

Esempi al calcolatore su:
1)Costruttori ed ereditarietà
2)Subtyping

Esempio 1: Costruttori ed ereditarietà

Introduzione

- Java prevede due automatismi legati ai costruttori:
 - Se una classe non ha costruttori viene creato automaticamente il **costruttore di default** (quello senza parametri)
 - Se in un costruttore di una classe derivata non viene invocato il costruttore della classe base, viene automaticamente inserita la **chiamata a super()**
- Questi due meccanismi sono molto comodi ma bisogna fare attenzione perché si possono creare situazioni di errore non facili da comprendere
- **Sviluppiamo un esempio che ha lo scopo di chiarire questi aspetti**

Esempio 1 – prima versione

- **Counter**
 - con attributo `val` intero `protected` e metodi pubblici `reset()`, `inc()`, `getValue()`
 - con costruttore di default definito che inizializza a 1 l'attributo `val` e costruttore con un parametro intero `n` che inizializza `val` a `n` (anche un po' di stampe ...)
- **BiCounter** estende **Counter**
 - con un metodo pubblico `dec()`
 - **senza alcun costruttore**
- **Main**
 - con un metodo pubblico statico `main()`
 - Definisce `Counter c` e lo crea
 - Definisce `BiCounter c1` e lo crea

Prima versione

```
public class Counter
{
    protected int val;
    public Counter()
    {
        System.out.println("Counter:
        costruttore default!");
        val = 1;
    }
    public Counter(int v)
    {
        System.out.println(
            "Counter:costruttore");
        val = v;
    }
    public void reset()
    { val = 0; }
    public void inc()
    { val++; }
    public int getValue()
    { return val;}
}
```

```
public class BiCounter
    extends Counter
{
    public void dec()
    {
        val--;
    }
}
```

Non definiamo un
costruttore per
BiCounter

```
public class Main
{
    public static void
        main(String Args[])
    {
        Counter c =
            new Counter();
        BiCounter c1 =
            new BiCounter();
    }
}
```

Risultato

```
C:\Program Files\Xinox Software\JCreator LE\GE2001.exe  
Counter : costruttore di default!  
Counter : costruttore di default!  
Press any key to continue...
```

Creazione istanza di Counter

Creazione istanza di BiCounter

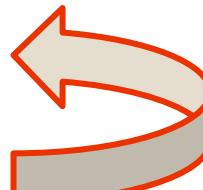
- Il compilatore Java ha aggiunto un costruttore di default in BiCounter e ha inserito in esso una chiamata a `super()`

Prima versione – come se fosse:

```
public class Counter
{
    protected int val;
    public Counter()
    {
        System.out.println("Counter:
        costruttore default!");
        val = 1;
    }
    public Counter(int v)
    {
        System.out.println(
            "Counter:costruttore");
        val = v;
    }
    public void reset()
    { val = 0; }
    public void inc()
    { val++; }
    public int getValue()
    { return val;}
}
```

```
public class BiCounter
    extends Counter
{
    public BiCounter()
    { super(); }

    public void dec()
    {
        val--;
    }
}
```



Non definiamo un
costruttore per
BiCounter

```
public class Main
{
    public static void
        main(String Args[])
    {
        Counter c =
            new Counter();
        BiCounter c1 =
            new BiCounter();
    }
}
```

Esempio 1 – seconda versione

- **Counter**
 - *invariato*
- **BiCounter estende Counter**
 - con un metodo pubblico `dec()` (*come prima*)
 - **definisce un costruttore che inizializza val a 1 (e fa un po' di stampe ...)**
- **Main**
 - *invariato*

Seconda versione

```
public class Counter
{
    protected int val;
    public Counter()
    {
        System.out.println("Counter:
        costruttore default!");
        val = 1;
    }
    public Counter(int v)
    {
        System.out.println(
            "Counter:costruttore");
        val = v;
    }
    public void reset()
    { val = 0; }
    public void inc()
    { val++; }
    public int getValue()
    { return val;}
}
```

```
public class BiCounter
    extends Counter
{
    public BiCounter()
    {   System.out.println("Counter2:
        costruttore di default!");
        val = 1;
    }
    public void dec()
    { val--; }
}
```

```
public class Main
{
    public static void
        main(String Args[])
    {
        Counter c =
            new Counter();
        BiCounter c1 =
            new BiCounter();
    }
}
```

Risultato

```
C:\Program Files\Xinox Software\JCreator LE\GE2001.exe
Counter : costruttore di default!
Counter : costruttore di default!
Counter2 : costruttore di default!
Press any key to continue...
```

Creazione istanza di Counter

Creazione istanza di BiCounter

- Il compilatore ha aggiunto una chiamata a `super()` nel costruttore di `BiCounter`

Seconda versione – come se fosse:

```
public class Counter
{
    protected int val;
    public Counter()
    {
        System.out.println("Counter:
        costruttore default!");
        val = 1;
    }
    public Counter(int v)
    {
        System.out.println(
            "Counter:costruttore");
        val = v;
    }
    public void reset()
    { val = 0; }
    public void inc()
    { val++; }
    public int getValue()
    { return val;}
}
```

```
public class BiCounter
    extends Counter
{
    public BiCounter()
    {    super();
        System.out.println("Counter2:
        costruttore di default!");
        val = 1;
    }
    public void dec()
    { val--; }
}
```

```
public class Main
{
    public static void
        main(String Args[])
    {
        Counter c =
            new Counter();
        BiCounter c1 =
            new BiCounter();
    }
}
```

Esempio 1 – terza versione

- **Counter**
 - con attributo `val` intero `protected` e metodi pubblici `reset()`, `inc()`, `getValue()`
 - **senza costruttore di default definito**, ma solo costruttore con un parametro intero `n` che inizializza `val` a `n` (anche un po' di stampe ...)
- **BiCounter** estende **Counter**
 - con un metodo pubblico `dec()`
 - costruttore di default definito che inizializza `val` a 1
- **Main**
 - *invariato*

Terza versione

```
public class Counter
{
    protected int val;
```

**Eliminiamo il
costruttore di default**

```
    public Counter(int v)
    {
        System.out.println(
            "Counter:costruttore");
        val = v;
    }
    public void reset()
    { val = 0; }
    public void inc()
    { val++; }
    public int getValue()
    { return val;}
}
```

```
public class BiCounter
    extends Counter
{
    public BiCounter()
    {   System.out.println("Counter2:
        costruttore di default!");
        val = 1;
    }
    public void dec()
    { val--; }
}
```

```
public class Main
{
    public static void
        main(String Args[])
    {
        Counter c =
            new Counter();
        BiCounter c1 =
            new BiCounter();
    }
}
```

Risultato

```
C:\work\costruttori\Main.java:6: cannot resolve symbol
symbol  : constructor Counter  ()
location: class Counter
    Counter c = new Counter();
                  ^
C:\work\costruttori\Counter.java:1: cannot resolve symbol
symbol  : constructor Counter  ()
location: class Counter
public class BiCounter extends Counter {
    ^
2 errors
Process completed.
```

- Nella classe Counter, il costruttore di default non è stato inserito automaticamente, perché Counter ha almeno un costruttore (quello con parametro)
- Abbiamo quindi **2 errori di compilazione**

Una prima correzione

- Il primo errore è dovuto al fatto che in main() creiamo un'istanza di Counter invocando il costruttore di default che non esiste più
- Modifichiamo quindi main() in modo che l'istanza venga creata invocando il costruttore non di default

```
public class Main
{
    public static void main(String Args[])
    {
        Counter c = new Counter(1);
        Counter c1 = new BiCounter();
    }
}
```

Risultato

```
C:\work\costruttori\Counter.java:1: cannot resolve
symbol
symbol   : constructor Counter   ()
location: class Counter
public class BiCounter extends Counter {
        ^
1 error
```

- Ci resta ancora un errore dovuto al fatto che nel costruttore di BiCounter **il compilatore ha inserito una chiamata a super()** (costruttore di default della classe base) e questo non esiste in Counter

Seconda correzione

- Inseriamo quindi nel costruttore di BiCounter una chiamata esplicita al costruttore effettivamente esistente di Counter (quello con il parametro intero)

```
public class BiCounter
    extends Counter
{
    public BiCounter()
    {   System.out.println("Counter2:
        costruttore di default!");
        super(0);
    }
    public void dec()
    { val--; }
}
```

Risultato

```
C:\work\costruttori\Counter2.java:3: cannot resolve symbol
symbol  : constructor Counter  ()
location: class Counter
{
^
C:\work\costruttori\Counter2.java:5:
call to super must be first statement in constructor
    super(0);
      ^
2 errors
```

- Otteniamo ancora degli errori di compilazione
- Infatti la chiamata a **super(0)** deve essere la prima istruzione del costruttore!

Terza correzione

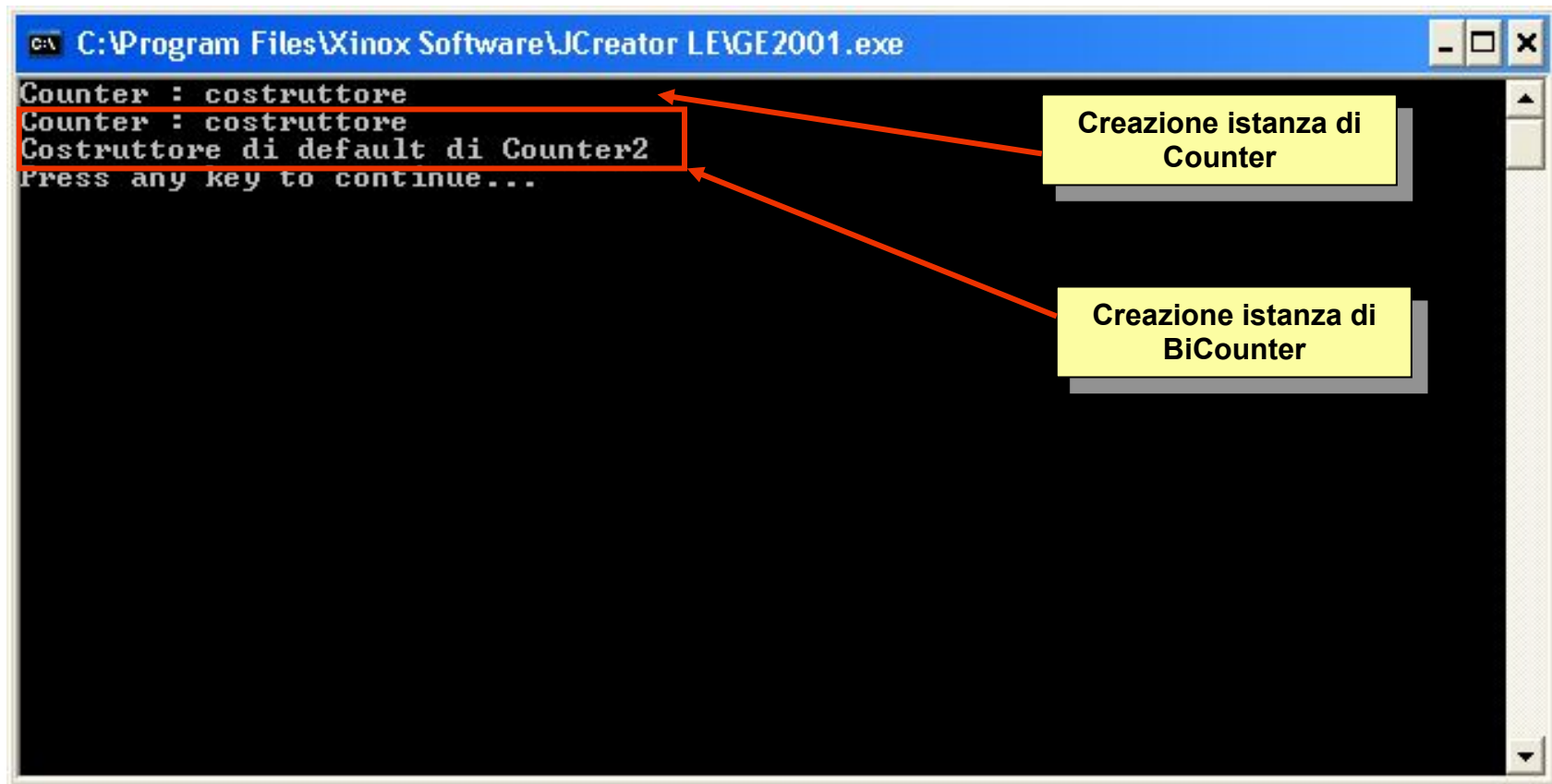
- Mettiamo la chiamata a super in prima posizione:

```
public class BiCounter
    extends Counter
{
    public BiCounter()
    {
        super(0);
        System.out.println(
            "Costruttore di default
            di Counter2");
    }
    public void dec()
    {
        val--;
    }
}
```

```
public class Main
{
    public static void
        main(String Args[])
    {
        Counter c = new Counter(1);
        Counter c1 = new BiCounter();
        c1.inc();
    }
}
```

Risultato finale

- Adesso funziona tutto correttamente



The screenshot shows a Windows command prompt window titled "C:\Program Files\Xinox Software\JCreator LE\GE2001.exe". The output text is as follows:

```
Counter : costruttore  
Counter : costruttore  
Costruttore di default di Counter2  
press any key to continue...
```

Two yellow callout boxes with arrows point to the first two lines of output:

- The first box, labeled "Creazione istanza di Counter", points to the first "Counter : costruttore" line.
- The second box, labeled "Creazione istanza di BiCounter", points to the second "Counter : costruttore" line.

Ricapitolando...

- La chiamata al costruttore del parent (**super()** con o senza argomenti) deve essere la **prima istruzione dei costruttori** delle classi derivate
- Se non è specificato **super()** nei costruttori delle classi derivate, è invocato automaticamente il costruttore di **default** della classe base
- Il **costruttore di default** può essere **generato automaticamente** solo se nella classe base non è definito alcun costruttore
- Se non è presente un costruttore di default (definito dal programmatore o generato in automatico) di una classe, le classi derivate da questa **devono esplicitamente usare la super(...)** ed invocare il **corretto costruttore (non di default)** se questo c'è