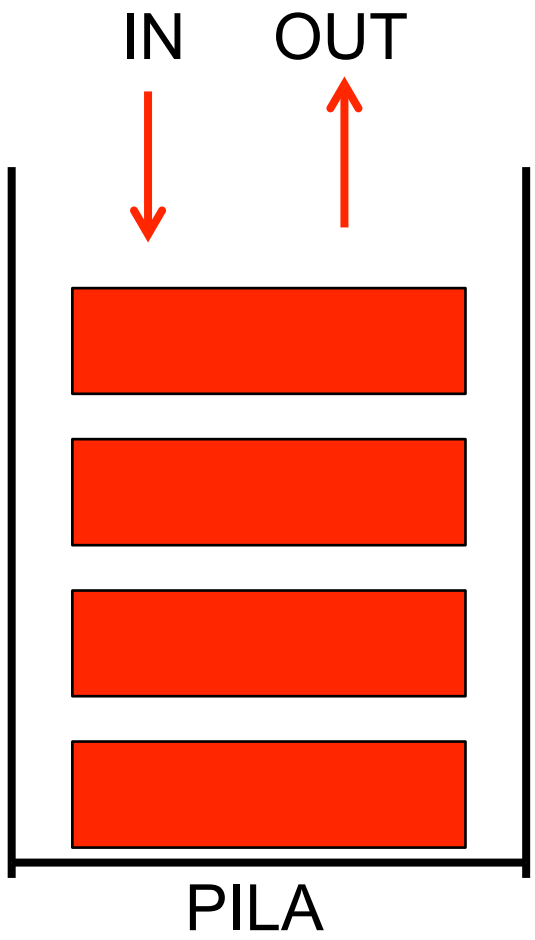
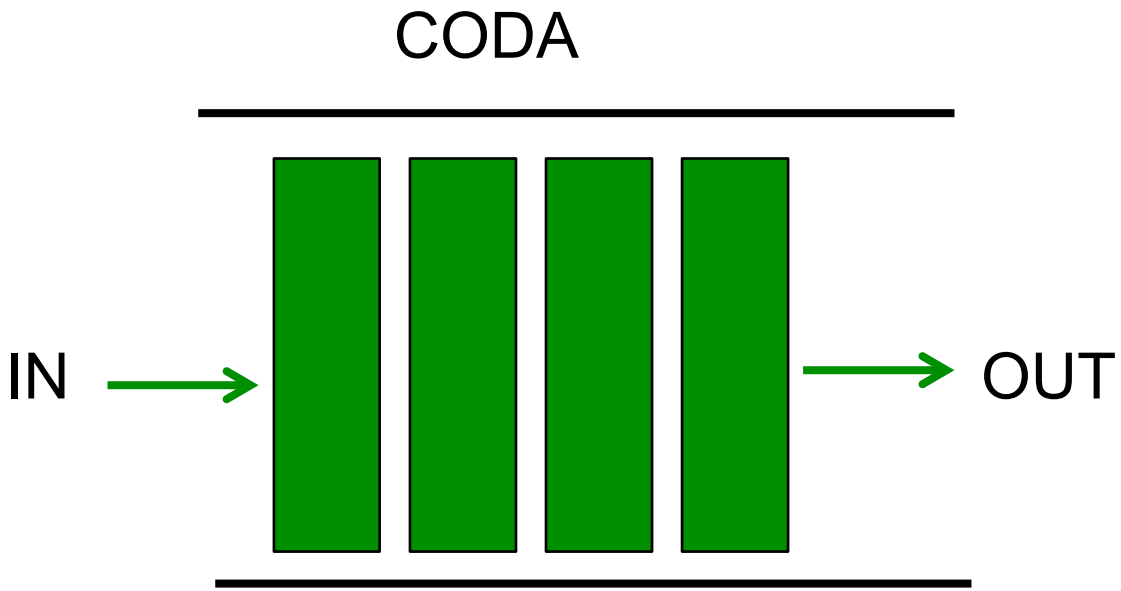
Esercizio:
Costruite una Coda analoga alla Pila

Pila e Coda





Trasformare la Pila in Coda

```
int estrai() {
        assert(marker>0):"Invalid marker";
        int retval=contenuto[0];
        for (int k=1; k<marker; k++ )
          contenuto[k-1]=contenuto[k];
        marker--;
        return retval;
    }
```

Tutto uguale, eccetto…

```
int estrai() {//la estrai di Pila
  assert(marker>0):"Invalid marker";
  return contenuto[--marker];
}
```

Che cos'é una zebra?

Trasformare la Pila in Coda

```
package strutture;
public class Coda extends Pila{
  int estrai() {
      assert(marker>0):"Invalid marker";
      int retval=contenuto[0];
      for (int k=1; k<marker; k++ )
        contenuto[k-1]=contenuto[k];
      marker--;
      return retval;
   }
```

```
int estrai() {//la estrai di Pila
  assert(marker>0):"Invalid marker";
  return contenuto[--marker];
}
```

Trasformare la Pila in Coda


```
public static void main(String args[]) {
        int dim=5;
        Coda s=new Coda();
        for (int k=0;k<2*dim;k++)
           s.inserisci(k);
        for (int k=0;k<3*dim;k++)
            System.out.println(s.estrai());
    }
}
```

Ereditierietà

La estensioni possono essere:

STRUTTURALI

(aggiunta di varabili di istanza)

e/o

COMPORTAMENTALI

(aggiunta di nuovi metodi

e/o

modifica di metodi esistenti)

subclassing & overriding


```
public class Point {
 public int x=0;
 public int y=0;
 Point(int x,int y){
    this.x=x;
    this.y=y;
 }
 public String toString(){
     return "("+x+","+y+")";
 }
 public static void main(String a[]){
     Point p=new Point(5,3);
     System.out.println(p);
   }
}
```

Output:
(5,3)

subclassing & overriding


```
public class NamedPoint extends Point {
   String name;
   public NamedPoint(int x,int y,String name) {
      super(x,y); //prima istruzione!
      this.name=name;
   }
   public String toString(){ //Overriding
      return name+" ("+x+","+y+")";
   }
   public static void main(String a[]){
      NamedPoint p=new NamedPoint(5,3,"A");
      System.out.println(p);
   }
}
```

Output:
A (5,3)

subclassing & overriding


```
public class NamedPoint extends Point {
   String name;
   public NamedPoint(int x,int y,String name) {
      super(x,y); //prima istruzione!
      this.name=name;
   }
   public String toString(){ //Overriding
      return name+super.toString();
   }
   public static void main(String a[]){
      NamedPoint p=new NamedPoint(5,3,"A");
      System.out.println(p);
   }
}
```

Output:
A (5,3)

Overloading - Overriding

Overloading:

Funzioni con uguale nome e diversa firma possono coesistere.

move(int dx, int dy)

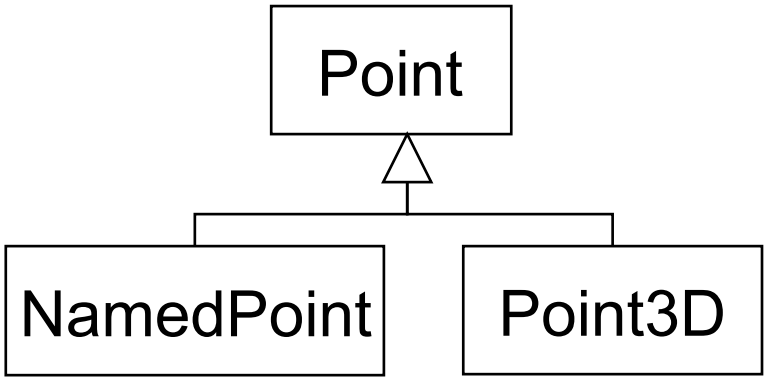
move(int dx, int dy, int dz)

Overriding:

Ridefinizione di una funzione in una sottoclasse (mantenendo immutata la firma)

Es. estrai() in Coda e Pila

Esercizio



a) Scrivere un metodo move(int dx, int dy) in Point.

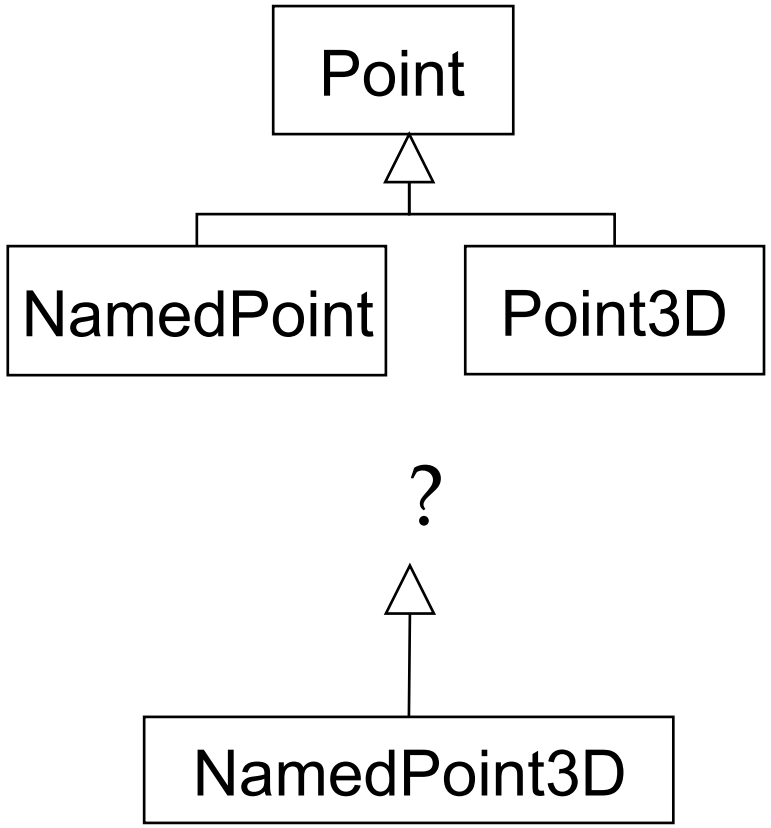
b) Estendere Point a Point3d aggiungendo una coordinata z, e fornendo un metodo move(int dx, int dy int dz) in Point3D.

Esempi

Persona – Studente - Docente

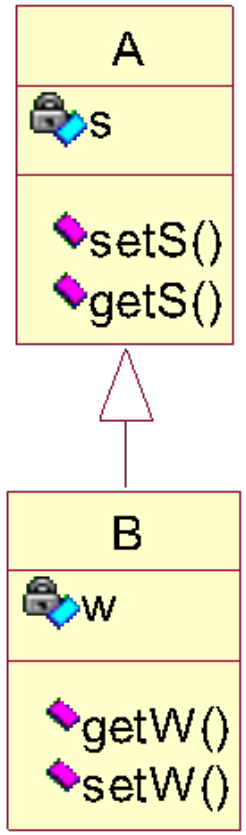
Veicolo – Auto - Moto

Problemi con l'ereditarietà



UML – Class Diagram

- rappresenta le classi e gli oggetti che compongono il sistema, ed i relativi attributi ed operazioni
- specifica, mediante le associazioni, i vincoli che legano tra loro le classi
- può essere definito in fasi diverse (analisi, disegno di dettaglio)



UML: Ereditarietà – "is"


```
class A {
    int s;
    public void setS(int) {…};
    public int getS() {…};
}
class B extends A {
    int w;
    public void setW(int) {…};
    public int getW() {…};
}
```