Programmazione in Java
Parte I: Fondamenti

Lezione 1

Dott. Marco Faella

Testi consigliati:

- Progettazione del software e design pattern in Java
  di C.S. Horstmann
  Apogeo

  Gli esempi di codice si possono scaricare da: www.apogeonline.com

- Core Java 2, (7a edizione), volumi 1 e 2
  di C.S. Horstmann e Cornell
  Pearson/Prentice Hall

  Gli esempi di codice si possono scaricare da: www.phptr.com

Per installare Java:

- java.sun.com
- Scaricare:
  - JDK 6.0 (Java Development Kit: compilatore + JVM)
  - JDK 6.0 + NetBeans
  - Non è sufficiente JRE (Java Runtime Environment: solo la JVM)
- Editor: ultraedit, jEdit
- http://people.na.infn.it/mfaella

- Questo corso presuppone la conoscenza di almeno un linguaggio di programmazione procedurale, come il C, il Fortran, una delle varianti moderne del BASIC, il Pascal, etc.

- Ad esempio, bisogna conoscere i seguenti concetti

  - variabili ed espressioni
  - procedure/funzioni
  - istruzioni di controllo di flusso (almeno if-then-else e for)

- Orientato agli oggetti
- Relativamente recente: 1995
- Creato e controllato da Sun (ora Oracle)
- Ispirato alla sintassi C/C++, con numerose semplificazioni rispetto a quest'ultimo
- Attualmente molto utilizzato per applicazioni web lato server

- Ibrido tra compilato e interpretato
- Il compilatore crea il *bytecode* (linguaggio intermedio tra sorgente e macchina)
- La Java Virtual Machine (JVM) esegue (cioè, interpreta) il bytecode
- Si facilita la *portabilità*

- Sorgenti: .java
- Bytecode: .class
- Compilatore: javac
- JVM: java

HelloWorld.java

I tipi base di Java

- I tipi base sono nove: void, boolean, char, byte, short, int, long, float e double

- void è solo il tipo di ritorno delle procedure (metodi senza valore di ritorno)
- boolean può assumere solo i valori true e false
- char è un singolo carattere
- Le stringhe non sono un tipo base, ma una classe (String)

- byte, short e int sono tipi numerici interi, con segno
  - byte: 8 bit, da -128 a 127
  - short: 16 bit, da -32000 a 32000 (circa, precisamente: da -2^{15} a 2^{15} -1)
  - int: 32 bit, da -2 miliardi a 2 miliardi (circa, precisamente: da -2^{31} a 2^{31} -1)
  - long: 64 bit, da -2^{63} a 2^{63} -1

- float e double sono tipi numerici a virgola mobile
  - float: 32 bit
  - double: 64 bit

Le conversioni implicite

- Tra tipi base esistono le seguenti *conversioni implicite* (o promozioni):
  - Da byte a short, da short a int, da int a long, da long a float e da float a double
  - Da char a int

- Le conversioni implicite sono transitive
- Se esiste una conversione implicita dal tipo x al tipo y, è possibile assegnare un valore di tipo x ad una variabile di tipo y

- Alcune di queste conversioni possono comportare una *perdita di informazione*
  - Ad esempio:

```
int i = 1000000001; // un miliardo e uno
float f = i;
```

Dopo queste istruzioni, f contiene un miliardo, perchè un float non ha abbastanza bit di mantissa per rappresentare le 10 cifre significative di i.

1) Inserire una dichiarazione (e inizializzazione) per la variabile i, in modo che il seguente ciclo sia infinito.

```
while (i == i+1) {...}
```

2) Quante volte viene eseguito il seguente ciclo?

```
for (double x=0; x<=1 ;x+=0.1) {...}
```

- Una classe è uno "**stampo**" per creare oggetti
- Contiene **dati** e gli **algoritmi** che li elaborano

- I dati:
  - rappresentano lo stato dell'oggetto
  - sono variabili di cui ogni oggetto avrà una copia
  - vengono chiamate "**campi istanza**"

- Gli algoritmi:
  - sono le funzioni associate alla classe
  - vengono chiamate "**metodi istanza**"
  - possono accettare parametri e restituire un valore
  - vengono invocati a partire da un oggetto della classe

- Campi istanza e metodi istanza sono tutti chiamati ***membri*** della classe

- Un tipo particolare di metodo è chiamato ***costruttore***, e serve a creare nuove istanze di quella classe

- Sintatticamente, un costruttore si presenta come un metodo che:
  - ha lo stesso nome della classe
  - non ha tipo restituito

- Compito di un costruttore è di inizializzare lo stato (cioè, i campi istanza) dell'oggetto
- Viene invocato un costruttore ogni volta che si crea un nuovo oggetto, tramite il costrutto **new**

- Esempio:

```
String s = new String(“ciao”);
```

- Una classe può avere **più costruttori**
- Se una classe non ha costruttori, il compilatore aggiunge automaticamente un costruttore senza argomenti (chiamato **costruttore di default**), che non esegue alcuna operazione

- Esempio: Strings.java

Data1.java

Memory layout

- Si utilizzerà talvolta la nozione di “memory layout” di un programma
- Ci si riferisce ad una rappresentazione grafica dello stato della memoria ad un determinato punto di un programma
- Come primo esempio, dato il frammento di codice Java:

```
class A {
     // campi istanza
     private String x;
     private int n;

     // costruttore
     public A(String x, int n) {
          this.x = x;
          this.n = n;
     }
}
...
A a = new A(“ciao”, 7);
```

- il suo memory layout può essere raffigurato come in Figura 1

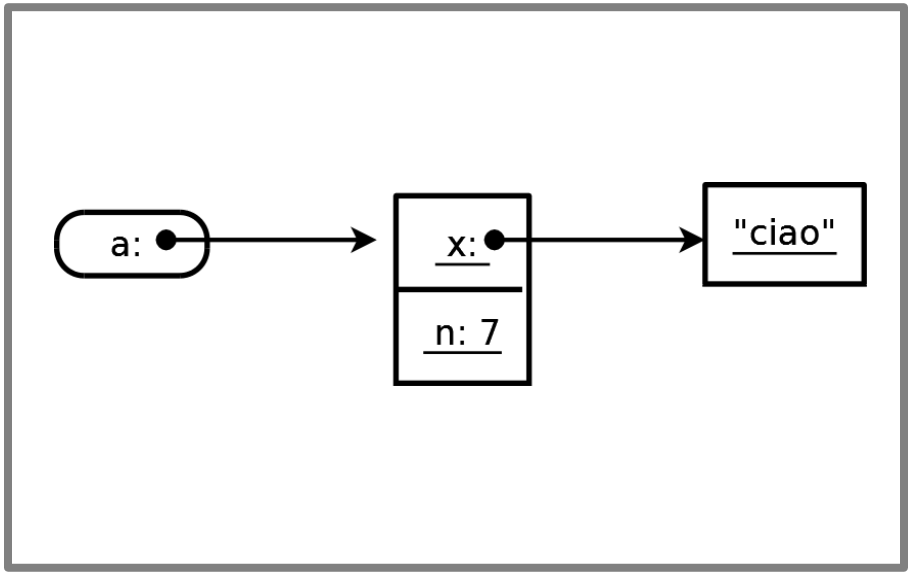


Figura 1: Il memory layout del frammento di programma a fianco.

Data2.java

Riferimenti e passaggio di argomenti

- A differenza di altri linguaggi orientati agli oggetti (ad es., il C++), non esistono variabili che contengono oggetti, solo *riferimenti* ad oggetti, ovvero variabili che contengono l'*indirizzo* di un oggetto

- Tali riferimenti Java sono quindi simili ai puntatori del linguaggio C
- Tuttavia, i riferimenti Java sono molto più restrittivi
    - Niente aritmetica dei puntatori (p++) e conversioni tra puntatori e numeri interi
    - Niente doppi puntatori
    - Niente puntatori a funzioni
- Quindi, rispetto ai puntatori, i riferimenti Java sono meno potenti, più facili da utilizzare e meno soggetti ad errori al run-time

- Java prevede solo il passaggio per *valore*: sia i tipi base che i riferimenti sono passati per valore
- Non è possibile passare oggetti per valore, l'unico modo di manipolare (ed in particolare, passare) oggetti è tramite i loro riferimenti (ovvero, indirizzi)

Parameters.java

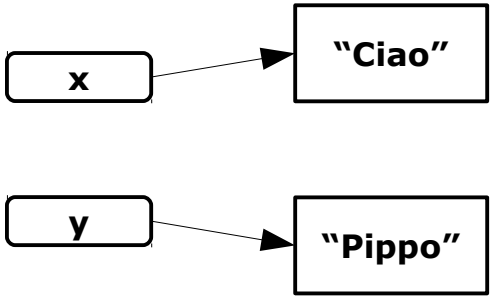
Stralcio di Parameters.java:

```
public static void swap_str(  String a,
                              String b)
{
     String tmp = a;
     a = b;
     b = tmp;
}

...

String x = “Ciao”, y = “Pippo”;
swap_str(x, y);
```

Memory layout:



(prima di invocare swap_str)

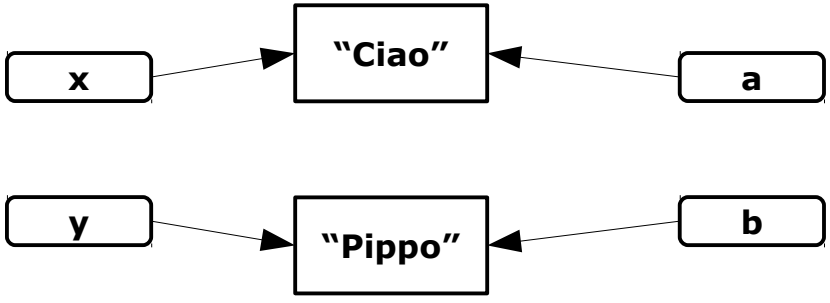
Stralcio di Parameters.java:

```
public static void swap_str(  String a,
                              String b)
{
      String tmp = a;
      a = b;
      b = tmp;
}

...

String x = “Ciao”, y = “Pippo”;
swap_str(x, y);
```

Memory layout:



(all'inizio di swap_str)

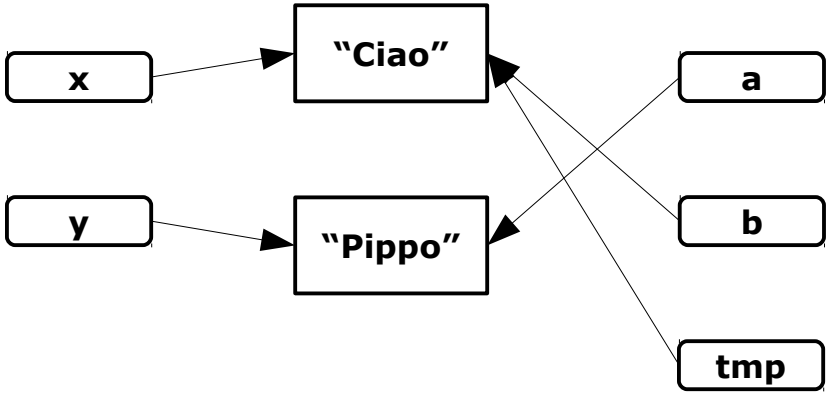
Stralcio di Parameters.java:

```
public static void swap_str(  String a,
                              String b)
{
     String tmp = a;
     a = b;
     b = tmp;
}

...

String x = “Ciao”, y = “Pippo”;
swap_str(x, y);
```

Memory layout:



(alla fine di swap_str)

1) Inserire una dichiarazione (e inizializzazione) per la variabile i, in modo che il seguente ciclo sia infinito.

```
while (i == i+1) {...}
```

Soluzione: float i = 1000000000; // un miliardo

Sommare 1 ad i non ne modifica il valore, in quanto un float non ha abbastanza cifre binarie nella sua mantissa per rappresentare il numero “un miliardo e uno”.

2) Quante volte viene eseguito il seguente ciclo?

```
for (double x=0; x!=1 ;x+=0.1) {...}
```

Soluzione: infinite volte.

La variabile x non assumerà mai esattamente il valore 1, perché 0.1 non è rappresentabile in maniera esatta con un double. Infatti, in rappresentazione binaria 0.1 è il numero periodico 0.0<u>0011</u>.

La morale di questi esercizi è che l'aritmetica in virgola mobile nasconde molte insidie. A tale proposito, si veda anche l'interessante disamina "How Java's Floating-Point Hurts Everyone Everywhere", facilmente rintracciabile sul web.