Linguaggi di Programmazione I – Lezione 7

Prof. Marcello Sette
mailto://marcello.sette@gmail.com
http://sette.dnsalias.org

22 aprile 2008

Panoramica della lezione

Introduzione a Java

Un esempio

JVM e JRE

Un po' di sintassi

Incapsulazione

Costruttori

File sorgenti e pacchetti

Cartelle e pacchetti

Introduzione a Java 3 / 41

Tecnologia Java

- **Un linguaggio di programmazione**, con sintassi simile a C++ e semantica simile a SmallTalk. È di solito menzionato (ma non serve solo) per produrre *applet*: applicazioni WEB che risiedono sul server ma sono eseguite dal browser WEB del cliente.
  Le *applicazioni* Java sono programmi autonomi che non richiedono un browser WEB per essere eseguiti. Tali programmi richiedono solo un elaboratore su cui sia installato *Java Runtime Environment* (JRE).
- **Un ambiente di sviluppo**, cioè un insieme di numerosi strumenti per la realizzazione di programmi: un compilatore, un interprete, un generatore di documentazione, uno strumento di aggregazione di file, ...

I sorgenti

File: `Saluti.java`

```
public class Saluti {
  private String saluto;
  Saluti(String s) {
    saluto = s;
  }
  public void saluta(String chi) {
    System.out.println(saluto + " " + chi);
  }
}
```

File: `ProvaSaluti.java`

```
public class ProvaSaluti {
  public static void main(String[] args) {
    Saluti ciao = new Saluti("Ciao");
    Saluti arrivederci = new Saluti("Arrivederci");
    ciao.saluta("Alberto");
    ciao.saluta("Marcello");
    arrivederci.saluta("Stefania");
  }
}
```


Compilazione ed esecuzione

Posto che i due file precedenti siano nella cartella corrente, il comando:

```
$ javac ProvaSaluti.java
```

produrrà il file oggetto `ProvaSaluti.class`, ma anche il file `Saluti.class`.
Una esecuzione attraverso l'interprete Java:

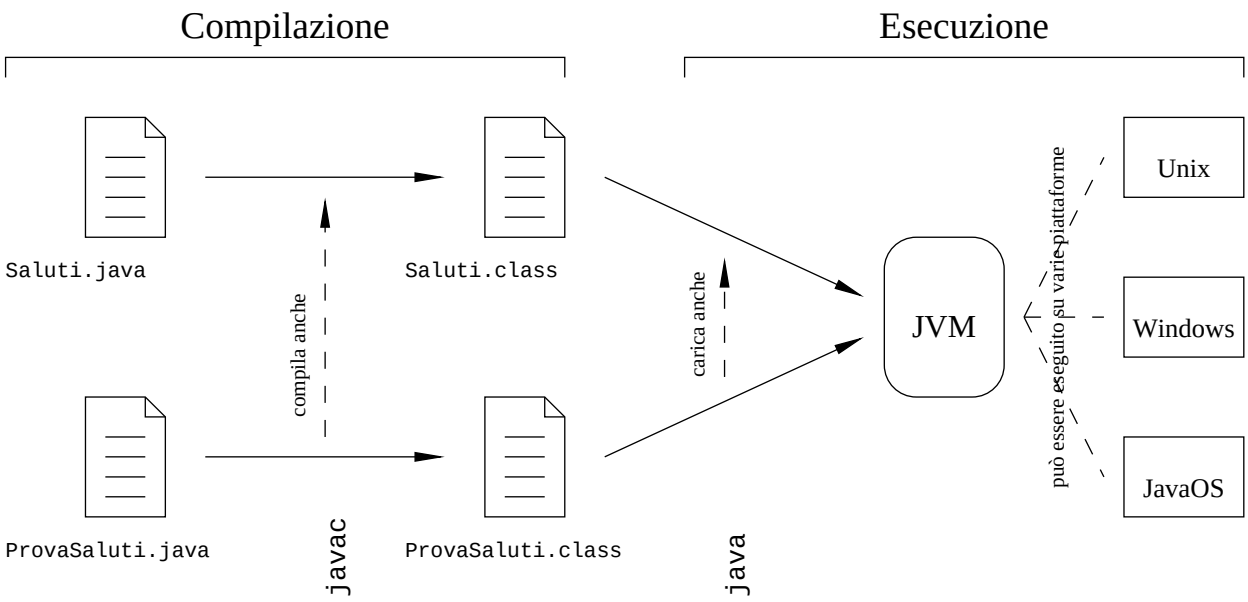
```
$ java ProvaSaluti
Ciao Alberto
Ciao Marcello
Arrivederci Stefania

$
```

Errori comuni

- Errori in compilazione:
  - Installazione JavaSDK non corretta.
  - Nome di metodi di sistema non corretti.
  - Errata corrispondenza tra nome di classe e nome di file.
  - Numero di classi pubbliche in un file.
- Errori in esecuzione:
  - Errata scrittura della classe da caricare inizialmente in memoria.
  - Errata scrittura della firma del metodo main.

L'esempio esemplificato



JVM

La Java Virtual Machine

è una macchina immaginaria che è emulata dalla macchina reale. I programmi per la JVM sono contenuti in file `.class`*, ciascuno dei quali contiene al più una classe pubblica*

Quindi, la JVM:

- fornisce le specifiche della piattaforma hardware;
- legge i codici *bytecode* compilati, che sono indipendenti dalla piattaforma;
- è realizzata in software o in hardware;
- è realizzata come strumento di sviluppo software oppure in un browser Web.

Punti di vista

Dal punto di vista della realizzazione, la JVM specifica:

- l'insieme di istruzioni della CPU;
- l'insieme dei registri;
- il formato del file Class;
- lo stack di esecuzione;
- lo heap gestito dal garbage collector;
- l'area di memoria.

Dal punto di vista dell'utilizzatore:

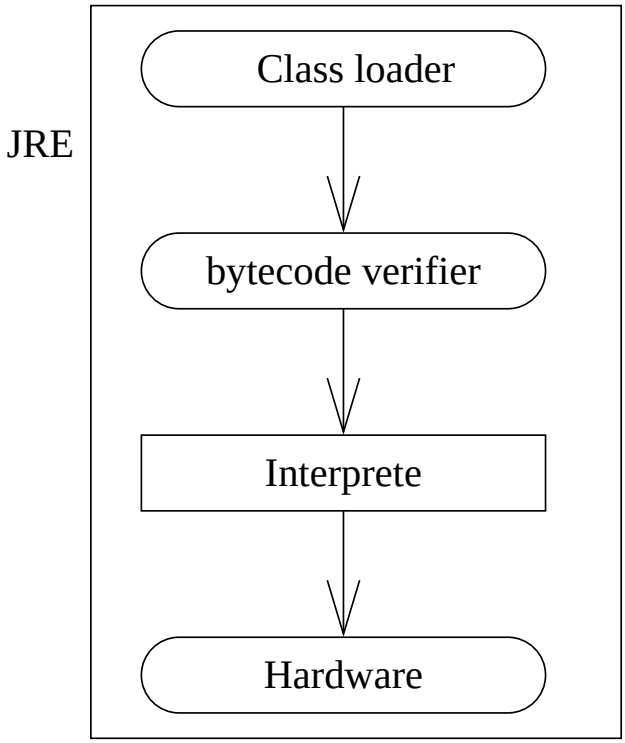
- Il formato del programma compilato, eseguibile dalla JVM, consiste in *bytecode* molto compatti ed efficienti.
- La maggior parte del type-checking è svolto durante la compilazione.
- Ogni realizzazione della JVM deve essere capace di eseguire un qualunque programma conforme alle specifiche della JVM.

Garbage Collector

- La memoria allocata che non è più necessaria deve essere resa disponibile.
- In altri linguaggi la de-allocazione è responsabilità del programmatore.
- Java fornisce un thread a livello di sistema per tracciare l'allocazione di memoria.
- Il *Garbage Collector*:
  - ricerca e libera la memoria non più necessaria;
  - è eseguito automaticamente;
  - è dipendente dalle varie realizzazioni di JVM.

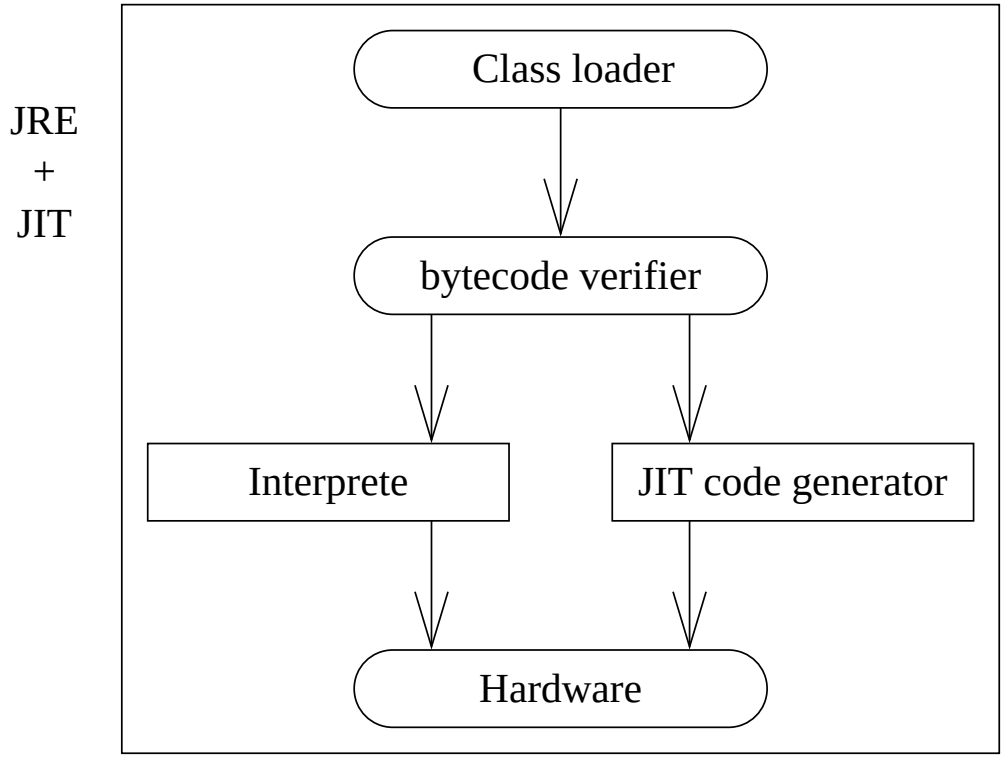
JRE

Java Runtime Environment è una realizzazione della JVM. In dettaglio:



JRE + JIT

Per migliorare la velocità di esecuzione:



Sicurezza

Miglioramento della sicurezza del codice che proviene da fonti esterne.

- Il *Class Loader*:
  - carica solo le classi necessarie per l'esecuzione del programma;
  - mantiene le classi del file system locale in uno spazio di nomi separato: ciò limita applicazioni "cavallo di Troia", poiché le classi locali sono sempre caricate per prime;
  - previene il così detto *spoofing*.
- Il *Bytecode Verifier* controlla che:
  - il codice rispetti le specifiche della JVM;
  - il codice non violi l'integrità del sistema;
  - il codice non generi stack overflow o underflow;
  - non vi siano conversioni di tipo illegali (e.g. la conversione di interi in puntatori).

Esercizi

1. Correggere gli errori nei frammenti di codice forniti.
2. Scrivere, compilare ed eseguire il programma precedente `ProvaSaluti`.

Classi

La sintassi per la dichiarazione di una classe è:

```
<class_declaration> ::= <modifier> class <name> {
                          <attribute_declaration>*
                          <constructor_declaration>*
                          <method_declaration>*
                        }
```

Esempio:

```
public class Automobile {
  private double maxPosti;
  public void setMaxPosti(double valore) {
    maxPosti = valore;
  }
}
```


Attributi

La sintassi per la dichiarazione di un attributo è:

```
<attribute_declaration> ::=
   <modifier> <type> <name> [= <default_value>];

<type> ::= byte | short | int | long | char |
           float | double | boolean | <class>
```

Esempio:

```
public class Foo {
  public int x;
  public float y = 10000.0F;
  private String nome = "Marcello Sette";
}
```


Metodi

La sintassi per la dichiarazione di metodi è:

```
<method_declaration> ::=
   <modifier> <return_type> <name> (<parameter>*) {
      <statement>*
   }
```

Esempio:

```
public class Cosa {
  private int x;
  public int getX () {
    return x;
  }
  public void setX (int nuovo_x) {
    x = nuovo_x;
  }
}
```

Accesso a membri di oggetto

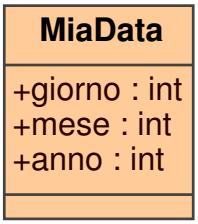
- La notazione "dot": `<object>.<member>`.
- È usata per l'accesso a membri **non privati** di una oggetto.
- Esempio:

```
public class TestCosa {
  public static void main (String[] args) {
    Cosa cc = new Cosa();

    cc.setX(47);
    System.out.println("cc.x vale " + cc.getX());
  }
}
```


Incapsulazione 23 / 41

Il problema



Il codice cliente ha accesso diretto ai dati interni:

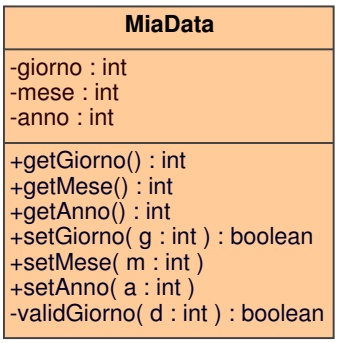
```
MiaData d = new MiaData();

d.giorno = 32;
// non valido

d.mese = 2; d.giorno = 30;
// plausibile ma sbagliato

d.giorno = d.giorno + 1;
// nessun controllo del limite superiore
```

La soluzione



Il cliente deve usare i metodi per accedere ai dati interni:

```
MiaData d = new MiaData();

d.setGiorno(32);
// non valido, ma ritorna falso

d.setMese(2);
d.setGiorno(30);
// plausibile ma sbagliato; setGiorno ritorna false

d.setGiorno(d.getGiorno() + 1);
// restituisce false se si eccede il limite
```


Costruttori 26 / 41

Sintassi

La sintassi per la dichiarazione di un costruttore è:

```
<constructor_declaration> ::=
   <modifier> <class_name> (<parameter>*) {
      <statement>*
   }
```

Esempio:

```
public class Cosa {
  private int x;
  public Cosa () {
    x = 47;
  }
  public Cosa (int nuova_x) {
    x = nuova_x;
  }
}
```

Attenzione

- i costruttori NON sono metodi;
- i costruttoni NON sono ereditati;
- un costruttore NON ha valore di ritorno;
- gli unici modificatori validi per il costruttore sono: `public`, `protected` e `private`.

Costruttore di default

- C'è sempre almeno un costruttore in ogni classe.
- Se il programmatore non scrive nessun costruttore, allora verrà usato un costruttore di default.
- Il costruttore di default non ha argomenti.
- Il costruttore di default non ha corpo.
- Il costruttore di default permette di creare istanze di classi (con l'espressione `new Xxx()`) senza dover scrivere un costruttore.

Attenzione: Se si aggiunge una dichiarazione di costruttore con argomenti ad una classe che in precedenza non aveva costruttori espliciti, allora si perde il costruttore di default. Da quel punto in poi, una chiamata a `new Xxx()` genererà un errore di compilazione.

File sorgenti e pacchetti 30 / 41

Sorgenti

```
<source_file> ::=
   [<package_declaration>]
   <import_declaration>*
   <class_declaration>*
```

- l'ordine degli elementi è importante;
- il nome del file sorgente deve essere lo stesso dell'unica classe pubblica contenuta nel file; se il file sorgente non contiene classi pubbliche allora il nome del file può essere qualunque.
  Esempio: il file `RapportoCapacitaVeicolo.java`

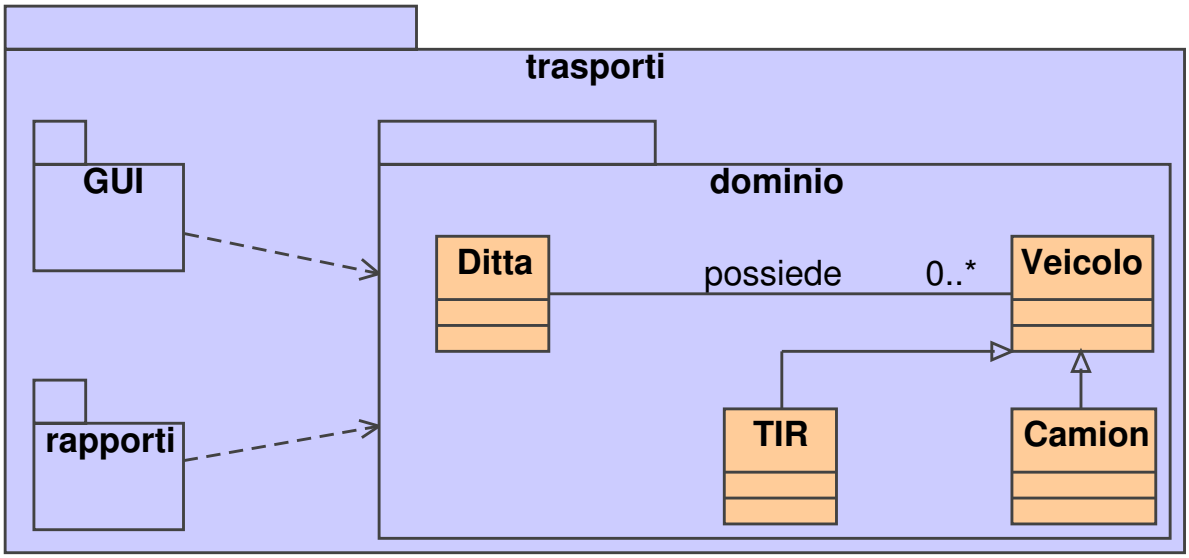
```
package trasporti.rapporti.Web;

import trasporti.dominio.*;
import java.util.List;
import java.io.*;

public class RapportoCapacitaVeicolo {
  private List veicoli;
  public void generaRapporto (Writer output) {
  ...
  }
}
```

Pacchetti

- Aiutano a gestire grandi sistemi software.
- Possono contenere classi e sotto-pacchetti.



Enunciato `package`

La sintassi dell'enunciato `package` è:

```
<package_declaration> ::=
   package <top_pkg_name>[.<sub_pkg_name>]*;
```

ove:

- la dichiarazione specifica che il contenuto del file appartiene al pacchetto dichiarato;
- è permessa una sola dichiarazione di pacchetto per file sorgente;
- si deve specificare la dichiarazione all'inizio del file sorgente;
- se non è dichiarato il pacchetto, allora il contenuto del file appartiene al pacchetto di default;
- i nomi di pacchetto devono essere gerarchici e separati da punti;
- un nome di pacchetto corrisponde di solito ad un nome di cartella;
- in genere i nomi di pacchetto sono scritti in lettera minuscola, mentre i nomi di classe sono in lettera minuscola ma ogni loro parola comincia per lettera maiuscola.

Enunciato `import`

La sintassi dell'enunciato `import` è:

```
<import_declaration> ::=
   import <top_pkg_name>[.<sub_pkg_name>]*.<classes>;

<classes> ::= <class_name> | *
```

- Precede ogni dichiarazione di classe.
- Dice al compilatore dove trovare le classi da usare.
- Importa solo lo spazio dei nomi, non le classi. Serve cioè solo ad abbreviare la scrittura del codice, permettendo di citare solo il nome di una classe e non obbligatoriamente tutto il percorso per raggiungerla.

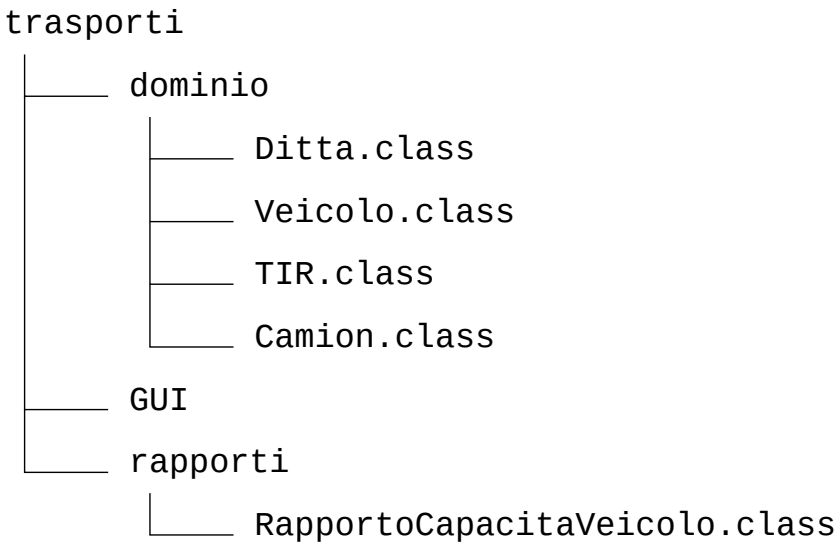
Esempio:

```
import trasporti.dominio.*;
import java.util.List;
import java.io.*;
```


Cartelle e pacchetti 35 / 41

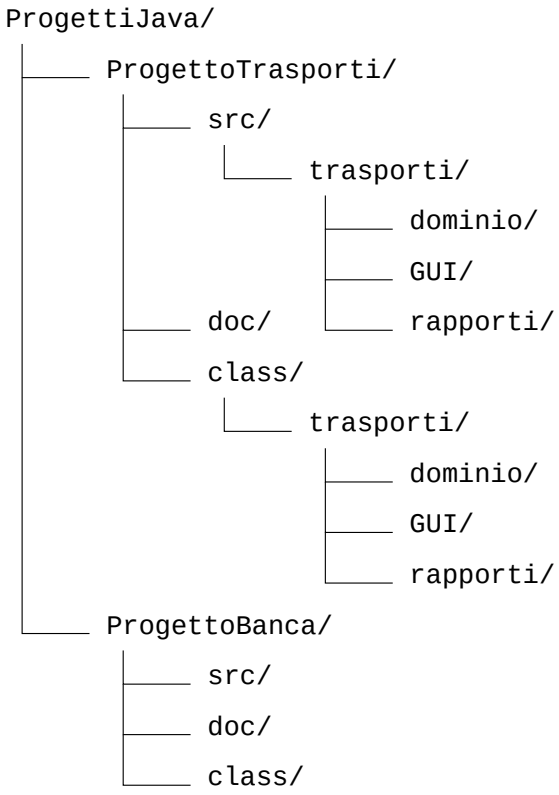
Organizzazione consigliata

- Pacchetti contenuti in alberi di cartelle.
- Nomi di cartelle uguali a nomi di pacchetto.
- Nell'esempio precedente:



Ordine!

È buona norma di programmazione quella di separare, nello sviluppo di un progetto, i file sorgente dai file compilati. Un metodo possibile è:



Come?

- Normalmente il compilatore pone i file `.class` nella stessa cartella dei file `.java`.
- I file `.class` possono essere re-indirizzati in un'altra cartella usando l'opzione `-d` del comando `javac`.
- Se si sta compilando un insieme di file all'interno della gerarchia dei pacchetti (cioè non nella cartella principale dei sorgenti), allora bisogna anche specificare l'opzione `-sourcepath`.

Per esempio:

```
$ cd ProgettiJava/ProgettoTrasporti/src/trasporti/dominio
$ javac -sourcepath ProgettiJava/ProgettoTrasporti/src \
        -d ProgettiJava/ProgettoTrasporti/class *.java
```

Fase di rilascio

Una applicazione può essere rilasciata su una macchina cliente in più modi:

1. Se si utilizza un file JAR, quel file deve essere copiato nella cartella delle "estensioni di libreria". Per esempio:
   `/usr/jdk1.4/jre/lib/ext/` Ambiente Linux
   `C:\jdk1.4\jre\lib\ext\` Ambiente Windows
2. Se l'applicazione è rilasciata come una gerarchia di file `.class`, allora bisogna porre tutta la gerarchia all'interno della cartella delle "classi JRE". Per esempio:
   `/usr/jdk1.4/jre/classes/` Ambiente Linux
   `C:\jdk1.4\jre\classes\` Ambiente Windows

Uso della documentazione

- Dimostrazione sull'utilizzo.
- Descrizione della gerarchia delle classi della API.

Esercizi

1. Uso della documentazione.
2. Realizzazione della incapsulazione.
3. Creazione di un semplice pacchetto.