Università degli Studi di Roma “La Sapienza”
Facoltà di Ingegneria – Corso di Laurea in Ingegneria Gestionale

Corso di Progettazione del Software

Proff. Toni Mancini e Monica Scannapieco
Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”

S.JOO.3 – Java: le classi Object e Class

versione del February 2, 2008

La classe Object

Implicitamente, **tutte le classi** (predefinite o definite da programma) sono derivate, direttamente o indirettamente, dalla classe `Object`. Ad esempio definire la seguente classe:

```
public class Punto {  ...  }
```

equivale a definirla come sottoclasse di `Object`:

```
public class Punto extends Object {  ...  }
```

Di conseguenza, tutti gli oggetti, qualunque sia la classe a cui appartengono, **sono anche implicitamente istanze** della classe predefinita `Object`.
Queste sono alcune funzioni della classe `Object` che vedremo nel seguito:

- `public String toString()`
- `public final Class getClass()`
- `public boolean equals(Object)`
- `public int hashCode()`
- `protected Object clone()`

Stampa di oggetti e funzione toString()

La funzione `public String toString()` di `Object` associa una **stringa stampabile** all'oggetto di invocazione.
Se ne può fare overriding in modo opportuno nelle singole classi in modo da generare una **forma testuale** conveniente per gli oggetti della classe.

```
// File Codice/J2/Esempio20.java
class B {
  private int i;
  B(int x) { i = x; }
  public String toString() { return "i: " + i; }
}

public class Esempio20 {
  public static void main(String[] args) {
    B b = new B(5);
    System.out.println(b);
  }
}

/* Stampa:
   i: 5
   Nota: se non avessimo ridefinito toString() avrebbe stampato
   B@601bb1
*/
```

Esercizio 10: overriding di toString()

Facendo riferimento alle classi `Punto` e `Segmento` viste in precedenza, ridefinire la funzione `toString()` per esse.
In particolare, vogliamo che un punto venga stampato in questo formato:

```
<1.0;2.0;4.0>
```

e che un segmento venga stampato in questo formato:

```
(<1.0;2.0;4.0>,<2.0;3.0;7.0>)
```

Stampa in classi derivate

Nel fare overriding di `toString()` per una classe derivata è possibile riusare la funzione `toString()` della classe base.

```
class B {
  protected int x, y;
  public String toString() { // ...
  // ...
}

class D extends B {
  protected int z;
  public String toString() {
    return super.toString() + // ...
  }
  // ...
}
```

La classe Java Class

- Esiste implicitamente un oggetto di classe `Class` per ogni classe (o interfaccia) `B` del programma, sia di libreria che definita da utente.
- Questo oggetto può essere denotato in due modi:
  - tramite letterali aventi la forma:

    ```
    ...  B.class ... // ha tipo Class
    ```

  - tramite riferimenti di tipo `Class`

    ```
    Class c = ...
    ```

- Gli oggetti di tipo `Class` sono creati dal sistema runtime in modo automatico. Si noti che `Class` non ha costruttori accessibili dai clienti.
- La classe `Class` ha una funzione dal significato particolare:

  ```
  boolean isInstance(Object)
  ```

  che restituisce `true` se e solo se il suo parametro attuale è un riferimento ad oggetto di una classe compatibile per l'assegnazione con la stessa classe dell'oggetto di invocazione.

La funzione isInstance()

- La funzione `isInstance()` può essere usata per verificare se un oggetto è istanza di una classe.

```
... B.class.isInstance(b) ... // vale true se b e' istanza di B
```

- Al riguardo, si ricorda che un oggetto di una classe `D` derivata da una classe `B` è anche oggetto della classe `B`.

```
class D extends B ...

D d1 = new D();
... B.class.isInstance(d1) ... // vale true;
```

Esercizio 11: cosa fa questo programma?

```
// File Codice/J2/Esercizio11.java

class B {}
class D extends B {}

public class Esercizio11 {
  public static void main(String[] args) {
    B b1 = new B();
    D d1 = new D();
    System.out.println(B.class.isInstance(d1));
    System.out.println(D.class.isInstance(b1));
  }
}
```

La funzione isInstance() (cont.)

- La funzione `isInstance()` può essere anche usata per verificare se un oggetto è istanza di una classe che implementa una interfaccia.

```
interface I {...}

... I.class.isInstance(b) ... // vale true, se b e' istanza di
                              // una classe che implementa I

class D implements I {...}

D d1 = new D();
... I.class.isInstance(d1) ... // vale true;
```

Esercizio 11bis: cosa fa?

```
// File Codice/J2/Esercizio11bis.java

interface I {}
class D implements I {}

public class Esercizio11bis {
  public static void main(String[] args) {
    I i1 = new D();
    D d1 = new D();
    System.out.println(I.class.isInstance(i1));
    System.out.println(I.class.isInstance(d1));
  }
}
```

l'operatore Java instanceof

Java è dotato di un operatore predefinito `instanceof` per verificare l'appartenenza di un oggetto ad una classe o la conformità di un oggetto ad una interfaccia.

In particolare le seguenti espressioni booleane si comportano in modo identico:

```
... B.class.isInstance(b) ...

... b instanceof B ...
```

Si noti che nell'ultima espressione si è usato `B` e non `B.class`. Questo perché l'operatore `instanceof` non fa uso di un oggetto della classe `Class`, ma del **nome della classe**. Ne segue che per poter applicare `instanceof` la classe a cui applicarlo deve essere nota a tempo di compilazione. Quindi la seguente istruzione non è riscrivibile utilizzando `instanceof`:

```
Class c = ...

...c.isInstance(b)...
```

La funzione getClass() di Object

La classe `Object` contiene una funzione `public final Class getClass()` (che non può essere ridefinita) che restituisce la classe dell'oggetto di invocazione, cioè la classe più specifica di cui l'oggetto di invocazione è istanza.

Attraverso l'uso di `getClass()` (e di `equals()` definito per gli oggetti di tipo `Class`), possiamo, ad esempio, verificare se due oggetti appartengono alla stessa classe:

```
class B {
  private int x;
  public B(int n) {x=n;}
  ...
}

B b1 = new B(10);
...
B b2 = new B(100);
... b1.getClass().equals(b2.getClass()) ... // vale true
```