Interfacce e polimorfismo

La classe DataSet

```
/**
   Serve a computare la media di
     un insieme di valori numerici
*/
public class DataSet {
  /**
     Costruisce un insieme vuoto
  */
     public DataSet() {
            sum = 0;
            count = 0;
            minimum = 0;
            maximum=0;
     }
```

```
/**
   Aggiunge il valore di un dato
     all'insieme
     @param x : valore di un dato
  */
  public void add(double x) {
     sum += x;
     if (count == 0 || minimum > x)
        minimum = x;
     if (count == 0 || minimum < x)
        maximum = x;
     count++;
  }
```

La classe DataSet

```
/**
    Restituisce la media dei dati
    @return la media o 0 se nessun
    dato è stato aggiunto
 */
public double getAverage() {
   if (count == 0) return 0;
   else return sum / count;
}

/**
   Restituisce il piu` grande dei dati
   @return il massimo o 0 se nessun
   dato e` stato aggiunto
*/
public double getMaximum() {
   return maximum;
}
```

```
/**
    Restituisce il piu` piccolo dei dati
    @return il minimo o 0 se nessun
    dato è stato aggiunto
 */
public double getMinimum(){
   return minimum;
}

private double sum;
private double minimum;
private  double maximum;
private int count;
}
```


La classe DataSetTest

```
import java.io.*;
public class DataSetTest{
   public static void main(String[] args) throws IOException {
         String input;
         InputStreamReader reader = new  InputStreamReader(System.in);
         BufferedReader console = new BufferedReader(reader);
         DataSet ds = new DataSet();

         while (( input =   console.readLine()).length() !=0){
                  double d = Double.parseDouble(input);
                  ds.add(d);
         }
         System.out.println("la media e`:"+ ds.getAverage());
    }
}
```

Scrivere codice riutilizzabile

- Supponiamo ora di voler calcolare la media dei saldi di un insieme di conti bancari
  - Dobbiamo modificare la classe DataSet in modo che funzioni con oggetti di tipo BankAccount

La classe DataSet per i conti correnti

```
/**
   Serve a computare la media dei
    saldi di un insieme di conti
    correnti.
*/

public class DataSet {
   /**
      Costruisce un insieme vuoto
   */
   public DataSet() {
      sum = 0;
      count = 0;
      minimum = null;
      maximum= null;
   }
```

```
/** Restituisce la media dei saldi dei
    conti correnti
*/
  public double getAverage()
   {
      if (count == 0) return 0;
      else return sum / count;
   }

   /**Restituisce il conto con il saldo
     più grande
 */
   public BankAccount getMaximum()
   {
      return maximum;
   }
```

La classe DataSet per i conti correnti

```
// Restituisce il conto con il saldo più piccolo
  public BankAccount getMinimum () {  return minimum; }
// Aggiunge un conto corrente
  public void add(BankAccount x) {
     sum = sum  + x.getBalance();
     if (count == 0 || minimum.getBalance() > x.getBalance())     minimum = x;
     if (count == 0 || maximum.getBalance() < x.getBalance())    maximum = x;
     count++;
  }
  private double sum;
  private BankAccount minimum;
  private BankAccount maximum;
  private int count;
}
```


Scrivere codice riutilizzabile

- Supponiamo ora di voler calcolare la media dei valori di un insieme di monete
  - Dobbiamo modificare di nuovo la classe DataSet in modo che funzioni con oggetti di tipo Coin

La classe DataSet per le monete

```
/**
 Serve a computare la media dei
    valori di un insieme di monete
*/

public class DataSet {
   /**
      Costruisce un insieme vuoto
   */
   public DataSet() {
      sum = 0;
      count = 0;
      minimum = null;
      maximum= null;
   }
```

```
/** Restituisce la media dei valori
    delle monete
*/
  public double getAverage()
   {
      if (count == 0) return 0;
      else return sum / count;
   }

   /**Restituisce una moneta con il
      valore più grande
 */
   public Coin getMaximum()
   {
      return maximum;
   }
```


La classe DataSet per i conti correnti

```
 //   Restituisce una moneta con il valore più piccolo
   public Coin getMinimum() {  return minimum; }
//  Aggiunge una moneta
   public void add(Coin x) {
      sum = sum  + x.getValue();
      if (count == 0 || minimum.getValue() > x.getValue())     minimum = x;
      if (count == 0 || maximum.getValue() < x.getValue())     maximum = x;
      count++;
   }
   private double sum;
   private Coin minimum;
   private Coin maximum;
   private int count;
}
```

Scrivere codice riutilizzabile

- Le classi DataSet per
  - i valori numerici
  - i conti correnti
  - le monete

  …differiscono solo per la misura usata nell'analisi dei dati:
  - DataSet per double usa il valore dei dati
  - DataSet per oggetti di tipo BankAccount usa il valore dei saldi
  - DataSet per oggetti di tipo Coin usa il valore delle monete

Scrivere codice riutilizzabile

- Supponiamo che esista un metodo **getMeasure** che fornisce la grandezza da usare nell'analisi dei dati
  - **Esempio:**

    x.getMeasure();
    - se x è un double, restituisce il valore di x
    - se x è un conto, restituisce il saldo del conto
    - se x è una moneta, restituisce il valore della moneta
- Possiamo implementare un'unica classe DataSet riutilizzabile!

Scrivere codice riutilizzabile

- Abbiamo bisogno di una classe che fornisca il metodo getMeasure
- Il comportamento di getMeasure varia a seconda di ciò che rappresenta realmente l'oggetto (double, conto, moneta,...)
- Non è quindi possibile scrivere un'implementazione unica di getMeasure che vada bene per tutti gli oggetti

Le interfacce

- Un'interfaccia dichiara una collezione di metodi e le loro firme (con tipo del valore restituito) ma non fornisce alcuna implementazione dei metodi
- Es.: l'interfaccia che dichiara il metodo getMeasure è

```
public interface Measurable{
  double getMeasure();
}
```

Differenze tra classi e interfacce

- Tutti i metodi di un'interfaccia sono *astratti*, cioè non hanno un'implementazione
- Tutti i metodi di un'interfaccia sono automaticamente public
- Un'interfaccia non ha variabili di istanza
  - Esistono variabili del tipo di un'interfaccia ma non esistono istanze di un interfaccia
  - Una variabile del tipo di un'interfaccia può contenere istanze delle classi che implementano l'interfaccia

La classe DataSet per le monete

```
/**
 Serve a computare la media di un
    insieme di valori
*/

public class DataSet {
   /**
      Costruisce un insieme vuoto
   */
   public DataSet() {
      sum = 0;
      count = 0;
      minimum = null;
      maximum= null;
   }
```

```
// Restituisce la media dei valori

  public double getAverage()
   {
      if (count == 0) return 0;
      else return sum / count;
   }

   /**Restituisce un oggetto
     Measurable con il valore più
     grande
 */
    public Measurable getMaximum()
    {
       return maximum;
    }
```

La classe DataSet per i conti correnti

```
//  Restituisce un oggetto Measurable con il valore più piccolo
  public Measurable getMinimum () {  return minimum; }
//  Aggiunge un oggetto Measurable
  public void add(Measurable x) {
     sum = sum  + x.getMeasure();
     if (count == 0 || minimum.getMeasure() > x.getMeasure())     minimum = x;
     if (count == 0 || maximum.getMeasure() < x.getMeasure())     maximum = x;
     count++;
  }
  private double sum;
  private  Measurable minimum;
  private  Measurable maximum;
  private int count;
}
```


Classi che implementano l'interfaccia

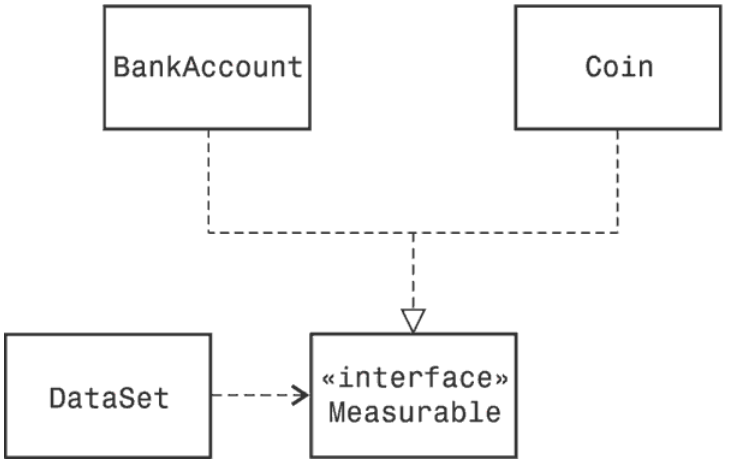
- La nuova classe DataSet può essere usata per analizzare oggetti di qualsiasi classe che realizza l'interfaccia Measurable
- Una classe *realizza* (implementa) un'interfaccia se fornisce l'implementazione di tutti i metodi dichiarati nell'interfaccia
  - Corrispondenza con i metodi dell'interfaccia data dai prototipi dei metodi
  - Può contenere metodi non dichiarati nell'interfaccia
- **Esempio:**

```
public class BankAccount implements Measurable {
  public double getMeasure()    {
        return balance;
   }
  …// tutti gli altri metodi di BankAccount
}
```

Classi che implementano l'interfaccia

- Allo stesso modo posso scrivere la classe Coin che implementa l'interfaccia Measurable

```
public class Coin implements Measurable {
    public double getMeasure()    {
        return value;
    }
    …// tutti gli altri metodi di Coin
}
```

Schema UML della classe `DataSet` che dipende da `Measurable` e delle classi che implementano l'interfaccia `Measurable`

La classe DataSetTest

```
/**
   Questo programma collauda la classe DataSet per i conti correnti
*/
public class DataSetTest
{
   public static void main(String[] args) {
      DataSet ds = new DataSet();

      ds.add(new BankAccount(0));
      ds.add(new BankAccount(10000));
      ds.add(new BankAccount(2000));
```

Continua nella prossima slide

La classe DataSetTest

```
      System.out.println("Saldo medio = "+ ds.getAverage());

      Measurable max = ds.getMaximum();
      System.out.println("Saldo piu` alto = "+ max.getMeasure());

      DataSet coinData = new DataSet();

      coinData.add(new Coin(0.25, "quarter"));
      coinData.add(new Coin(0.1, "dime"));
      coinData.add(new Coin(0.05, "nickel"));

      System.out.println("Valore medio delle monete = "+coinData.getAverage());
      max = coinData.getMaximum();
      System.out.println("Valore max delle monete = "+ max.getMeasure());
   }
}
```

Conversione fra tipi

- E` possibile convertire dal tipo di una classe al tipo dell'interfaccia implementata dalla classe
- Esempi:

```
ds.add(new BankAccount(100));
```

  - il tipo BankAccount dell' argomento è convertito nel tipo Measurable del parametro del metodo add

```
BankAccount b = new BankAccount(100);
Coin c = new Coin(0.1,"dime");
Measurable x = b; // x si riferisce ad un oggetto di tipo BankAccount
 x = c;   //ora x si riferisce ad un oggetto di tipo Coin
```

  - Possiamo assegnare ad una variabile di tipo Measurable un oggetto di una qualsiasi classe che implementa Measurable

Conversione fra tipi

- Ovviamente non e` possibile convertire dal tipo di una classe al tipo di un'interfaccia che NON e` implementata da quella classe
  - **Esempio:**
    - Measurable r = new Rectangle(1,2,5,3);
      // errore: Rectangle non implementa Measurable

Conversione fra tipi

- Per convertire un tipo interfaccia in un tipo classe occorre un cast
  - Esempio:

```
BankAccount b = new BankAccount(100));
Measurable x = b;
BankAccount account = (BankAccount) x;
```


Conversione fra tipi

- Consideriamo la seguente istruzione:

```
Measurable max = bankData.getMaximum();
 // l'oggetto di tipo BankAccount restituito da getMaximum
// e` convertito nel tipo Measurable
```

- Anche se max si riferisce ad un oggetto che in origine è di tipo BankAccount, non e` possibile invocare il metodo **deposit** per max
  - **Esempio:** max.deposit(35); // ERRORE
- Per poter invocare i metodi di BankAccount che non sono contenuti nell'interfaccia Measurable si deve effettuare il cast dell'oggetto al tipo BankAccount
  - **Esempio:**

```
BankAccount acc= (BankAccount ) max;
acc.deposit(35);  //OK
```

Conversione fra tipi

- E` possibile effettuare il cast di un oggetto ad un certo tipo solo se l'oggetto in origine era di quel tipo
  - Esempio:

```
BankAccount b = new BankAccount(100));
Measurable x = b;
Coin c = (Coin) x; /* errore che provoca un'eccezione: il tipo originale
                      dell'oggetto a cui si riferisce x
                      non e`Coin ma BankAccount */
```


Conversione fra tipi

- L'operatore **instanceof** permette di verificare se un oggetto appartiene ad un determinato tipo
- Al fine di evitare il lancio di un'eccezione, prima di effettuare un cast di un oggetto ad un certo tipo classe possiamo verificare se l' oggetto appartiene effettivamente a quel tipo classe
  - **Esempio:**

```
if (x instanceof Coin ){
 Coin c = (Coin) x;
  … }
```

Polimorfismo

Measurable x;
x = new ... (BankAccount OR Coin)

```
double i = x.getMeasure();
```

- Quale metodo getMeasure viene invocato?
  - Le classi BankAccount e Coin forniscono due diverse implementazioni di getMeasure!
- La JVM utilizza il metodo getMeasure() della classe a cui si riferisce l'oggetto.

Polimorfismo

- L'invocazione

```
double i = x.getMeasure();
```

  può chiamare metodi diversi a seconda del tipo reale dell'oggetto x
- Il metodo getMeasure() viene detto polimorfico o multiforme
- Realizzato in Java attraverso:
  - Uso di interfacce
  - Ereditarietà -- Overriding (prossime lezioni)
- Altro caso di polimorfismo in senso lato
  - Overloading --- metodi sono distinti da parametri espliciti

Polimorfismo vs Overloading

- Entrambi invocano metodi distinti con lo stesso nome, ma...
  - Con l'overloading scelta del metodo appropriato avviene in fase di compilazione, esaminando il tipo dei parametri
    - early binding, effettuato dal compilatore
  - Con il polimorfismo avviene in fase di esecuzione
    - late binding, effettuato dalla JVM

Riutilizzo di codice: problema 1

- Se vogliamo utilizzare il metodo getMeasure() per misurare oggetti di tipo Rectangle, come facciamo?
  - Non possiamo riscrivere la classe Rectangle in modo che implementi l'interfaccia Measurable
    - E' una classe di libreria: non abbiamo i permessi!!

Riutilizzo di codice: problema 2

- Sappiamo misurare un oggetto in base ad un unico parametro
  - saldo, valore moneta, etc..
- Come facciamo a misurare un oggetto in base a due parametri?
  - un rettangolo con perimetro ed area
  - c/c bancario con saldo e tasso di interesse

Interfacce strategiche

- Mettono in atto una particolare strategia di elaborazione
  - Con

```
public interface Measurable{
   double getMeasure();
}
```

misurazione demandata all'oggetto stesso

  - Con

```
public interface Measurer{
  double measure(Object anObject);
// restituisce la misura dell'oggetto anObject
 }
```

misurazione implementata in una classe dedicata
(Interfaccia strategica)

Misurazione dell'area dei rettangoli

```
class RectangleMeasurer implements Measurer
{
  public double measure(Object anObject)
       {
         Rectangle aRectangle = (Rectangle) anObject;
         double area =
           aRectangle.getWidth() * aRectangle.getHeight();
         return area;
       }
}
```


Nota

- Il prototipo del metodo measure della nostra classe deve essere lo stesso del metodo omonimo nell'interfaccia Measurer
  - measure deve accettare un parametro di tipo Object
  - Vogliamo misurare rettangoli!
  - Occorre un cast per convertire il parametro di tipo Object in un Rectangle

    Rectangle aRectangle = (Rectangle) anObject;

Soluzione ai problemi: interfacce strategiche

- La nuova classe DataSet viene costruita con un oggetto di una classe che realizza l'interfaccia Measurer
- Tale oggetto viene memorizzato nella variabile istanza measurer ed è usato per eseguire le misurazioni

La classe DataSet con l'oggetto Measurer

```
/**
 Serve a computare la media di un
    insieme di valori
*/

public class DataSet {
   /**
      Costruisce un insieme vuoto
   */
   public DataSet(Measurer M){
      sum = 0;
      count = 0;
      minimum = null;
      maximum = null;
      measurer = M;
   }
```

```
// Restituisce la media dei valori

  public double getAverage()
  {
     if (count == 0) return 0;
     else return sum / count;
  }

   /**Restituisce un oggetto con il
     valore più grande
 */
   public Object getMaximum()
   {
      return maximum;
   }
```

La classe DataSet con l'oggetto Measurer

```
 //   Restituisce un oggetto con il valore più piccolo
   public Object getMinimum () {  return minimum; }
 //  Aggiunge un oggetto
   public void add(Object x) {
      sum = sum  + measurer.measure(x);
      if (count == 0 || measurer.measure(minimum ) > measurer.measure(x))
            minimum = x;
      if (count == 0 || measurer.measure(maximum ) < measurer.measure(x))
            maximum = x;
      count++;
    }
   private double sum;                  private Object minimum;
   private  Object maximum;             private int count;
   private  Measurer measurer;
}
```


Misurare i rettangoli

- Costruiamo un oggetto di tipo RectangleMeasurer e passiamolo al costruttore di DataSet
  - Measurer m = new RectangleMeasurer();
  - DataSet data = new DataSet(m);
- Aggiungiamo rettangoli all'insieme dei dati
  - data.add(new Rectangle(5,10,20,30));
  - data.add(new Rectangle(10,20,30,40));
- E se aggiungiamo dati di tipo diverso?
  - Viene sollevata un'eccezione nel metodo measure al punto in cui si tenta un cast a Rectangle

Misurare i rettangoli

- RectangleMeasurer è una classe ausiliaria
  - Utilizzata solo per creare oggetti di una classe che implementa l'interfaccia Measurer
  - Possiamo dichiararla all'interno del metodo che ne ha bisogno (classe interna)

Classi interne

- Classi definite all'interno di altre classi
- I metodi della classe interna
  - hanno accesso alle variabili e ai metodi a cui possono accedere i metodi della classe in cui sono definite (accesso all'ambiente in cui è difinita)
  - possono accedere a variabili locali solo se sono state dichiarate final
    - Una variabile di tipo riferimento ad un oggetto è final quando si riferisce sempre allo stesso oggetto
    - Lo stato dell'oggetto può cambiare, ma la variabile non può riferirsi ad un altro oggetto

Esempio

```
import java.awt.Rectangle;

public class DataSetTest {
  public static void main(String[] args){
      //classe interna
    class RectangleMeasurer
                    implements Measure{
      public double measure(Object o){
        Rectangle aRectangle =
                        (Rectangle) o;
        double area = aRectangle.getWidth()
                    * aRectangle.getHeight();
        return area;
        }
    }
```

```
    Measurer m = new RectangleMeasurer();
    DataSet data = new DataSet(m);
    data.add(new Rectangle(5, 10, 20, 30));
    data.add(new Rectangle(10, 20, 30, 40));
    data.add(new Rectangle(20, 30, 5, 10));

    System.out.println("La media delle aree è = "
            + data.getAverage());
    Rectangle max =
    (Rectangle)data.getMaximum();
     System.out.println("L'area maggiore è = " +
    m.measure(max));
  }
}
```


Eventi di temporizzazione

- La classe Timer in javax.swing genera una sequenza di eventi ad intervalli di tempo prefissati
  - Utile per la programmazione di una animazione
- Un evento di temporizzazione deve essere notificato ad un ricevitore di eventi
- Per creare un ricevitore bisogna definire una classe che implementa l'interfaccia ActionListener

Esempio

```
class MioRicevitore implements ActionListener
  {
    public void actionPerformed(ActionEvent event)
    {
  //azione da eseguire ad ogni evento di
  //       temporizzazione
    }
}
ActionListener listener = new MioRicevitore();
Timer t = new Timer(interval, listener);
t.start();
```


Eventi di temporizzazione

- Un temporizzatore invoca il metodo actionPerformed dell'oggetto listener ad intervalli regolari
- Il parametro interval indica il lasso di tempo tra due eventi in millisecondi
- Vediamo un programma che conta all'indietro fino a zero con un secondo di ritardo tra un valore e l'altro

Programma CountDown

```
import java.awt.event.ActionEvent;     import java.awt.event.ActionListener;
import javax.swing.JOptionPane;        import javax.swing.Timer;

public class TimerTest{   //   Questo programma collauda la classe Timer
  public static void main(String[] args){
    class CountDown implements ActionListener {
       public CountDown(int initialCount){ count = initialCount;}
       public void actionPerformed(ActionEvent event){
            if (count >= 0) System.out.println(count);
            count--;
     }
     private int count;
    }
    CountDown listener = new CountDown(10);
    Timer t = new Timer(1000, listener);      t.start();

    JOptionPane.showMessageDialog(null, "Quit?");     System.exit(0);
  }
}
```


Eventi di temporizzazione

- Il ricevitore di eventi può aver bisogno di modificare oggetti nel metodo actionPerformed
- Occorre memorizzare questi oggetti nelle variabili di istanza della classe che implementa ActionListener
- In alternativa, dove possibile si può trarre vantaggio dalle classi interne

Esempio

```
import java.awt.event.ActionEvent;
…….   // import come esempio precedente

/** Uso di un temporizzatore per aggiungere interessi ad un conto bancario una volta al
    secondo */

public class TimerTest {
  public static void main(String[] args){

    final BankAccount account =  new BankAccount(1000);

    class InterestAdder implements ActionListener{
       public void actionPerformed(ActionEvent event){
         double interest = account.getBalance() * RATE / 100;
         account.deposit(interest);
         System.out.println("Balance = " + account.getBalance());
       }
    }

    InterestAdder listener = new InterestAdder();
    ……..  // uso Timer e finestra JOptionPane come esempio precedente
  }
  private static final double RATE = 5;
}
```