

Università degli studi di Ferrara
Corso di laurea in Informatica
a.a. 2010-2011

Tutorato di Linguaggi 2 e Laboratorio

Dr. Giacomo Spettoli

(24/01/2011)



Nelle puntate precedenti...

- Definizione di OOP (Object Oriented Programming) e sue caratteristiche distintive
- Esempi di Linguaggi ad oggetti più utilizzati
- Perché proprio Java?
- Installazione jdk e ide eclipse sui vari OS
- Cos'è un oggetto e cos'è una classe
- Esercizio: la classe “Studente”.



...In questa puntata

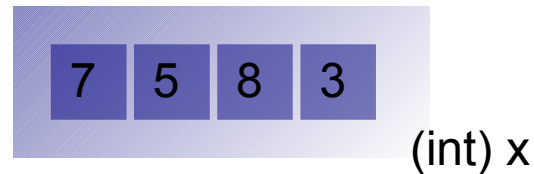
- Variabili **referimento**
- Passaggio di parametri: per valore o per riferimento?
- Costrutti **condizionali** e **cicli**
- **Scope** delle variabili
- Modificatori di accesso: **private** e **public**
- **Incapsulamento**
- **Overloading**



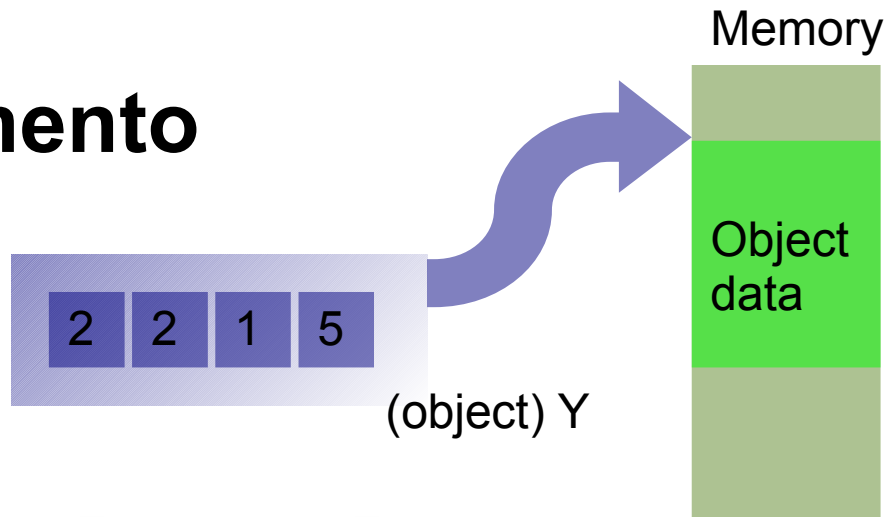
Variabili riferimento

In Java ci sono due tipi di variabili:

- variabili di tipo primitivo (int, double, float...)



- variabili **riferimento**



Passaggio di parametri

Quindi il passaggio di parametri tra i metodi
come avviene?

- **per valore** se i parametri sono tipi primitivi
- **per riferimento** se i parametri sono oggetti



Cicli e Costrutti condizionali (1/5)

- Il costrutto if-then-else (come il C):

```
if (<condizione>) {  
    <...codice...>  
}  
else if (<condizione>) {  
    <...codice...>  
}  
else {  
    <...codice...>  
}
```



Cicli e Costrutti condizionali (2/5)

- Il costrutto **switch** (come il C):

```
switch(<variabile di controllo>) {  
    case X:<codice>; break;  
    case Y:<codice>; break;  
    default:<codice>;  
}
```



Esempio

```
1  int a = 1;
2  switch (a) {
3      case 1:
4          System.out.println("A");
5          break;
6      case 2:
7          System.out.println("B");
8          break;
9      default: System.out.println("C");
10 }
```



Cicli e Costrutti condizionali (3/5)

- Ciclo **for** (stesso comportamento del C):

for(assegnazione variabile; limite massimo; incremento) {...codice...}

```
1  int a = 10;  
2  for(int i=0; i<a; i++) {  
3    System.out.println(i+a);  
4  }
```



Cicli e Costrutti condizionali (4/5)

- Ciclo **while** e **do-while** (stesso comportamento del C):

while (condizione) { ... codice ... }

do { ... codice ... } while (condizione)

```
1  int a = 1;  
2  while (a < 10) {  
3    System.out.println(a);  
4    a += 1;  
5  }
```



Cicli e Costrutti condizionali (5/5)

- Ciclo **for-each** (novità!):

```
for (type var: collezione)  
{...codice...}
```

A partire dalla j2se v.5 è disponibile questo ciclo che permette di scorrere una collezione (es. un array)

```
1  int myArray[] = {1, 3, 5};  
2  for (int x:myArray)  
3      System.out.println(x);  
4  }
```



Scope delle variabili

In Java (ma più in generale nei linguaggi ad oggetti) ci due sono ambiti di visibilità di una variabile:

- blocco o **metodo**  **variabili locali**
- file o **classe**  **variabili globali**



...ad esempio:

```
1 public class Esempio{
2     int a = 5;
3     ...
4     public void metodo1 () {
5         System.out.println(a); // 5
6     }
7     public void metodo2 () {
8         int a = 3;
9         System.out.println(a); // 3!
10        System.out.println(this.a); // 5!
11    }
12 }
```

La keyword **this** si può omettere, ma solo quando non c'è **ambiguità**!



Modificatori di accesso

Fino ad ora tutti i metodi potevano accedere a tutte le variabili di classe e tutti i metodi potevano essere invocati ovunque.



Quindi un oggetto poteva modificare gli attributi di un'altro oggetto senza controllo



Nessuna sicurezza e consistenza



Modificatori di accesso (2)

Per risolvere questo problema nei linguaggi ad oggetti è possibile impedire l'accesso (lettura/scrittura) agli attributi con i **modificatori di accesso.**



Modificatori di accesso (3)

- **public**: tutti possono accedere al campo/metodo
- **protected**: possono accedere al campo/metodo solo le classi dello stesso package e le sottoclassi (classi che estendono la classe corrente).
- **<nessun modificatore>**: possono accedere al campo/metodo solo le classi dello stesso package e le sottoclassi dello stesso package.



Modificatori di accesso (4)

- **private**: il campo/metodo è accessibile solo all'interno della classe dove è definito.

```
1 public class Esempio{
2     private int x;
3 }
4
5 public class Prova{
6     public static void main(String[] args){
7         Esempio e = new Esempio();
8         e.x = 1          // ERROR!
9         System.out.println(e.x)  // ERROR!
10    }
11 }
```



Incapsulamento

- Bello! Ma ora come faccio ad accedere le variabili del oggetto?



Metodi setter e getter



Quindi impedisco di accedere direttamente ai campi di un oggetto ma li rendo disponibili in modo controllato con i metodi



Incapsulamento (esempio)

```
1 class Eempio {
2     private int x;
3     private int numModifiche = 0;
4     public getX() {
5         return x;
6     }
7     public setX(int nuovoValore) {
8         x = nuovoValore;
9         numModifiche += 1;
10    }
11 }
12 class Prova {
13     public static void main(String args[]) {
14         Eempio e;
15         e=new Eempio();
16         e.setX(12); // ok!
17         System.out.println(e.getX()); // ok!
18     }
19 }
```



Overloading

- L'overloading (o sovraccarico) è una delle caratteristiche più interessanti del Java.
- Permette di definire più metodi (anche e soprattutto costruttori) con lo stesso nome.



Overloading

- Questo meccanismo permette quindi di avere un unico nome per la “funzionalità”:

```
1 public int somma (int a, int b) {  
2     return a+b;  
3 }  
4  
5 public int somma (int a, int b, int c) {  
6     return a+b+c;  
7 }
```



Overloading

Ok, ma come fa la jvm a sapere quale usare?



In Java ogni classe identifica i suoi metodi attraverso le signature (firme)

public void metodo(type arg)



Quindi per far funzionare il tutto, i metodi devono avere una firma diversa, in particolare i parametri (il tipo e/o il numero).



Esercizio

Riscrivere la classe `Studente` in modo che:

- Abbia un attributo `matricola`
- Tutti i campi siano incapsulati e aggiungere i metodi setter/getter.
- Ci siano almeno 2 costruttori (eventualmente in cross-call)

Scrivere una classe `Università` che:

- Abbia un attributo static *`ultimaMatricola`* che viene incrementato ad ogni nuova iscrizione
- abbia un metodo ***`static Studente assegnaMatricola(Studente stud1)`*** che accetta come parametro uno `Studente` e gli assegna una matricola. La nuova matricola è generata con *`ultimaMatricola`* incrementata di 1.

