Marco Faella

Introduzione al corso

Lezione n. 1

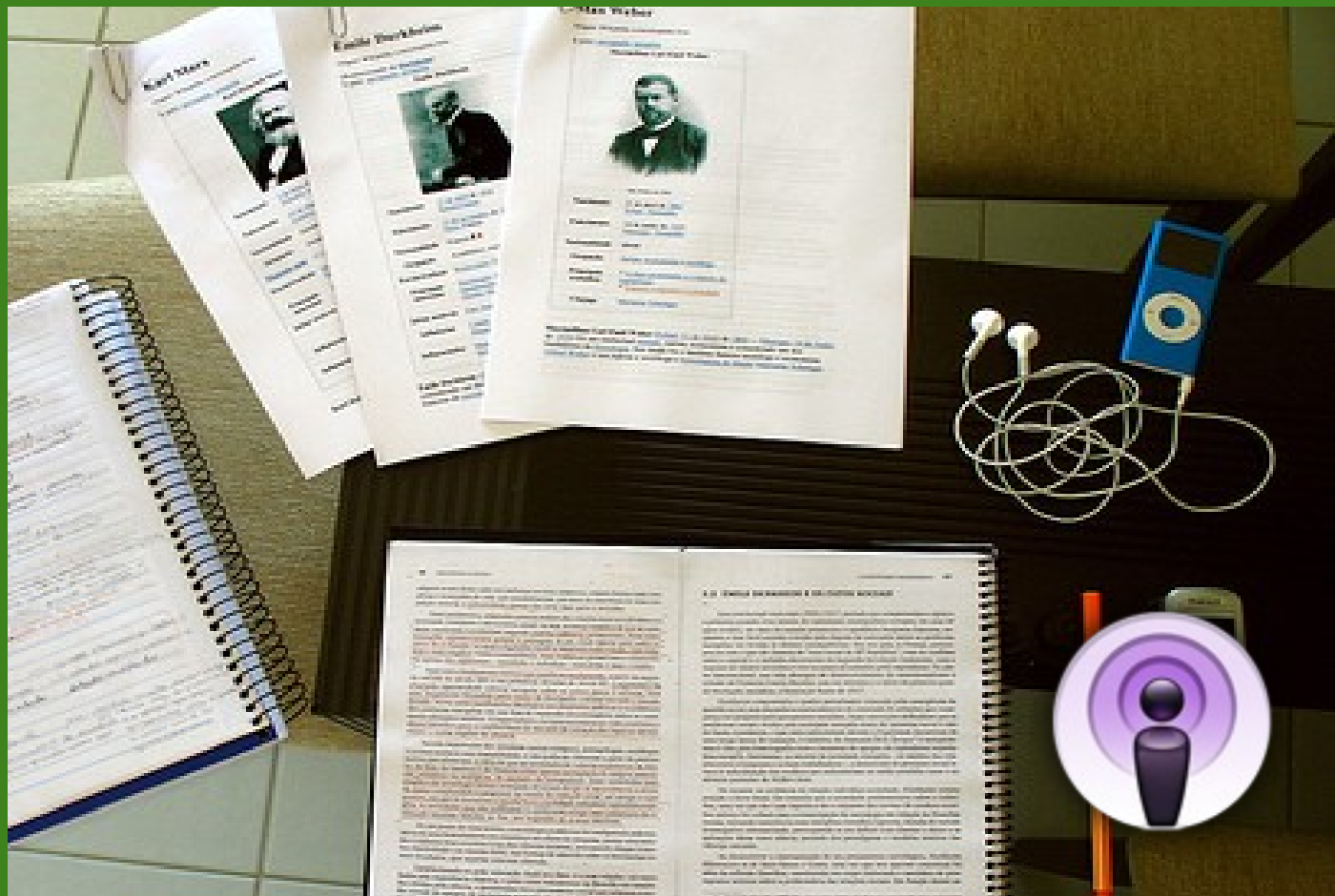
Parole chiave:
Java

Corso di Laurea:
Informatica

Insegnamento:
Linguaggi di Programmazione II

Email Docente:
faella.didattica@gmail.com

A.A. 2009-2010



Testi consigliati:

- Core Java, (10ª edizione, Java 8), volumi 1 e 2
  di Cay Horstmann
  Pearson/Prentice Hall

  oppure

- Java Precisely (3ª edizione, Java 8)
  di Peter Sestoft
  MIT Press

Inoltre:

- Seriously Good Software
  di M.F.
  Manning

Per l'approfondimento:

- Linguaggi di programmazione, principi e paradigmi
  di M. Gabbrielli e S. Martini
  McGraw-Hill

  Il capitolo 10, in particolare, offre un'introduzione rigorosa ai tipi parametrici (anche detti *generics*)

- Questo corso presuppone la conoscenza dei seguenti argomenti di Java elementare:
  - Tipi base
  - Operatori ed espressioni
  - Istruzioni di controllo di flusso (if-then-else, for, while, break, continue)
  - Classi
    - Costruttori, metodi, campi
  - Modificatori di visibilità
  - Modificatori static e final
  - Interfacce
  - Ereditarietà
  - Classi astratte
  - Eccezioni

Ripasso: tipi base e conversioni implicite

- I tipi base sono otto: boolean, char, byte, short, int, long, float e double
  - Inoltre, void è un tipo speciale usato solo come tipo di ritorno dai metodi

- 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, perché un float non ha abbastanza bit di mantissa per rappresentare le 10 cifre significative di i.

Ripasso: 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

- I 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)

Esempio: passaggio di argomenti

- Data la seguente classe, commentarne l'output

```
class Test {
    public static void swap(Object a, Object b) {
            Object tmp = a;
            a = b;
            b = tmp;
    }
    public static void swap(int a, int b) {
            int tmp = a;
            a = b;
            b = tmp;
    }
    public static void main(String args[]) {
            String x = "ciao", y = "Pippo";
            swap(x, y);
            S.o.println(x);  S.o.println(y);

            int i = 3, j = 7;
            swap(i, j);
            S.o.println(i);  S.o.println(j);
    }
}
```

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

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

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;`**

Sommare 1 ad i non ne modifica il valore, in quanto, come illustrato precedentemente, un float non ha abbastanza cifre binarie nella mantissa per rappresentare il numero "un miliardo e uno".

Nota:

float:	1 bit segno + 23 bit mantissa + 8 bit esponente	= 32 bit	(~7 cifre decimali significative)
double:	1 bit segno + 52 bit mantissa + 11 bit esponente	= 64 bit	(~16 cifre decimali significative)

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!=1.0; x+=0.1) {...}
```

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.

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 {
        private String x;
        private int n;
        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
- I diagrammi di memory layout di queste slide non evidenziano la differenza tra allocazione su stack e su heap

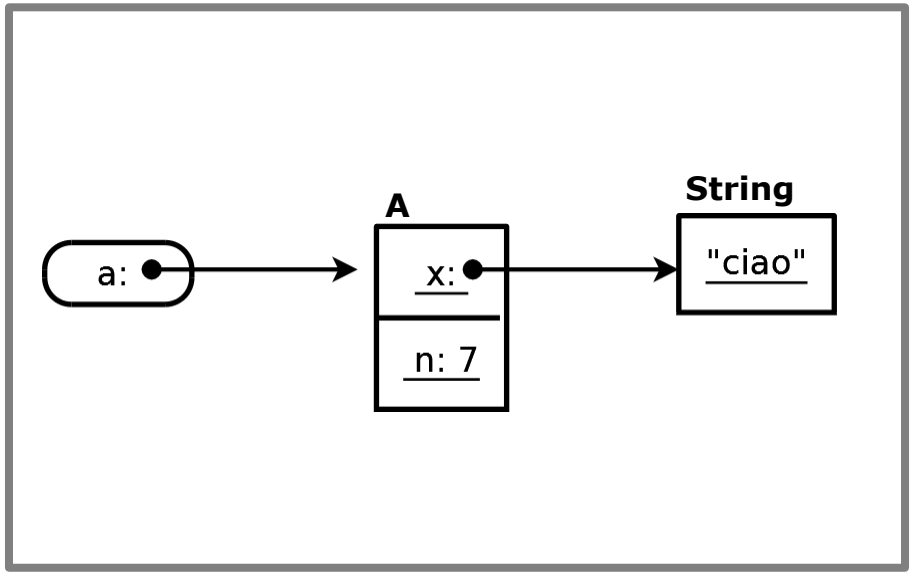


Figura 1: Il memory layout del frammento di programma a fianco.

Memory layout degli array

- In Java, gli array sono oggetti a tutti gli effetti
- In particolare, sono sottotipi di Object e ne ereditano i metodi
- Non esistono variabili che contengono (direttamente) array, ma solo riferimenti ad array

- Ad esempio, in figura viene riportato il memory layout del seguente frammento di programma:

```
int[] a = new int[5];
int[] b = a;
int[][] c = new int[3][];
c[1] = b;
c[2] = new int[7];
```

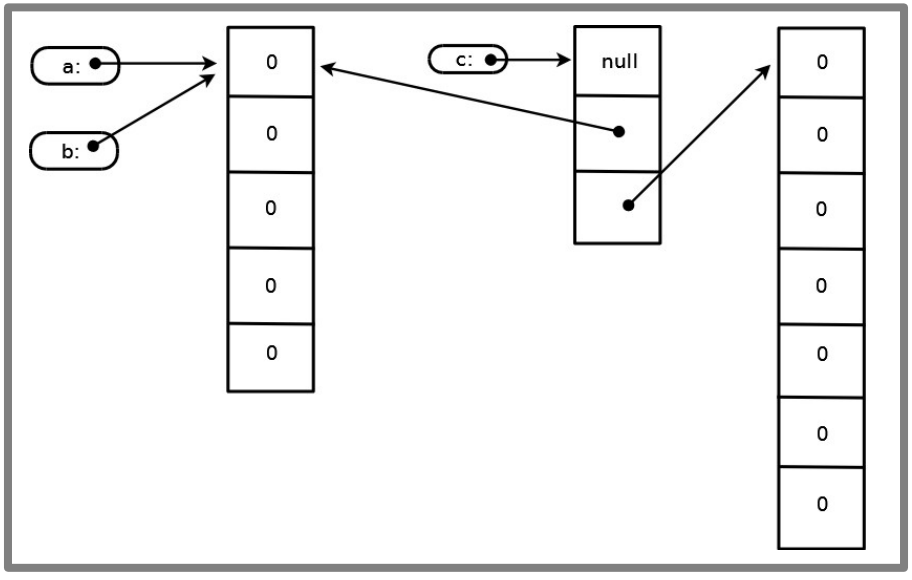


Figura 2: Il memory layout del frammento di programma a fianco.