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,"z");
    System.out.println(p);
  }
}
```

Output:
z (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,"z");
    System.out.println(p);
  }
}
```

Output:
z (5,3)

Ereditierietà

La estensioni possono essere:

STRUTTURALI

(aggiunta di varabili di istanza)

e/o

COMPORTAMENTALI

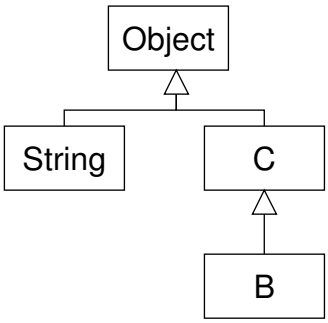
(aggiunta di nuovi metodi

e/o

modifica di metodi esistenti)

Unified Modeling Language

Esiste una notazione grafica per mostrare le relazioni di ereditarietà.



class C {…}
class B extends C {…}

Tutte le classi ereditano da Object!

ttori

```
public class A {
  public A() {
    System.out.println("Creo A");
  }
}
public class B extends A {
  public B() {
    System.out.println("Creo B");
  }
  public B(int k) {
    System.out.println("Creo B_int”);
  }
}
```

```
public static void main(String [] a) {
  B b=new B();
}
```

Output:
Creo A
Creo B

ttori

```
public class A {
  public A() {
    System.out.println("Creo A");
  }
}
public class B extends A {
  public B() {
    System.out.println("Creo B");
  }
  public B(int k) {
    System.out.println("Creo B_int”);
  }
}
```

```
public static void main(String [] a) {
   B b=new B(1);
  }
```

Output:
Creo A
Creo B_int

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

Modificatori: final

Variabili dichiarate final sono costanti.

Metodi dichiarati final non possono essere sovrascritti

Classi dichiarate final non possono essere subclassate.

La Pila in Java - 1


```
package strutture;
public class Pila {
  int size;
  int defaultGrowthSize;
  int marker;
  int contenuto[];
  final int initialSize=3;

  Pila() {
     size=initialSize;
     defaultGrowthSize=initialSize;
     marker=0;
     contenuto=new int[size];
  }
```

La Pila in Java - 1

```
package strutture;
public class Pila {
  int size;
  int                 hSize;
  int m    er;
  int contenuto[];
  final int initialSize=3;

  Pila() {
     size=initialSize;
     defaultGrowthSize=initialSize;
     marker=0;
     contenuto=new int[size];
  }
```

costante

La Pila in Java - 2

```
void inserisci(int k) {
     if (marker==size){
         cresci(defaultGrowthSize;)}
     contenuto[marker]=k;
     marker++;
  }

  int estrai() {
    if (marker==0) {
        System.out.println(
        "Non posso estrarre da una pila vuota");
        System.exit(1);
    }
    return contenuto[--marker];
  }
```

La Pila in Java - 3


```
  private void cresci(int dim){
    int temp[]=new int[size+
defaultGrowthSize];
    for (int k=0;k<size;k++)
      temp[k]=contenuto[k];

    contenuto=temp;

    size+=defaultGrowthSize;
  }
```


La Pila in Java - 4


```
public static void main(String args[]) {
      int dim=10;
      Pila s=new Pila(dim);
      for (int k=0;k<2*dim;k++)
          s.inserisci(k);
      for (int k=0;k<3*dim;k++)
         System.out.println(s.estrai());
  }

}
```

Using assertions (da Java 1.4)


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

Compilare con:
java –ea Pila.java

java.lang.AssertionError: Invalid marker
at pila.Pila.estrai(Pila.java:22)
at pila.Pila.main(Pila.java:39)

Using System.arrayCopy()


```
System.arraycopy(
  Object src, int src_position,
  Object dst, int dst_position, int length)
```

Copies the specified source array, beginning at the specified position, to the specified position of the destination array.

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());
    }
}
```