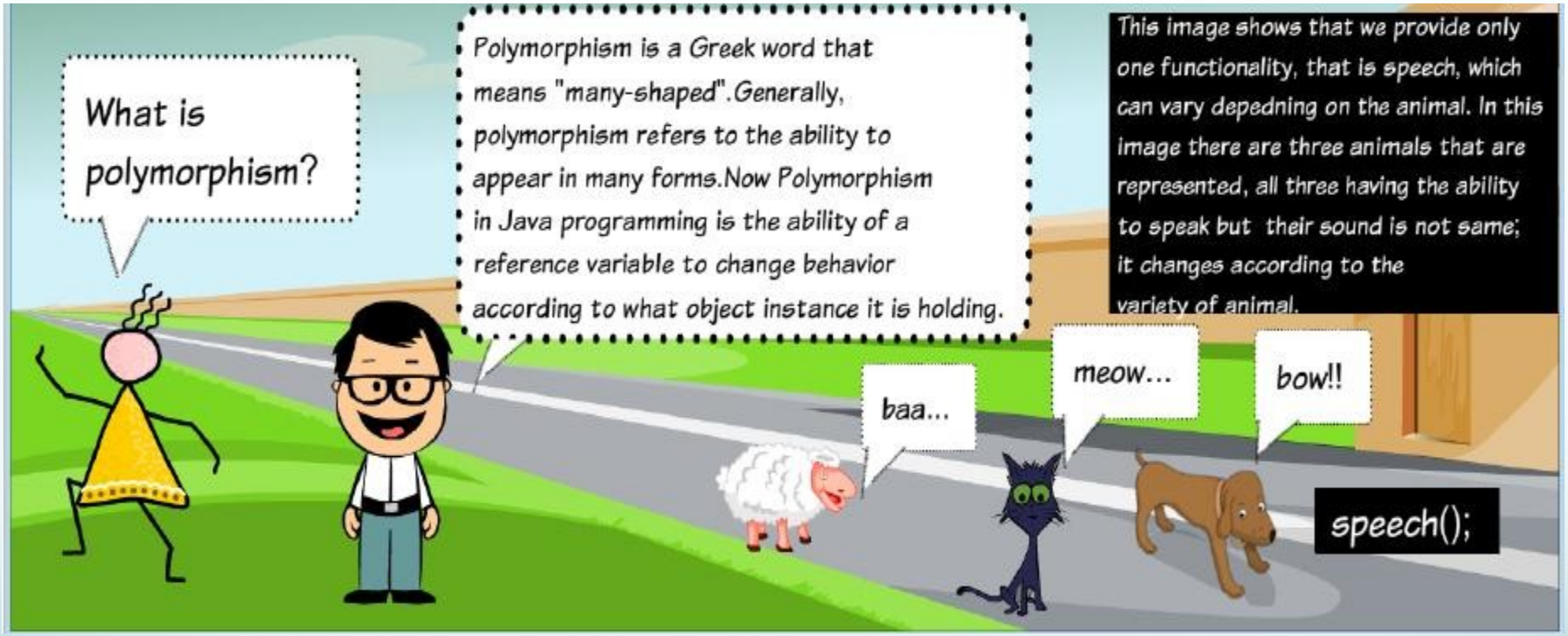
OOP:il polimorfismo

Prof.Francesco Viglietti
www.in4matika.altervista.org

Polimorfismo



Polimorfismo

É uno dei pilastri fondamentali della OOP. Il significato di polimorfismo è multiforme cioè la capacità di un oggetto di poter assumere diverse forme. Il concetto è spesso ostico ai neofiti della OOP, ma in realtà, con un esempio pratico si può cominciare a capire la potenza di questo sistema.

ESEMPIO: Un leone, una mucca e un cavallo avranno tutti e tre la capacità di emettere un proprio verso. Vediamolo nel dettaglio:

```
public class Leone {
   public void emettiVerso(){
      System.out.println("RUGGITO");
   }
}
```

```
public class Mucca {
   public void emettiVerso(){
      System.out.println("MUGGITO");
   }
}
```

```
public class Cavallo {
   public void emettiVerso(){
      System.out.println("NITRITO");
   }
}
```

Polimorfismo

In questi esempi il modo per far emettere un verso ai diversi "animali" è questo:

```
public class Fattoria {
   public static void main(String args[]){
      Leone leo = new Leone();
      leo.emettiVerso();
      Mucca muu = new Mucca();
      muu.emettiVerso();
      Cavallo cav = new Cavallo();
      cav.emettiVerso();
   }
}
```

Quindi l’output sarà RUGGITO, MUGGITO, NITRITO

Polimorfismo

Vediamo come sfruttare il polimorfismo. Creiamo una classe Animale che comprenda tutti i diversi animali della nostra Fattoria.

```
public interface Animale {
   public void emettiVerso(){
        System.out.println(“verso”);
   }
}
```

Con questa classe ci siamo assicurati che tutti gli Animali sappiano emettere un verso (con il metodo emettiVerso()). Facciamo in modo che il Leone, la Mucca e il Cavallo siano degli Animali ereditando la classe

```
public class Leone extends Animale {
   public void emettiVerso(){
      System.out.println("RUGGITO");
   }
}
public class Mucca extends Animale {
   public void emettiVerso(){
      System.out.println("MUGGITO");
   }
}
public class Cavallo extends Animale {
   public void emettiVerso(){
      System.out.println("NITRITO");
   }
}
```

Polimorfismo

Adesso modifichiamo leggermente la classe main Fattoria:

```
public class Fattoria {
   public static void main(String args[]){
      Animale leo = new Leone();
      leo.emettiVerso();
      Animale muu = new Mucca();
      muu.emettiVerso();
      Animale cav = new Cavallo();
      cav.emettiVerso();
   }
}
```

L'output sarà ugualmente: RUGGITO, MUGGITO, NITRITO
Praticamente i tre oggetti di diverso tipo sono tutti e tre degli Animali generici, ma conservano le proprie specifiche caratteristiche.
Il polimorfismo è possibile grazie al binding dinamico di Java. Infatti il compilatore non decide quale metodo chiamare nella fase di compilazione, ma si limita a creare delle tabelle che consentono al programma di scegliere il metodo giusto a runtime in base alla classe dell'oggetto.

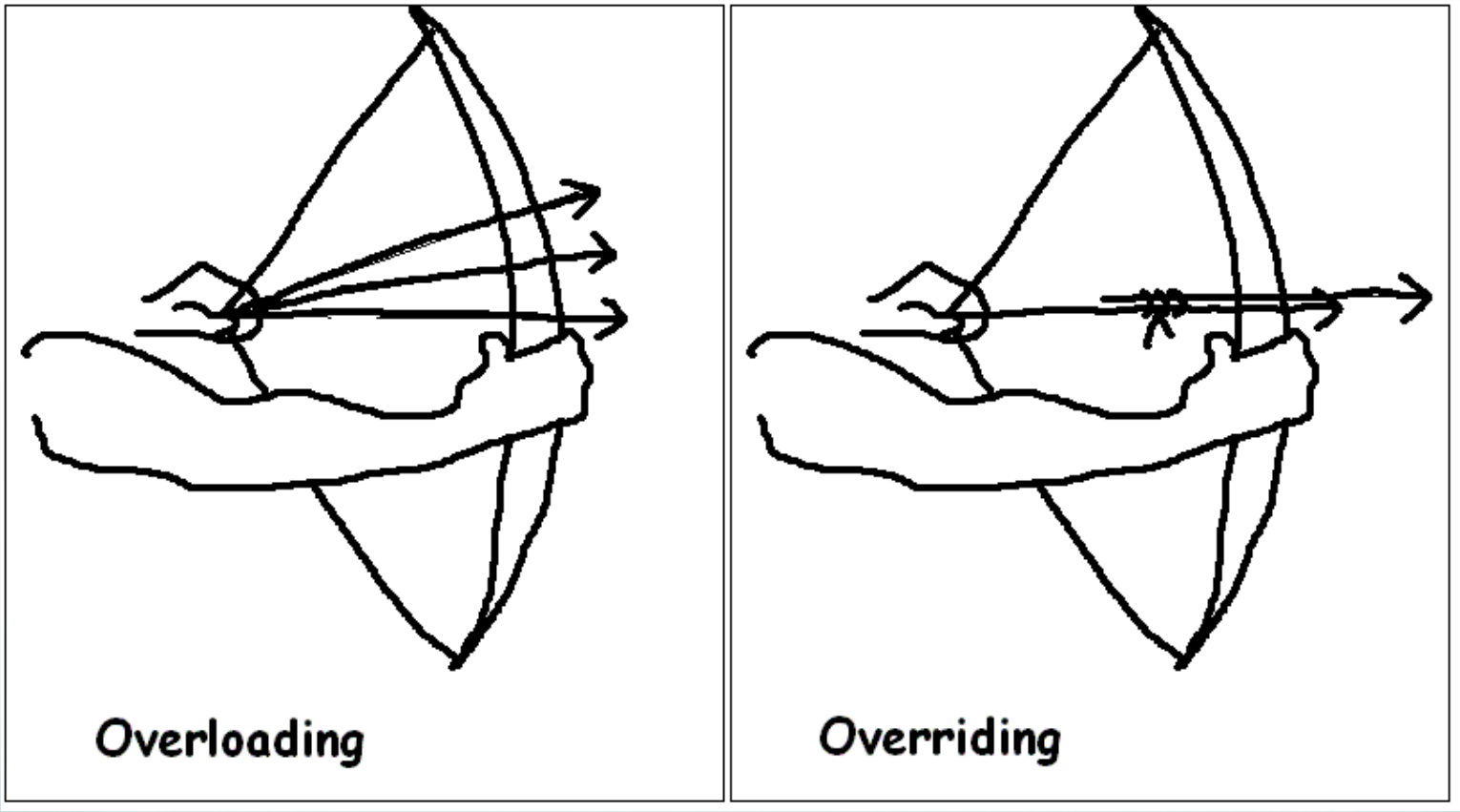
Polimorfismo dei metodi

Il **polimorfismo** dei metodi, indica la possibilità per i metodi di assumere forme, cioè implementazioni, diverse all'interno della gerarchia delle classi.

Esistono due tipi di polimorfismo:

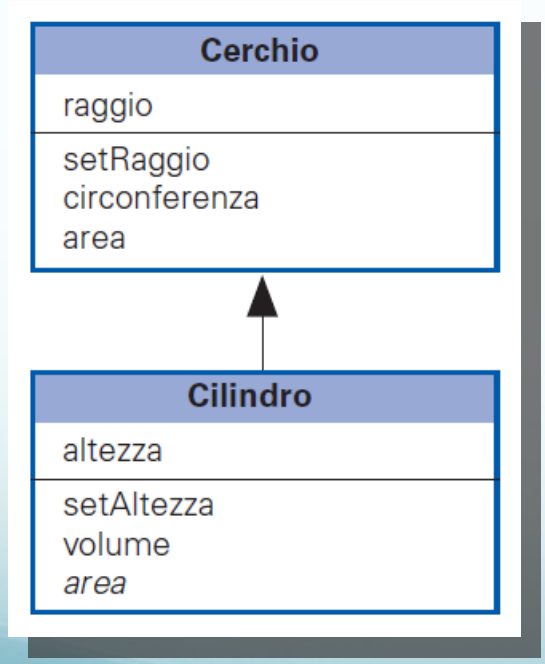
- l'**overriding**, o *sovrapposizione* dei metodi,
- l'**overloading**, o *sovraccarico* dei metodi.

Polimorfismo dei metodi



Overriding

- **Overriding:** ridefinisce, nella classe derivata, un metodo ereditato con lo scopo di modificarne il comportamento. Il nuovo metodo deve avere lo stesso nome e gli stessi parametri del metodo che viene sovrascritto.




```
public double area()
{
        double supBase, supLater;
        supBase=super.area()*2;
        supLater=circonferenza()*altezza;
        return supBase + supLater;
}
```

Overloading

- L’**overloading** di un metodo è la possibilità di utilizzare lo stesso nome per compiere operazioni diverse. Solitamente si applica ai metodi della stessa classe che si presentano con lo stesso nome, ma con un numero o un tipo diverso di parametri.