UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Metodologie di Programmazione

Lezione 16: La classe Object; la classe ArrayList

Lezione 16: Sommario

UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- La classe Object
- La classe ArrayList

La superclasse universale Object

- Tutte le classi in Java ereditano direttamente o indirettamente dalla classe java.lang.Object
- Tutti i suoi 11 metodi sono ereditati
- Quando si definisce una classe senza estenderne un'altra:

```
public class LaMiaClasse
{

}
```

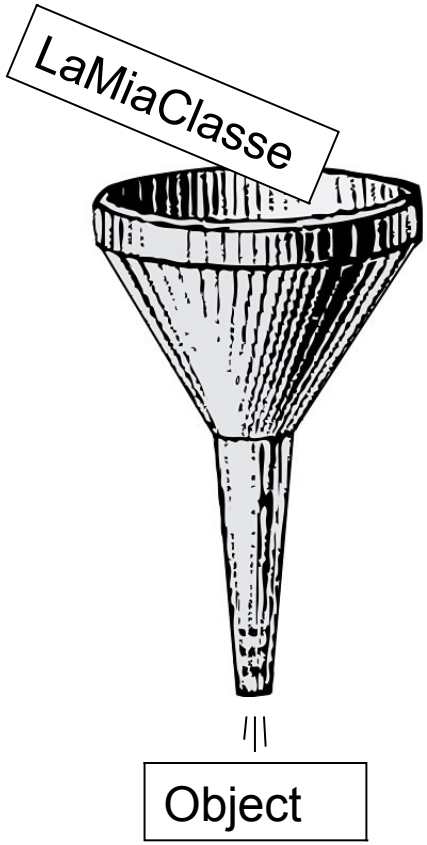
questo è equivalente a estendere Object:

```
public class LaMiaClasse extends Object
{

}
```

La superclasse Object in uno slogan

- La classe Object è una sorta di "massimo comun denominatore" di tutte le classi in Java



I metodi principali della classe Object

http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html

Posso confrontare qualsiasi oggetto!

Gli array sovrascrivono il metodo clone!

Metodo	Descrizione
Object **clone**()	Restituisce una copia dell'oggetto
boolean **equals**(Object o)	Confronta l'oggetto con quello in input
Class<? extends Object> **getClass**()	Restituisce un oggetto di tipo Class che contiene informazioni sul tipo dell'oggetto
int **hashCode**()	Restituisce un intero associato all'oggetto (per es. ai fini della memorizzazione in strutture dati, hashtable, ecc.)
String **toString**()	Restituisce una rappresentazione di tipo String dell'oggetto (per default: tipo@codice hash)

DNA dell'oggetto!

Sovrascrivere il metodo toString

- toString è uno dei metodi che ogni classe eredita direttamente o indirettamente dalla classe Object
- Non prende argomenti e restituisce una String
- Chiamato implicitamente quando un oggetto deve essere convertito a String (es. System.out.println(o))
- L'annotazione @Override serve a garantire che il metodo "sovrascriva" il metodo di una superclasse

```
public class Punto
{
    private int x, y, z;

    public Punto(int x, int y, int z)
    {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    @Override
    public String toString() { return "("+x+","+y+","+z+")"; }
}
```

Sovrascrive l'implementazione di Object.toString()

Sovrascrivere il metodo equals

- Il metodo equals viene invocato per confrontare il contenuto di due oggetti
- Per default, se sono "uguali", il metodo restituisce true

```
public boolean equals(Object o) { return this == o; }
```

- Tuttavia, la classe Object **non** conosce il contenuto delle sottoclassi
- Per mantenere il "contratto" del metodo è necessario sovrascriverlo

```
public class Punto
{
	private int x, y, z;

	// ...

	@Override
	public boolean equals(Object o)
	{
		if (o == null) return false;
		if (getClass() != o.getClass()) return false;
		Punto p = (Punto)o;
		return x == p.x && y == p.y && z == p.z;
	}
}
```

In questa situazione, meglio di instanceof (che accetta anche sottoclassi)

Sovrascrivere il metodo clone

- L'operatore di assegnazione = **non** effettua una copia dell'oggetto, ma solo del riferimento all'oggetto
- Per creare una copia di un oggetto è necessario richiamare clone()
- Tuttavia l'implementazione di Object.clone copia l'oggetto membro a membro
- Se il nostro oggetto contiene riferimenti e vogliamo evitare che la copia contenga un riferimento allo stesso oggetto membro, dobbiamo sovrascrivere clone()

Enumerazioni e Object

- Una enumerazione ha tante istanze quante sono le costanti enumerative al suo interno
- **Non** è possibile costruire altre istanze
- Le classi enumerative estendono la classe java.lang.Enum, da cui ereditano i metodi toString e clone
  - toString() restituisce il nome della costante
  - clone() restituisce l'oggetto enumerativo stesso senza farne una copia (che non è possibile fare…)
- Enum a sua volta estende Object, per cui il metodo equals restituisce true solo se le costanti enumerative sono identiche

Ancora sulle enumerazioni

- **Non** possono essere create nuove istanze
- **MA** possono essere costruite le istanze "costanti"
  - Si definisce un costruttore (**NON** pubblico, ma con visibilità di default)
  - Si costruisce ciascuna costante (un oggetto **separato** per ognuna)
  - Si possono definire altri metodi di accesso o modifica dei campi, ecc.

```
public enum TipoDiMoneta
{
    /**
     * Le costanti enumerative, costruite in modo appropriato
     */
    CENT(0.01), CINQUE_CENT(0.05), DIECI_CENT(0.10), VENTI_CENT(0.20), CINQUANTA_CENT(0.50), EURO(1.00), DUE_EURO(2.00);

    /**
     * Valore numerico della costante
     */
    private double valore;

    /**
     * Costruttore con visibilita' di default
     */
    TipoDiMoneta(double valore) { this.valore = valore; }

    /**
     * Metodo di accesso al valore
     */
    public double getValore() { return valore; }
}
```

Esempio: i pianeti “enumerati”

```
public enum Pianeta
{
    MERCURIO (3.303e+23, 2.4397e6),   VENERE   (4.869e+24, 6.0518e6),
    TERRA    (5.976e+24, 6.37814e6),  MARTE    (6.421e+23, 3.3972e6),
    GIOVE    (1.9e+27,   7.1492e7),   SATURNO  (5.688e+26, 6.0268e7),
    URANO    (8.686e+25, 2.5559e7),   NETTUNO  (1.024e+26, 2.4746e7);

    /**
     * Costante di gravitazione universale
     */
    public static final double G = 6.67300E-11;

    /**
     * Massa in kilogrammi
     */
    private final double massa;

    /**
     * Raggio in metri
     */
    private final double raggio;

    Pianeta(double massa, double raggio)
    {
        this.massa = massa;
        this.raggio = raggio;
    }

    private double getMassa() { return massa; }
    private double getRaggio() { return raggio; }
    public double getGravitaDiSuperficie() { return G * massa / (raggio * raggio); }
    public double getPesoDiSuperficie(double altraMassa) { return altraMassa * getGravitaDiSuperficie(); }
}
```

Le liste in Java

- Gli array sono strutture di base per la memorizzazione di sequenza
  - Tuttavia, sono di dimensioni statiche!
- Java mette a disposizione una serie di classi per la memorizzazione dinamica di sequenze
  - ArrayList, LinkedList, ecc.
- Sottoclassi di AbstractList e AbstractCollection, ma con implementazioni differenti

La classe java.util.ArrayList

- Implementa un vettore o una lista ad accesso casuale
- Utilizza un array per implementare la lista **MA:**
  - La dimensione della lista può aumentare o diminuire
  - La classe fornisce metodi per svolgere le operazioni più importanti
- Esempio di istanziazione:

```
ArrayList<String> l = new ArrayList<String>();
```

- Osserviamo la notazione <String> (un tipo tra parentesi angolari), che indica che utilizziamo una lista di stringhe
- Indica che la classe è generica, ovvero può essere utilizzata con tipi differenti

Alcuni metodi di java.util.ArrayList

- Aggiungere elementi in coda alla lista (o in posizione k):

```
l.add("a");
l.add(k, "b");
```

- Eliminare elementi dalla lista (o dalla posizione k):

```
l.remove("a");
l.remove(k);
```

- Dimensione della lista:

```
l.size();
```

- Assegnare un elemento a k:

```
l.set(k, "c");
```

- Leggere un elemento in posizione k:

```
l.get(k);
```

Errori comuni: lunghezza di array, liste e stringhe

- Fate **MOLTA** attenzione alle differenze:

Tipo	Come ottenere la lunghezza	Metodo o campo?
array	a.length	Campo
ArrayList	a.size()	Metodo
String	a.length()	Metodo