmatore può aggiungere le sue directory di utilità con le sue funzioni di libreria.

# Caricamento delle classi

- La JVM carica quindi nell'ordine:
  - le classi di base
  - le classi di libreria che si desidera importare
  - le classi definite dal programmatore
- La variabile di ambiente **`CLASSPATH`** deve quindi essere letta:
  - in fase di compilazione, per poter compilare le classi definite dal programmatore
  - in fase di esecuzione, per importare correttamente