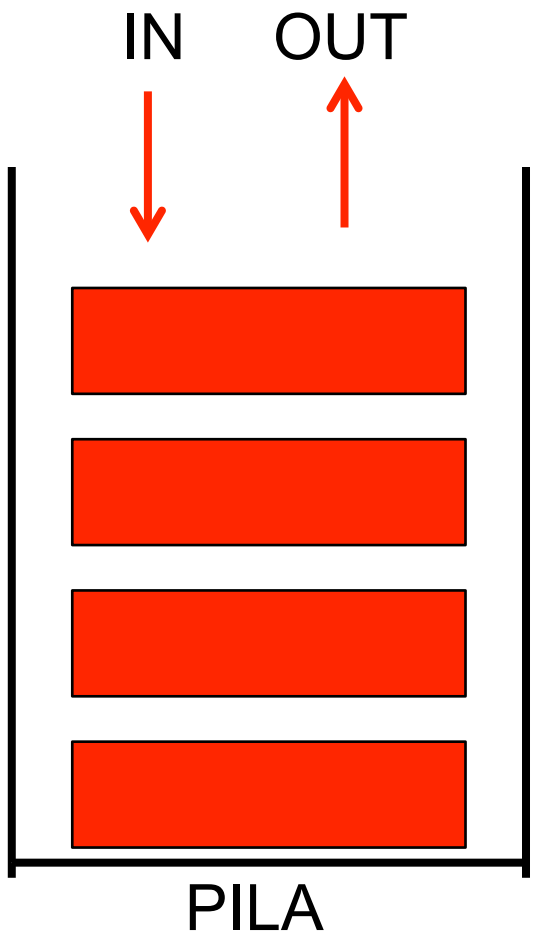
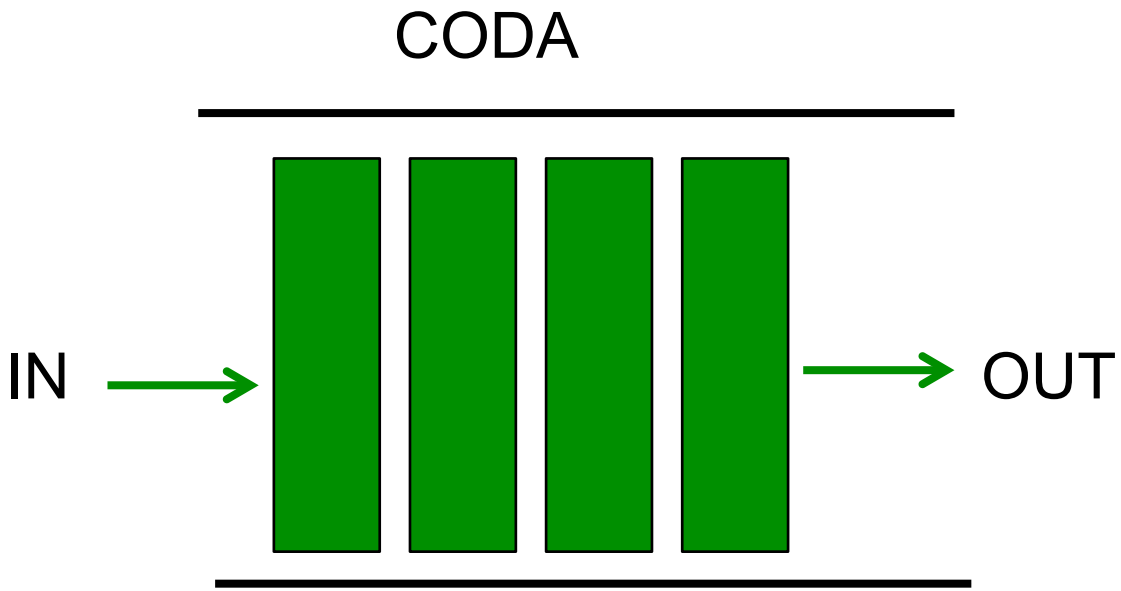
Esercizio:
Costruite una Coda analoga alla Pila

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)

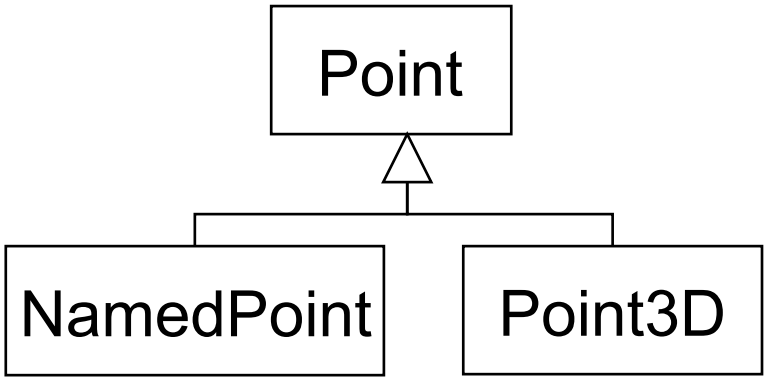
move(int dx, int dy, int dz)

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

Modificatori: abstract

Classi dichiarate abstract non possono essere istanziate, e devono essere subclassate.

Metodi dichiarati abstract devono essere sovrascritti

Una class non abstract non può contenere abstract metods

Modificatori: final

Variabili dichiarate final sono costanti.

Metodi dichiarati final non possono essere sovrascritti

Classi dichiarate final non possono essere subclassate.

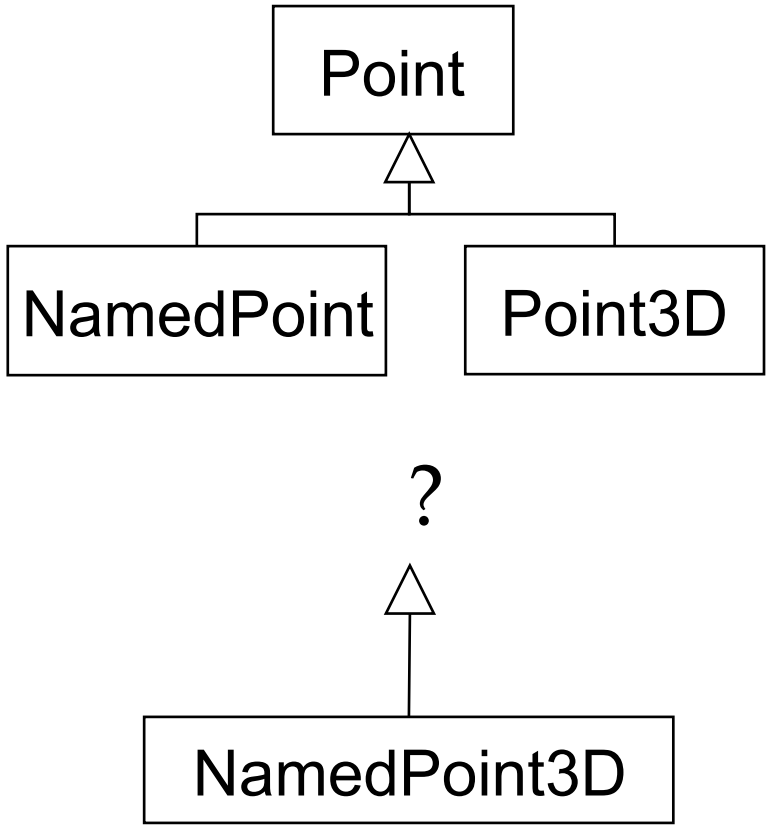
Modificatori: visibilità

public	visibile da tutti
(non def.)	visibile da tutti nello stesso package
protected	visibile dalle sottoclassi
private	nascosta da tutti

Uso di metodi "di accesso":

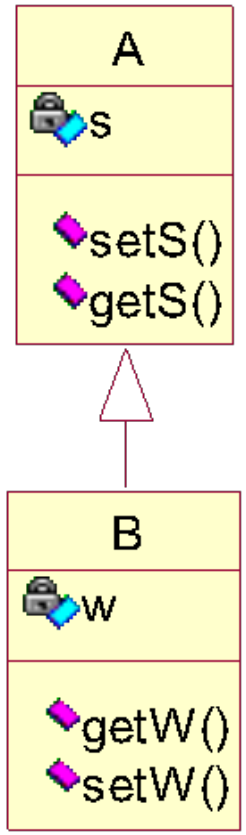
```
public class  ACorrectClass
    private String  aUsefulString;
    public String  getAUsefulString() {
        return aUsefulString; // "get" the value
    }
    private void setAUsefulString(String s) {
    //protected void setAUsefulString(String s) {
       aUsefulString = s; // "set" the value

   }
}
```

Problemi con l'ereditarietà



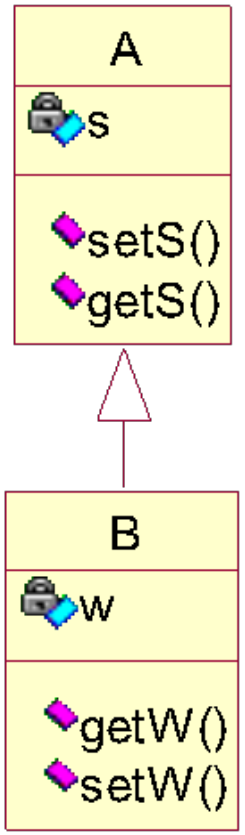
UML – Class Diagram

- 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)



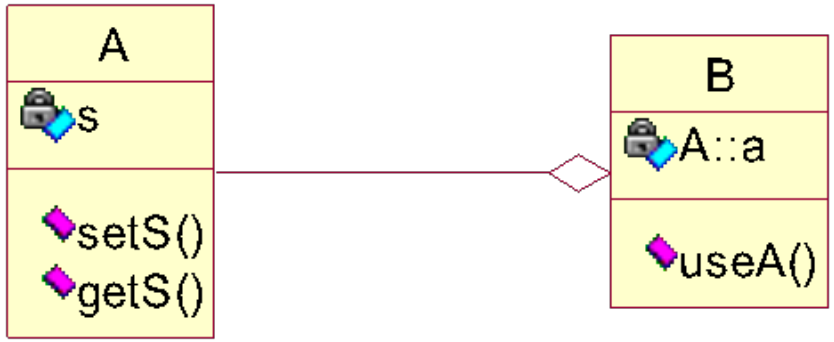
UML: Ereditarietà – "is"


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



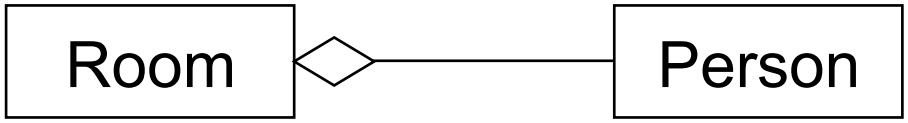
UML: Aggregazione


```
class A {
    int s;
    public void setS(int){…};
    public int getS() {…};
}
class B {A ob;
    public void useA() {…};
}
```

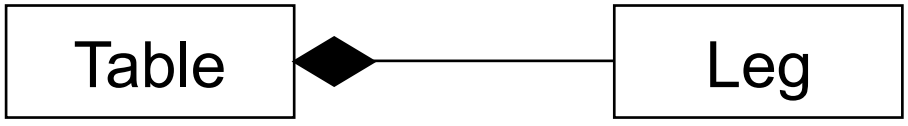


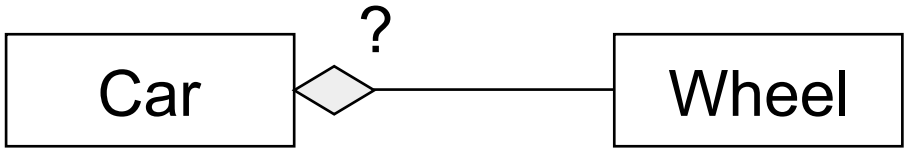
Aggregation - Composition

Use *aggregation (has-a)* when the lifecycle of the partecipating elements is different (one can exist without the other).

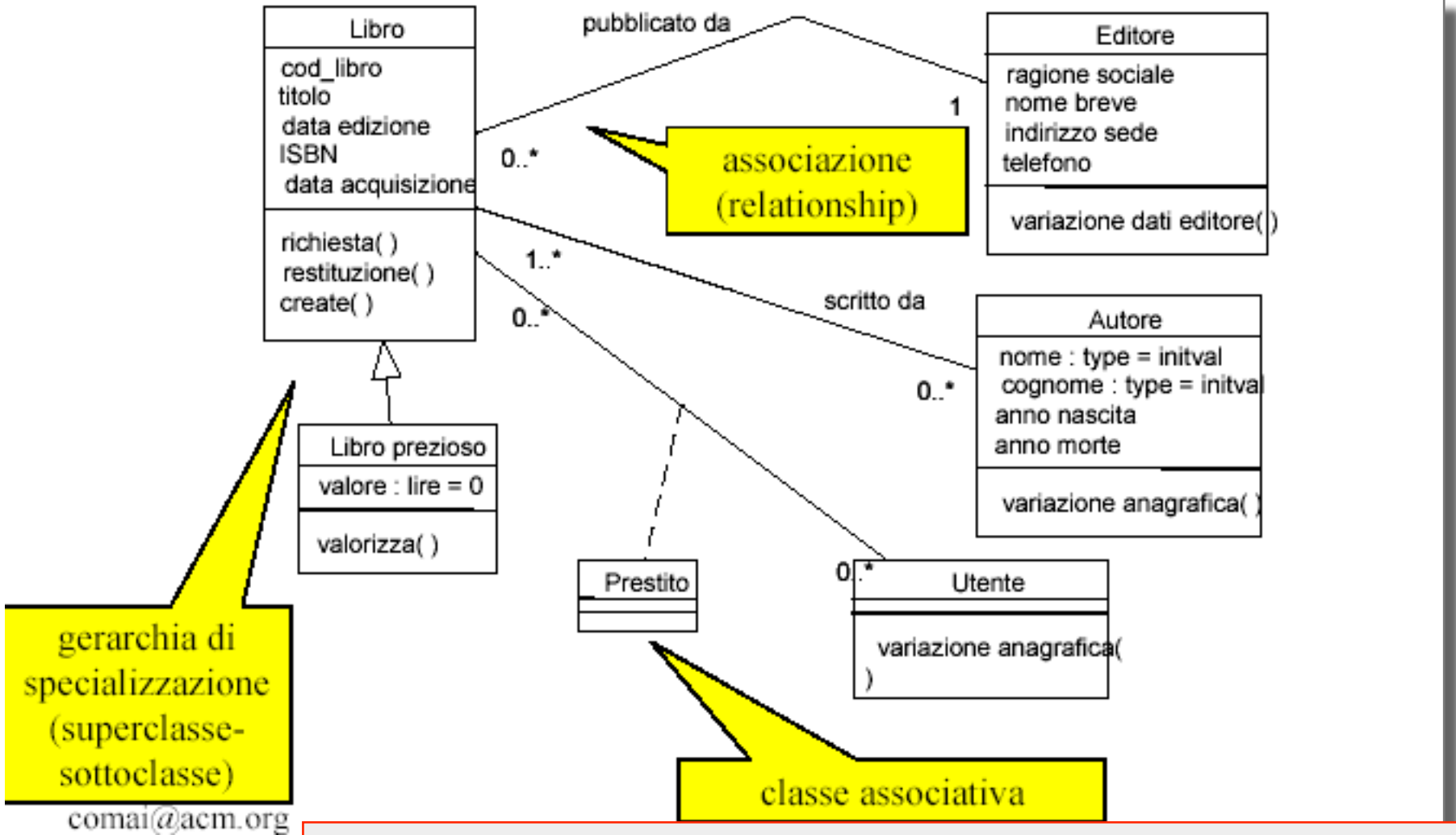


Use *composition (part-of)* when the *container* cannot be conceived without the *contained*.



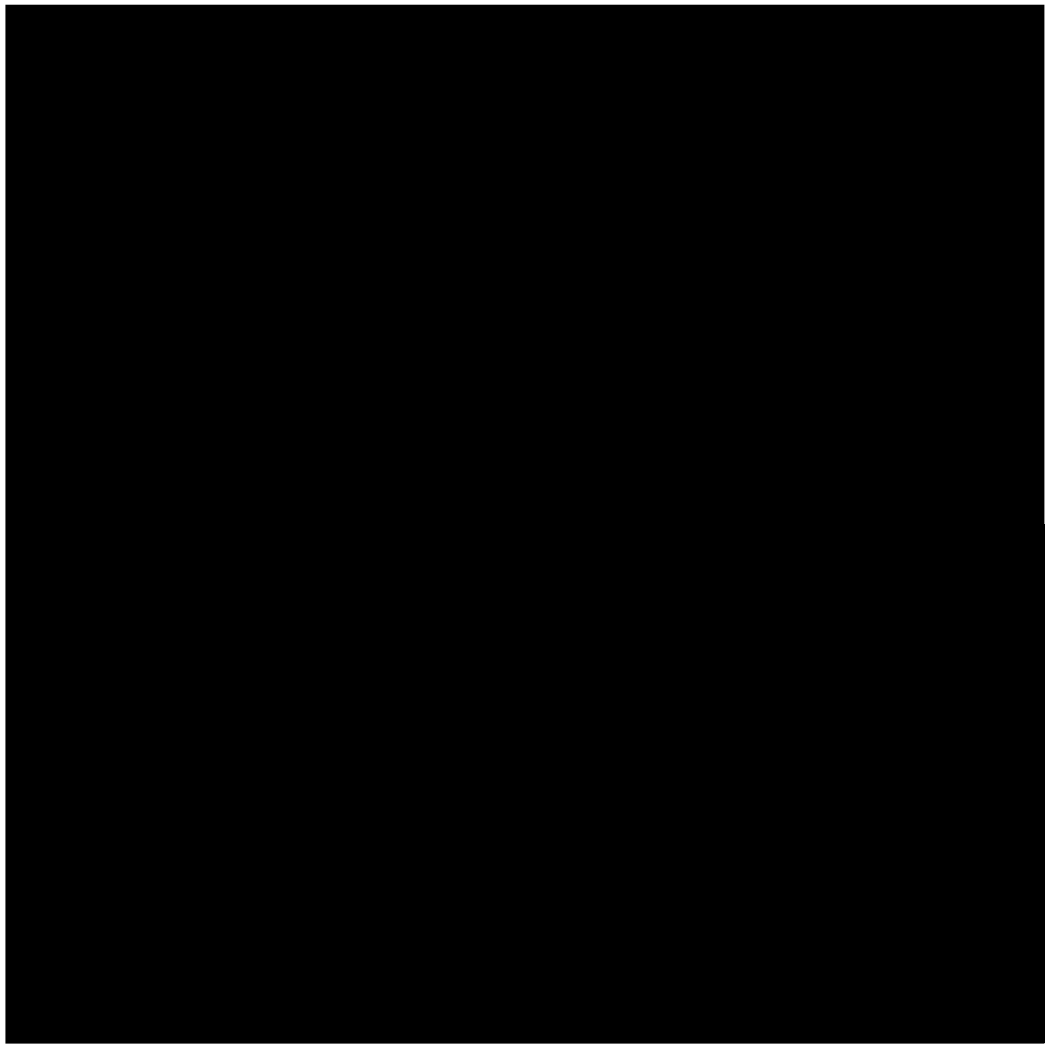


UML – Class Diagram

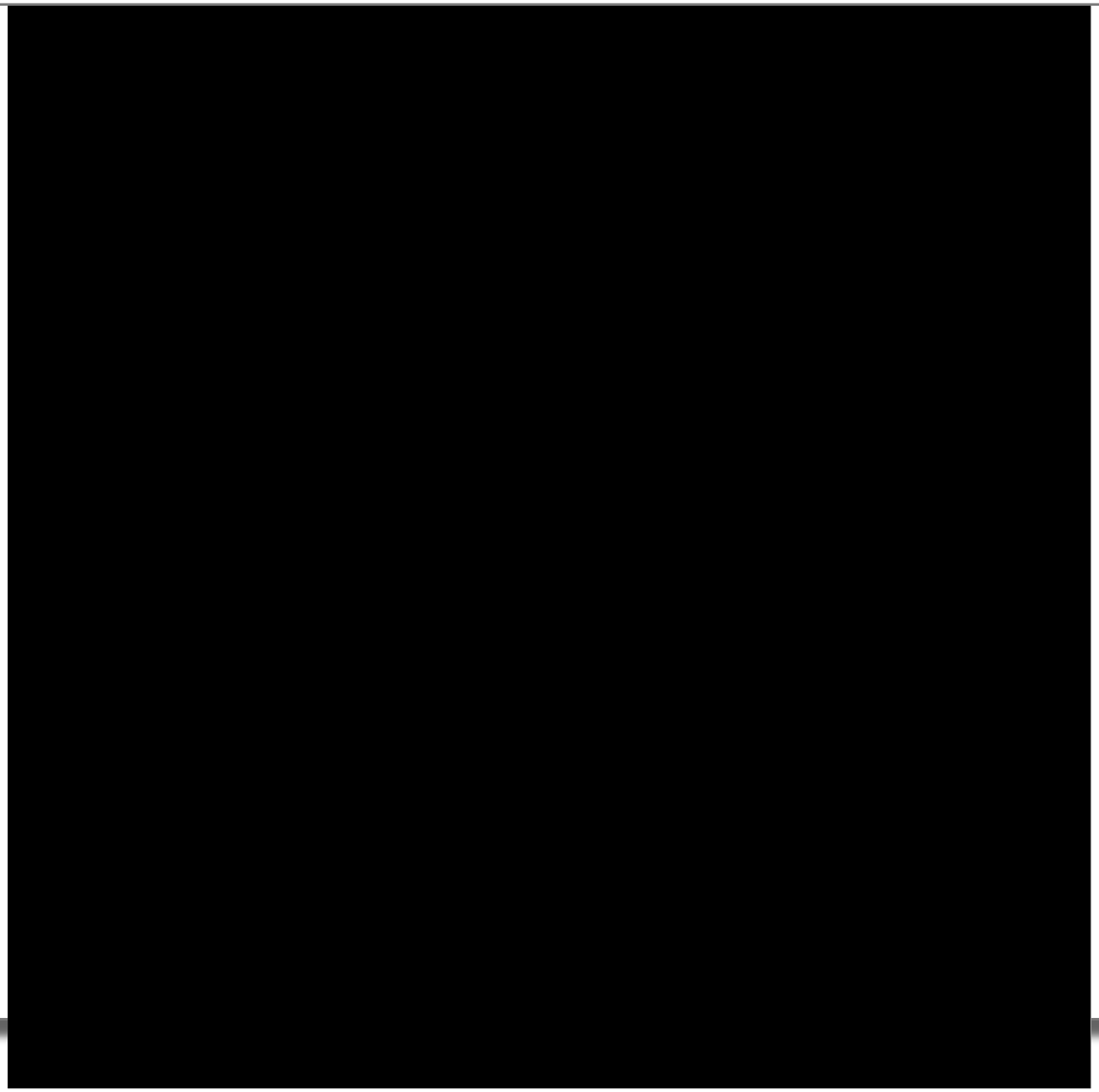


Disegno ripreso da: Adriano Comai
http://www.analisi-disegno.com/a_comai/corsi/sk_uml.htm

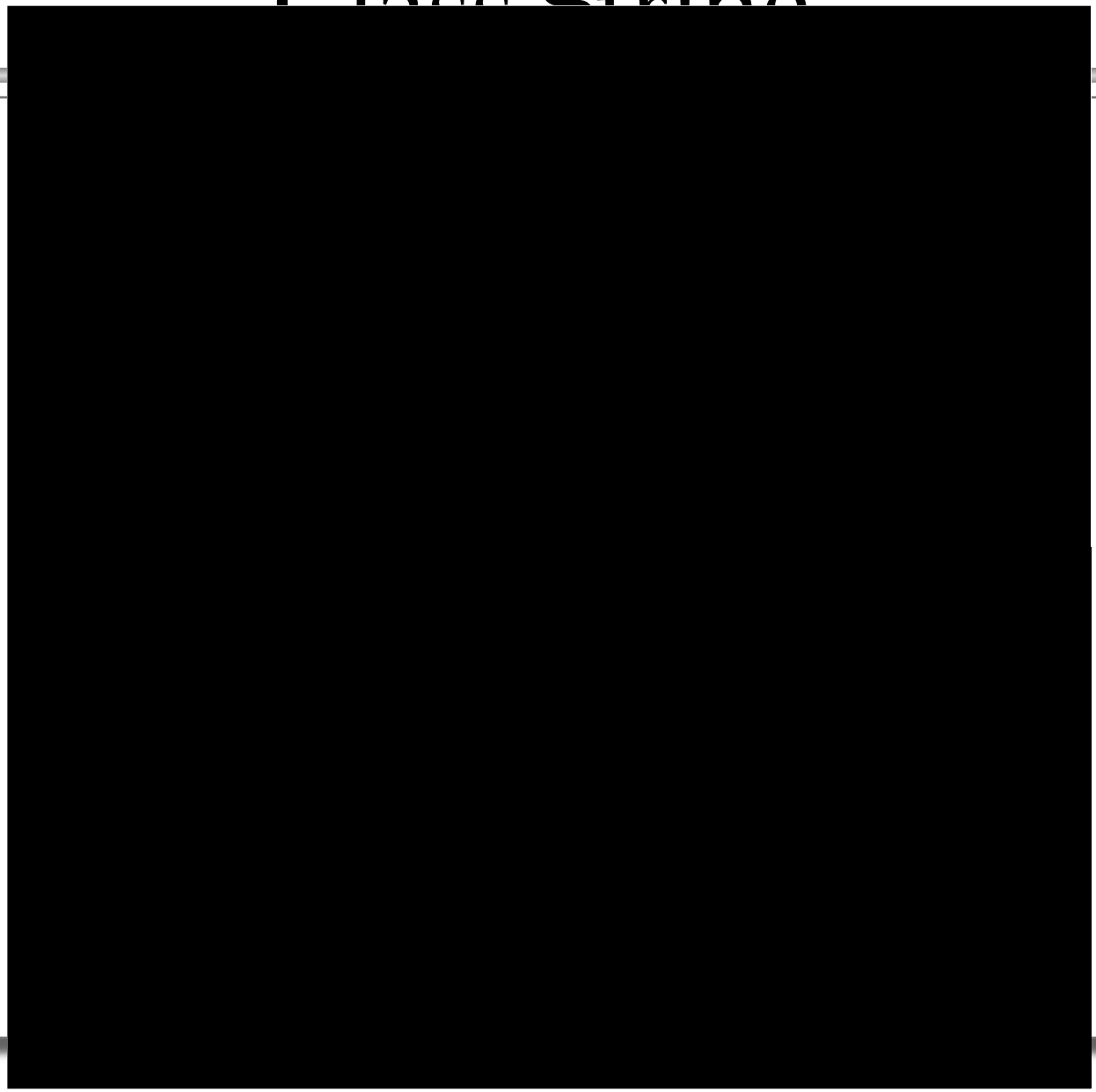
Class String



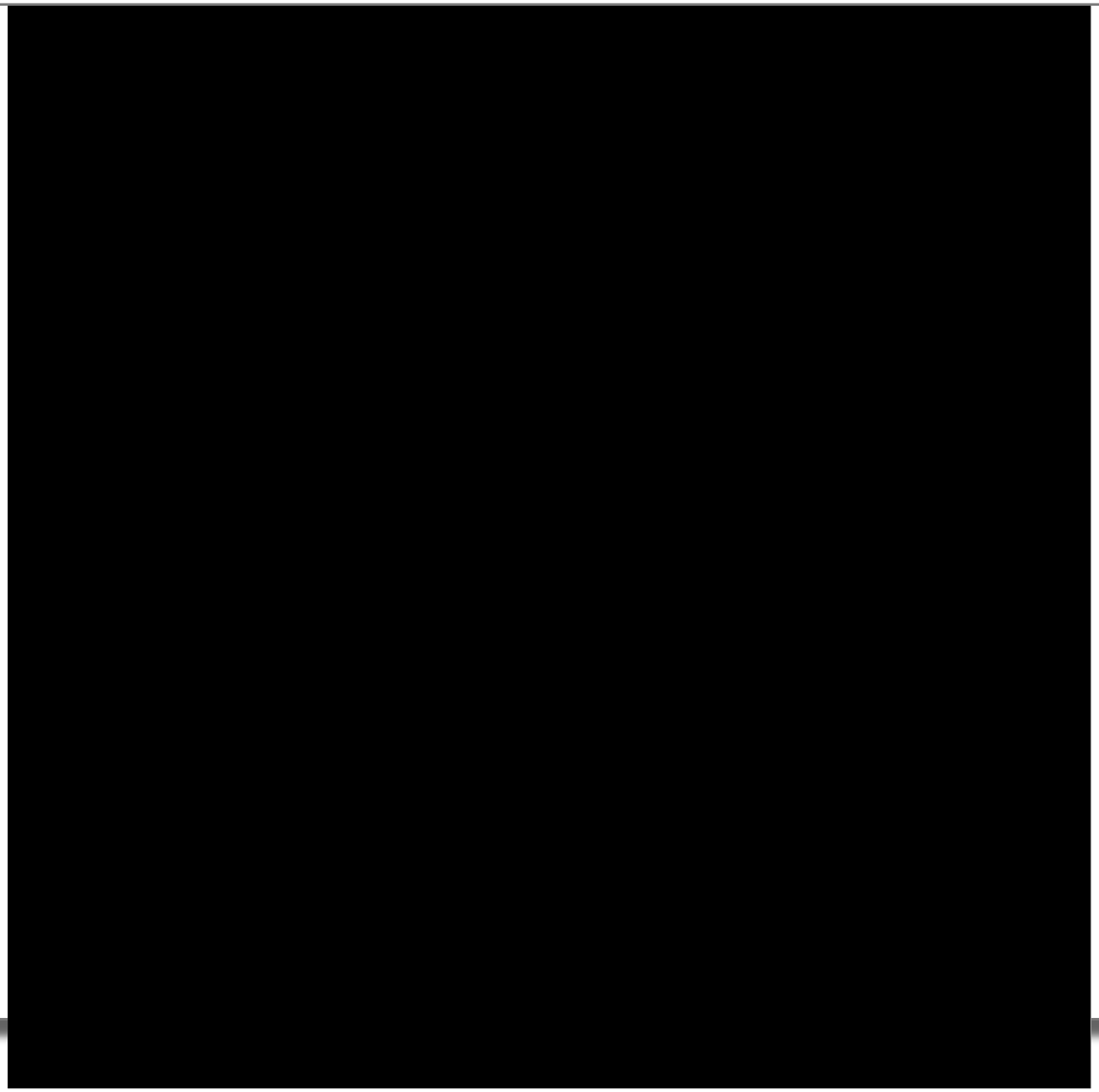
Class String



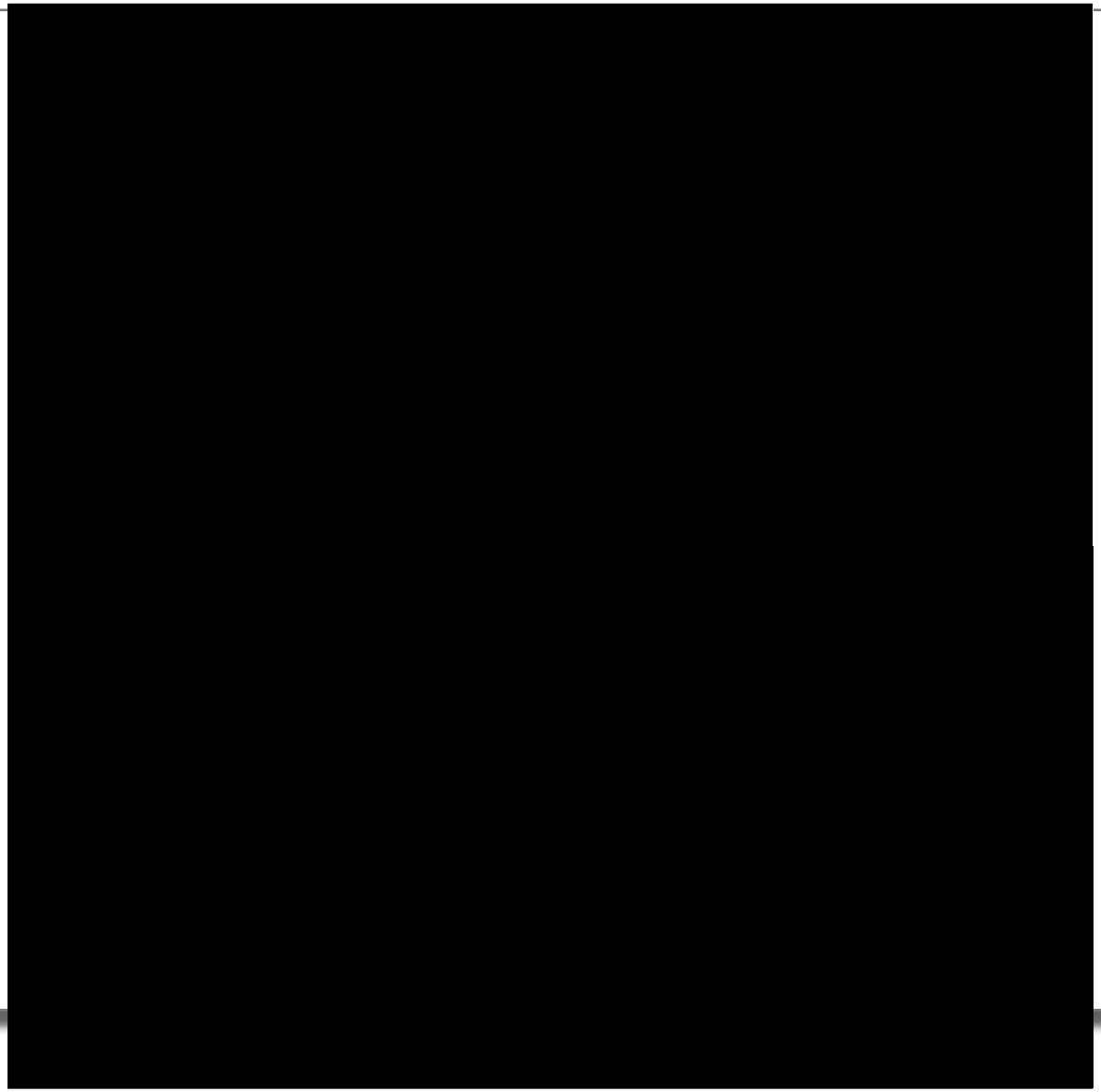
Class String



Class String



Class String



String

Per trasformare il contenuto di una stringa in un intero:

int Integer.parseInt(String s)

Per trasformare il contenuto di una stringa in un float:

float Float.parseFloat(String s)

Lettura di stringhe

```
import java.io.*;
public class A {
  public A()  {
    BufferedReader s = new BufferedReader(
            new InputStreamReader(System.in));
    try {
       System.out.println("Dammi una stringa");
       String str=s.readLine();
       System.out.println("Hai scritto "+str);
    }catch (IOException e) {e.printStackTrace();
    }
  }
  public static void main(String [] ar) {
    A a=new A();
  }
}
```

```
Dammi una stringa
abracadabra
Hai scritto abracadabra
```

Lettura di int

```
public A()  {
     int i=0;
     BufferedReader s = new BufferedReader(
             new InputStreamReader(System.in));
     try {
        System.out.println("Dammi un intero");
        i=Integer.parseInt(s.readLine());
        System.out.println("Hai scritto "+i);
     }catch (Exception e) {e.printStackTrace();}
}
```

```
Dammi un intero
2
Hai scritto 2
```


```
public A()  {
    int i=0;
    BufferedReader s = new BufferedReader(
            new InputStreamReader(System.in));
    try {
       System.out.println("Dammi un intero");
       i=Integer.parseInt(s.readLine());
       System.out.println("Hai scritto "+i);
    }catch (IOException e) {e.printStackTrace();}
}
```

```
Dammi un intero
pippo
java.lang.NumberFormatException: For input string: "gh"
  at
java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
  at java.lang.Integer.parseInt(Integer.java:426)
  at java.lang.Integer.valueOf(Integer.java:532)
  at pila.A.<init>(A.java:11)
  at pila.A.main(A.java:19)
Exception in thread "main"
```

```
public A()  {
    float f=0; boolean error;
    BufferedReader s = new BufferedReader(
            new InputStreamReader(System.in));
    try {
        do {
           System.out.println("Dammi un float");
           try{
              error=false;
              f=Float.parseFloat(s.readLine());
           } catch (NumberFormatException e) {
              error=true;
              System.out.println("Input non valido");
           }
        } while (error);
        System.out.println("Hai scritto "+f);
    }catch (IOException e) {e.printStackTrace();}
}
```

```
Dammi un float
pippo
Input non valido
Dammi un float
3
Hai scritto 3.0
```

Lettura di stringhe con GUI


```
import javax.swing.JOptionPane;
public A()  {
    ...
    String input = JOptionPane.showInputDialog(
      "How are you?");
    System.out.println(input);
    System.exit(1);
}
```

Essenziale!
Altrimenti la thread che gestisce la GUI rimane viva, e il processo non termina

I parametri del main sono inclusi in un vettore di String

Parametri di ingresso


```
/* sum and average command lines */
class SumAverage {
  public static void main (String args[]) {
    int sum = 0;
    float avg = 0;
    for (int i = 0; i < args.length; i++) {
      sum += Integer.parseInt(args[i]);
    }
    System.out.println("Sum is: " + sum);
    System.out.println("Average is: "
        + (float)sum / args.length);
  }
}
```