Polimorfismo

Riprendiamo l'esempio: Counter e CentoCounter - 2

- Quando c'è overriding, si può avere comportamento polimorfo (lo stesso codice/chiamata innesca comportamenti diversi):

```
public class Esempio {
   public static void main(String[] args) {
    int n;
    Counter c1;
    c1 = new Counter();
    c1.reset();
    for (int i=0;i<150;i++)
        c1.inc();
    n = c1.getValue();
    System.out.println(n);
   }
}
```

Riprendiamo l'esempio: Counter e CentoCounter - 2

- Quando c'è overriding, si può avere comportamento polimorfo (lo stesso codice/chiamata innesca comportamenti diversi):

```
public class Esempio {
   public static void main(String[] args) {
    int n;
    Counter c1;
    c1 = new CentoCounter();
    c1.reset();
    for (int i=0;i<150;i++)
        c1.inc();
    n = c1.getValue();
    System.out.println(n);
   }
}
```

Subtyping e polimorfismo - 2

- In. Java abbiamo che:
  - Il tipo/classe del riferimento determina quello che si può fare: possiamo invocare solo i metodi definiti nella classe a cui il riferimento appartiene (subtyping)
  - Il tipo/classe dell'istanza determina cosa viene effettivamente fatto: viene invocato il metodo definito nella classe a cui l'istanza appartiene (polimorfismo)

- Per risolvere le chiamate ai metodi Java utilizza late binding
- NOTA BENE: il codice delle chiamate è individuato dinamicamente sulla base della natura dell'istanza

Polimorfismo – Esempio classe persona ...

- Definiamo la classe Persona che gestisce i dati anagrafici di una persona (nome ed età per semplicità)
- La classe definisce un costruttore e il metodo print che stampa a video nome ed età:

```
public class Persona
{
  protected String nome;
  protected int anni;
  public Persona(String n, int a)
  { nome=n;
    anni=a;
  }
  public void print()
  {
    System.out.print(”Mi chiamo " + nome);
    System.out.println(" e ho ” + anni + "anni");
  }
}
```

Esempio: … e studenti

- Definiamo la classe Studente, sottoclasse di Persona che ridefinisce il metodo print():

```
public class Studente extends Persona
{
  protected int matr;
  public Studente(String n, int a, int m)
  {
    super(n,a);
    matr=m;
   }
  public void print()
  {
    super.print(); // stampa nome ed età
    System.out.println("Matr = " + matr);
  }
}
```

- In questo modo print() stampa nome, età e matricola

EsempioPoli

- Definiamo infine la classe EsempioPoli che implementa il metodo statico main ed è quindi la classe principale della nostra applicazione:

```
public class EsempioPoli
{
  public static void main(String args[])
  {
    Persona  p = new Persona("John",45);
    Studente s = new Studente("Tom",20,156453);
    p.print();  // stampa nome ed età
    s.print();  // stampa nome, età, matricola
    p=s;        // Ok, per il subtyping
    p.print();  // COSA STAMPA ???
  }
}
```

- Cosa stampa l'ultima istruzione?

Polimorfismo

- La risposta è: stampa nome, età e matricola!
- p è un riferimento di tipo Persona,
- Però p punta ad un'istanza di classe Studente
- Se scriviamo p.print() viene eseguito il metodo print() ridefinito dalla classe Studente e non quello originale definito nella classe Persona
- Quindi: anche se usiamo un riferimento che ha per tipo una superclasse il fatto che l'istanza a cui il riferimento punta appartenga alla sottoclasse fa sì che il codice del metodo invocato sia quello della sottoclasse
- Questa proprietà prende il nome di polimorfismo (verticale)
- Ereditarietà e polimorfismo sono i due principi che differenziano la programmazione object-oriented dalla programmazione object-based

Subtyping e polimorfismo - 1

- Subtyping e polimorfismo sono strettamente correlati
- Grazie al subtyping possiamo scrivere:
- ```
  Persona p;
  p = new Studente("Pietro",22,456327);
  ```
- Abbiamo cioè assegnato un'istanza di tipo Studente a un riferimento di tipo Persona
- Di conseguenza abbiamo potuto scrivere:

  ```
  p.print();
  ```
- In virtù del subtyping questa espressione è valida
- In virtù del polimorfismo il metodo print() che viene eseguito è quello di Studente
  ```

Riprendiamo l'esempio: Counter e CentoCounter - 2

- E' esattamente ciò che succede sostituendo un'istanza di Counter con una di CentoCounter nel nostro esempio:

```
public class Esempio {
   public static void main(String[] args) {
    int n;
    Counter c1;
    c1 = new CentoCounter();
    c1.reset();
    for (int i=0;i<150;i++)
        c1.inc();
    n = c1.getValue();
    System.out.println(n);
   }
}
```

- Il codice eseguito per la chiamata inc() è quello di CentoCounter

Subtyping e polimorfismo - 2

- Riassumendo:
  - Il tipo del riferimento determina quello che si può fare: possiamo invocare solo i metodi definiti nella classe a cui il riferimento appartiene (subtyping)
  - Il tipo dell'istanza determina cosa viene effettivamente fatto: viene invocato il metodo definito nella classe a cui l'istanza appartiene (polimorfismo)

- Per risolvere le chiamate ai metodi Java utilizza late binding
- NOTA BENE: il codice delle chiamate è individuato dinamicamente sulla base della natura dell'istanza