I costruttori

Servono a inizializzare i valori dei campi degli oggetti.

Vengono invocati automaticamente dopo che l'oggetto è stato creato.

Il costruttore standard

Questo tipo di costruzione mette dei valori iniziali nelle componenti:

scalari:
: il valore zero

oggetti:
: il valore `null`

Esempio di valori di default

Classe:

```
import java.awt.*;

class NomeClasse {
  Point p;
  int x;
}
```

Programma:

```
class ProvaCostr {
  public static void main(String args[]) {
    NomeClasse a;
    a=new NomeClasse();

    System.out.println(a.p);
    System.out.println(a.x);
  }
}
```

Stampa:

```
null
0
```

Alterare il costruttore standard

È possibile ridefinire il comportamento del costruttore.

```
import java.awt.*;

class NomeClasse {
  Point p;
  int x;

  NomeClasse() {
```

```
    this.p=new Point();
    this.p.move(10,20);

    this.x=100;
  }
}
```

Il programma

Uso lo stesso programma di prima:

```
class ProvaCostr {
  public static void main(String args[]) {
    NomeClasse a;
    a=new NomeClasse();

    System.out.println(a.p);
    System.out.println(a.x);
  }
}
```

Solo che ora stampa:

```
java.awt.Point[x=10,y=20]
100
```

Sintassi del costruttore

```
class NomeClasse {
  ...

  NomeClasse() {
    istruzioni;
  }
}
```

- è un metodo
- ha lo stesso nome della classe
- non ha valore di ritorno (è implicito)

È un metodo statico?

Si e no.

si: non richiede oggetto di invocazione (lo crea!)
no: al suo interno, `this` è definito

Cosa succede quando si crea un oggetto

```
  a=new NomeClasse();
```

Vengono fatte due cose:

- viene creata la zona di memoria per l'oggetto
- vengono eseguite le istruzioni del metodo `NomeClasse()` della classe, se c'è

Viene poi ritornato l'indirizzo della zona di memoria creata.

I costruttori di `Point`

Questi costruttori sono già definiti:

```
p=new Point();
```

Crea un oggetto punto di coordinate `(0,0)`

```
p=new Point(12,32);
```

Crea un oggetto punto di coordinate `(12,32)`

Come tutti i metodi, i costruttori possono essere sovraccarichi (più costruttori con argomenti diversi).

Differenza metodo-costruttore

Quando si invoca un metodo, viene fatta la copiatura dei parametri e poi si eseguono le istruzioni del metodo.

Nel caso dei costruttori, viene *prima* creato l'oggetto, e *poi* si esegue il costruttore come fosse un metodo.

C'è un passo in mezzo, in cui l'oggetto viene creato.

Di solito...

Il costruttore si usa per inizializzare le componenti degli oggetti.

Però può fare qualsiasi cosa fa un metodo.

Per esempio, può stampare una stringa:

```
class Abcd {
  int x;
  Point p;

  Abcd() {
    System.out.println("Questa e' una stringa");
  }
}
```

È sbagliato *metodologicamente*, ma si può fare.

Il costruttore vuoto

Se una classe non ha costruttori, allora per essa viene automaticamente definito il costruttore vuoto.

Se in una classe ci metto un costruttore, allora quello vuoto non viene aggiunto automaticamente.

Se mi serve il costruttore vuoto, lo devo definire esplicitamente, in questo caso.

```
class Esempio1 {
  int x;
}
```

Si può fare `new Esempio1();`

```
class Esempio2 {
  int x;

  Esempio2(int a) {
    this.x=a;
  }
}
```

Si può fare `new Esempio2(valore)`, ma *non* si può fare `new Esempio2()`

```
class Esempio3 {
  int x;

  Esempio3() {
  }

  Esempio3(int a) {
    this.x=a;
  }
}
```

Si possono fare tutte e due le cose.

Regoletta

Fare cosí:

- non definire costruttori (equivale a definire solo il costruttore vuoto)
- definire tutti i costruttori che servono, incluso quello vuoto

---

Argomenti dei costruttori

Di solito: gli argomenti sono i valori che vengono messi nelle componenti dell'oggetto creato (es. il costruttore di `Studente` qui sopra)

Qualche volta: sono dati che permettono di calcolare i valori iniziali dei campi dato

In generale: il costruttore è come tutti gli altri metodi (può fare quello che vuole con gli argomenti)

Ereditare i costruttori

I costruttori non si ereditano.

Si possono *riusare*

Invocare un costruttore della sovraclasse: `super`

Riuso di costruttori

```
class Studente {
  Studente(String nome) {
    this.nome=nome;
}

class Borsista extends Studente {
  Borsista(String nome, int stipendio) {
    super(nome);
    this.stipendio=stipendio;
  }
}
```

L'invocazione `super(argomenti)` equivale a invocare il costruttore della sovraclasse che ha questi argomenti

`super(argomenti)` deve essere la prima istruzione del costruttore di `Borsista`

Se la prima istruzione di un costruttore non è `super(argomenti)`, si assume automaticamente che sia `super()`

Esempi

```
class Studente {
  String nome;
  int anno;

  Studente() {
    this.nome="nessuno";
  }
}


class Borsista extends Studente {
}
```

Cosa stampa questo programma?

```
  public static void main(String args[]) {
    Borsista s=new Borsista();

    System.out.println(s.nome);
  }
```

Risposta

La class `Borsista` non ha costruttori espliciti

Quindi ha il costruttore implicito:

```
  Borsista() {
  }
```

Implicitamente, la prima istruzione di un costruttore è `super()`:

```
  Borsista() {
    super();
  }
```

Assenza del costruttore standard

```
class Studente {
  String nome;
  int anno;

  Studente(String nome) {
    this.nome=nome;
  }
}


class Borsista extends Studente {
}
```

Viene dato errore:

- la classe `Studente` ha un costruttore definito esplicitamente;
- ma non ha la definizione esplicita del costruttore standard;
- quindi, non ha il costruttore standard;
- `Borsista` non ha costruttori espliciti, quindi ha il costruttore

  ```
  Borsista() {
    super();
  }
  ```

- ma qui `super` sarebbe il costruttore standard di `Studente`, che non esiste

Viene quindi dato errore.