Programmazione Java

Struttura di una classe, Costruttore, Riferimento this

Davide Di Ruscio
Dipartimento di Informatica
Università degli Studi dell'Aquila

diruscio@di.univaq.it

Sommario

» Struttura di una classe

» Costruttore

» Riferimento `this`

» Inizializzazione di dati static

Classe

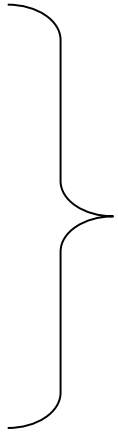

```
[ public | abstract | final ]
            class <nome della classe>
                  [extends Tipo]
                  [implements ListaTipi] {


[<dichiarazione di attributi>]

[<dichiarazione dei costruttori>]

[<dichiarazione dei metodi>]


}
```



Corpo della classe

Struttura di una classe (2)

Esempio

```
public class Point {
   private int x, y;
   public Point(int dx, int dy) {
      x = dx;
      y = dy;
   }
   public void move(int dx, int dy) {
      x += dx;
      y += dy;
   }
……
}
```

Struttura di una classe (3)

Attributi

```
<modificatore> <tipo> <identificatore> [ = <inizializzazione> ]
```

» Esempio

```
public class Point {
  private int x;
  private int y = 100;


}
```

Struttura di una classe (4)

Metodo

```
<modificatore> <tipo ritorno> <identificatore> ([lista parametri]) {

  [<istruzioni>]

}
```

» Esempio

```
public class Point {
   private int x, y;
   public void move(int dx, int dy) {
      x += dx;
      y += dy;
   }
}
```

Struttura di una classe (5)

Costruttore

```
<modificatore> <nome della classe> ([<lista parametri>]) {

[<istruzioni>]

}
```

» Esempio

```
public class Persona {
  private String nome;
   public Persona(String n) {
        nome = n;
   }
}
```

Struttura di una classe (5)

Costruttore

```
<modificatore> <nome della classe> ([<lista parametri>]) {

[<istruzioni>]

}
```

E' possibile ometterlo in tal caso verrà considerato il modificatore `package`

» Esempio

```
public class Persona {
  private String nome;
   public Persona(String n) {
        nome = n;
   }
}
```

Struttura di una classe (6)

Accesso ai membri di un oggetto

- `<oggetto>.<membro>`
- Membro può essere sia un attributo che un metodo
- Esempi

```
p.move(50,50);
p.x = 100;
```

Inizializzazione ed eliminazione

- In linguaggi come il C molti errori si verificano quando il programmatore dimentica di inizializzare una variabile
- L'eliminazione è un altro problema, è facile dimenticarsi di un elemento quando si smette di utilizzarlo
  - Le risorse utilizzate da quell'elemento vengono trattenute ed è facile che si esauriscano
- C++ ha introdotto il concetto di *costruttore*, un metodo speciale che viene automaticamente chiamato quando si crea un oggetto
- Anche Java ha adottato il costruttore, in più ha un garbage collector per gestire l'eliminazione degli oggetti non più utilizzati

Overloading

» Si parla di overloading quando una classe può contenere due o più metodi (statici e non) con
  – Stesso nome
  – Diversa segnatura
    • Tipo diverso in almeno un parametro se hanno lo stesso numero
    • Numero di parametri diversi
  – Il tipo di ritorno non è considerato

» Esempio

```
public void print(int i)
public void print(float f)
public void print(String s)
public void print(String s, int x)
public void print(int x, String s)
```

Overloading: distinguere i metodi ridefiniti

- Ciascun metodo ridefinito deve avere una lista univoca di tipi degli argomenti
  - Persino le differenze nell'ordine degli argomenti sono sufficienti per distinguere fra due metodi (anche se normalmente si preferisce non seguire questo approccio, perche genera codice difficile da gestire)

- Esempio
  - (vedere OverloadingOrder.java)

Overloading: ridefinizione con tipi di dato primitivi

- Un tipo di dato primitivo può essere automaticamente promosso da un tipo minore a uno maggiore
- Cosa accade quando un tipo di dato primitivo viene passato a un metodo ridefinito ?
  - vedere PrimitiveOverloading.java
  - Nell'esempio il valore costante 5 viene trattato come un int, per cui se è disponibile un metodo ridefinito che richiede un parametro di tipo int esso viene utilizzato
  - In tutti gli altri casi, se si ha un tipo di dato che è più piccolo del tipo dell'argomento del metodo, quel tipo di dato viene promosso
  - Char produce un effetto leggermente diverso, se non trova un'esatta corrispondenza, viene promosso a int

Overloading: ridefinizione con tipi di dato primitivi

» Cosa accade se il tipo dell'argomento è più grande del tipo dell'argomento previsto dal metodo ridefinito ?

- vedere Demotion.java
- Occorre effettuare un cast nel tipo necessario utilizzando il tipo tra parentesi, altrimenti il compilatore genererà un messaggio di errore

Costruttore (1)

» Viene utilizzato per creare un oggetto

» Sintassi

```
[ public |  private | protected ]  nomeClasse(lista parametri) {
  body
}
```

» Viene fornito un costruttore di default qualora non se ne dichiari uno
  – Non prende argomenti e non ha corpo
  – Se è presente un costruttore quello di default scompare

» E' possibile effettuare l'overloading anche dei costruttori

» Esempio

```
public class Point {
  public Point(int x, int y) {
     …..
  }
}
```

(vedere DefaultConstructor.java, Overloading.java)

Costruttore (2)


```
Point origin_one = new Point(23, 94);
Rectangle rect_one = new Rectangle(origin_one, 100, 200);
Rectangle rect_two = new Rectangle(50, 100);
```

- **Dichiarazione**
  - E' necessario dichiarare una variabile con un tipo che rappresenta l'oggetto
- **Istanziazione**
  - Operatore `new` crea un nuovo oggetto
  - Viene allocato nell'heap dello spazio per contenere l'oggetto
  - Se non c'è spazio sufficiente viene lanciata eccezione `OutOfMemoryError`
  - Tutte le variabili di istanza vengono inizializzate al loro valore di default

Costruttore (3)

Inizializzazione

1. Vengono assegnati i valori ai parametri formali del costruttore (se ce ne sono)
2. Vengono eseguiti le inizializzazioni esplicite delle variabili di istanza come appaiono nel codice sorgente
3. Vengono eseguiti gli inizializzatori di istanza
4. Viene eseguito il corpo del costruttore
   - Se il corpo inizia con l'invocazione di un altro costruttore (`this`) si ritorna al passo 1
   - Altrimenti viene invocato il costruttore (ripetendo i passi) del padre (`super`)

(vedi ConstructorOrder.java, Flower.java, ConstructorOrder2.java, ConstructorOrder3.java)

Costruttore (4)

- Valore di default per le variabili di istanza (valido anche per quelle di classe)
  - byte: `(byte)0`
  - short: `(short)0`
  - int: `0`
  - float: `0F`
  - double: `0`
  - char: `'\u0000'`
  - boolean: `false`
  - reference: `null`

Costruttore (5)


```
class Point {
   int x = 100;
   int y = 100;
   {
     x = 200;
     y = 200;
   }
   Point() {
     x = 300;
     y = 300;
   }
   Point(int dx, int dy) {
     this();
     x = dx;
     y = dy;
   }
}
```

Costruttore (6)


```
class ColoredPoint extends Point {
    int color = 0xFF00FF;

}

class ConstructorPoint{
      public static void main(String[] args) {
         ColoredPoint cp = new ColoredPoint();
         System.out.println(cp.color);
      }
}
```

(vedi ConstructorPoint.java)

Ordine di inizializzazione

- All'interno di una classe, l'ordine di inizializzazione è determinato dall'ordine in cui le variabili sono definite nella classe
- Le definizioni delle variabili possono essere sparpagliate tra le definizioni dei metodi, ma vengono sempre inizializzate prima che venga chiamato un metodo – perfino il costruttore

(Vedere OrderOfInitialization.java)

Riferimento this (1)

- Indica il riferimento all'oggetto stesso
- Viene utilizzato nei seguenti ambiti
  - Per invocare all'interno di un costruttore un altro
  - All'interno di metodi e/o costruttori per riferirsi a variabili di istanza e/o metodi

(vedere Leaf.java)

Riferimento this (2)


```
public class Point {
  int x, y;
   public Point(int x, int y) {
        this.x = x;
        this.y = y;
  }
  public Point(Point p) {
      this.x = p.x;
      this.y = p.y;
  }

  public void move(int x, int y) {
     this.x = x;
     this.y = y;
  }
  public Point clonePoint() {
     return new Point(this);
  }
}
```

Riferimento this (3)


```
public class Test {
  public static void main(String[] args) {
      Point p = new Point(100, 100);
      Point p1 = p.clonePoint();
      p.move(200, 200);
      System.out.println("p.x: " + p.x);
      System.out.println("p.y: " + p.y);
      System.out.println("p1.x: " + p1.x);
      System.out.println("p1.y: " + p1.y);

   }
}
```

(Vedere TestPoint.java)

this and static

- Non si può usare this per invocare un particolare metodo statico
- Non si possono chiamare metodi non static all'interno di metodi static (anche se si può fare il contrario)
- Con un metodo static non si invia un messaggio a un oggetto, poichè non c'e' this

(Vedere Static.java)

Esercizio

» Date le classi

```
class Point {
 int x; int y;
 Point (int x1, int y1) {
    x = x1; y = y1;
 }
 double distanceToOrigin() {
    return sqrt(x * x + y * y);
 }
}
```

```
class Line {
   Point pt1;
   Point pt2;
   Line (Point p, Point q){
      pt1 = p;
      pt2 = q; }
}
```

scrivere per Point e Line un metodo equals tale che pt1.equals(pt2) ritorna true se i due punti coincidono, l1.equals(l2) ritorna true se le due linee coincidono