Corso sul linguaggio Java

Modulo JAVA4

B2 – Object

Prerequisiti

- Programmazione elementare ad oggetti
- Ereditarietà
- Concetto di conversione di tipo (**casting**)

Introduzione

Lo scopo di questa Unità è quello di mostrare in pratica una delle classi più importanti di Java: la superclasse di tutte le classi.

Si tratta della classe chiamata **Object**, che esibisce metodi piuttosto importanti che realizzano funzionalità particolari, alcune delle quali vedremo in dettaglio.

La classe **Object**

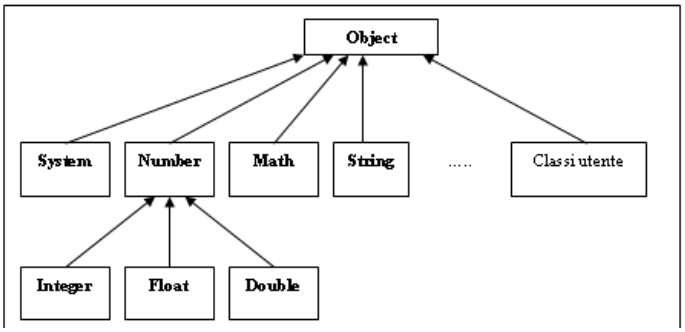
Abbiamo visto che quando si crea una classe, questa può essere derivata da un'altra mediante la clausola **extends**.

Se omettiamo la parola extends, dove viene collocata la nostra classe?

Esiste una classe, la classe Object, dalla quale derivano tutte le altre classi (comprese quelle che implementiamo). Essa è la classe più generale e si dice <u>superclasse universale</u>

La classe **Object**

Quando creiamo una nostra classe essa viene collocata come mostrato nel disegno.



La classe **Object**

Un qualunque oggetto quindi è riconducibile alla classe **Object**. Perciò per un qualunque oggetto dovremmo poter rispondere alle seguenti domande:

Come si può confrontare con un altro oggetto?
Come si può convertire in una stringa?
Come ottenere la classe di appartenenza?

La classe **Object** risponde a queste domande fornendo appositi metodi.

I metodi della classe **Object**

I seguenti sono i metodi della classe **Object** che possono essere ridefiniti nella classe utente.

public boolean equals (**Object** o)	**true** se i due oggetti sono uguali
public int hashCode ()	Codice identificativo dell'oggetto
protected Object clone ()	Crea una copia dell'oggetto
protected void finalize ()	Elimina l'oggetto dalla memoria
public String toString()	Dà stringa descrittiva dell'oggetto
public final Class getClass()	Dà la classe di appartenenza

Questo metodo non può essere ridefinito

public boolean equals (Object obj)

Questo metodo restituisce **true** se i due oggetti (il parametro e l'oggetto su cui si istanzia **equals**()) *si riferiscono allo stesso oggetto.*

Ad esempio:

```
Coppia c1 = new Coppia(3,4);
Coppia c2 = c1;   // uguaglia i riferimenti
System.out.println (c1.equals(c2));
```

stamperà **true**

public int hashCode()

Questo metodo restituisce un codice intero che identifica l'oggetto.

Ad esempio:

```
Coppia c1 = new Coppia(3,4);
System.out.println (c1.hashCode());
```

stamperà un valore intero (ad esempio 8567361) che rappresenta l'identificativo dell'oggetto *c1*.

Se due oggetti verificano il metodo **equals**(..) i loro *hashcode* sono uguali.

Due oggetti con *hashcode* uguale non è detto che siano uguali.

protected Object clone()

Questo metodo

- crea una copia dell'oggetto su cui è istanziato
- pone nella copia le stesse informazioni dell'oggetto di partenza

Gli oggetti ottenuti rappresentano la stessa cosa (che è *diverso dal dire che sono lo stesso oggetto*, verifica che si esegue con l'operatore "= =").

Esempio:

```
Coppia c = new Coppia(2, 3), c1 = new Coppia();
c1=c;                          // sono lo stesso oggetto
c1 = (Coppia) c.clone();       // c1 è la copia di c
```

Il metodo **clone**() può essere ridefinito.

protected void finalize()

Questo metodo viene chiamato dal modulo *garbage collector* della JVM quando non ci sono più riferimenti all'oggetto stesso.

In generale lo scopo di **finalize**() è quello di compiere operazioni di "pulizia" (interruzione di operazioni di I/O, chiusura di file, …) prima di eliminare definitivamente l'oggetto.

In pratica, il metodo viene ridefinito per compiere azioni specifiche, ma si tenga presenta che il garbage collector già svolge perfettamente il compito di pulizia.

Ad esempio:

```
Punto p = new Punto (3,4);
……
p.finalize();
```


public String toString()

Questo metodo restituisce una *rappresentazione testuale* dell'oggetto su cui è invocato e restituisce una stringa del tipo:

nomeclasse@hashcode

dove:

- *classe* indica il nome della classe;
- *hashCode* indica l'indirizzo esadecimale in memoria dell'oggetto.

Ad esempio:

```
Coppia c1 = new Coppia(3,4);
System.out.println (c1.toString());
```

stamperà
Coppia@f5da06

Si usa spesso nella fase di debugging.

È consigliabile ridefinirlo nella applicazioni.

public String toString()

Gli oggetti vengono indicati in questo formato *perché la classe* **Object** *non può conoscere la struttura dell'oggetto*.

Essendo una forma piuttosto criptica, conviene estrarre le informazioni contenute nell'oggetto e presentarle in modo migliore, ridefinendo ad esempio il metodo come segue:

```
public String toString()
  {
        return "Coppia = [X= " + X + ", Y= " + Y + "]";
  }
```

Per convenzione gli oggetti vanno stampati con il formato indicato (nell'esempio "Coppia = [X=3, Y=4]")

public final Class getClass()

Questo metodo restituisce informazioni sulla classe a cui appartiene l'oggetto che l'ha invocato.

Ad esempio:

```
Coppia c1 = new Coppia(3,4);
System.out.println (c1.getClass());
```

stamperà

Class Coppia

che rappresenta il nome della classe.

Analogo effetto si ottiene con:
System.out.println (c1.**getClass**().**getName**());

public final Class getClass()

Un'altra applicazione di questo metodo consente di creare un altro oggetto della stessa classe di quello che ha invocato il metodo.

Ad esempio:

```
Coppia c1 = new Coppia(3,4);
Object c2 = c1.getClass().newInstance();
```

L'oggetto *c2* è anche esso di classe *Coppia*.

Utilizzo della classe **Object**

Il metodo **toString** () va *ridefinito* per stampare l'oggetto nel formato

nomeclasse = [*attributo1* = *valore*, …, *attributoN* = *valore*]

```
public class Coppia
{   private int x, y;
    public int getX()  {  return x; }
    public int getY()  {  return y; }
    public Coppia (int a, int b)  { x=a;  y=b;  }
    public Coppia () { x=0;  y=0; }
    public String toString ()  { return "Coppia = [x="+x+",y="+y+"]";}
    public void setX (int a)  { x=a; }
    public void setY (int b)  { y=b; }
    public Object clone () { return new Coppia(2*x,2*y);  }
}  // end class
```

Il metodo **clone** () va *ridefinito* per creare l'oggetto con attributi del valore desiderato

Utilizzo della classe **Object**

```
public class TestCoppia
{  ...
   Coppia c = new Coppia(3, 4),  c1 = new Coppia();
   c1=c; // esegue la copia dell'oggetto
   System.out.println("c.equals(c1)="+ c.equals(c1));
   System.out.println("hashCode c: "+ c.hashCode());
   System.out.println("hashCode c1: "+ c1.hashCode());
   System.out.println(c.getClass());
   System.out.println(c1.getClass());
   System.out.println(c.toString());
   System.out.println(c1.toString());
    ….
}  // end class
```

Stampa **true**

Stampa codici hash identici. *c* e *c1 sono lo stesso oggetto*

Stampa Coppia

Stampa
Coppia = [x= 3, y= 4]
Coppia = [x= 3, y= 4]

Utilizzo della classe **Object**

```
public class TestCoppia
{  ...
   Coppia c = new Coppia(3, 4),  c1 = new Coppia();
   // esegue la copia del contenuto
   c1=(Coppia) c.clone();
   System.out.println("c.equals(c1)=" + c.equals(c1));
   System.out.println("hashCode c: " +  c.hashCode());
   System.out.println("hashCode c1: "+ c1.hashCode());
   System.out.println(c.toString());
   System.out.println(c1.toString());
}  // end class
```

Stampa **false**

Stampa codici hash diversi. *c* e *c1 NON sono lo stesso oggetto*

Stampa
Coppia = [x= 3, y= 4]
Coppia = [x= 6, y= 8]

Argomenti

- La classe **Object**
- I metodi della classe **Object**
- **public boolean equals(Object** obj)
- **public int hashCode**()
- **protected Object clone**()
- **protected void finalize**()
- **public String toString**()
- **public final Class getClass**()
- Utilizzo della classe **Object**

Altre fonti di informazione

- P.Gallo, F.Salerno – Informatica Generale 1, ed. Minerva Italica
- M.Romagnoli, P.Ventura – Linguaggio C/C++, ed. Petrini
- M. Bigatti – Il linguaggio Java, ed. Hoepli