Interfacce e polimorfismo

La classe DataSet

```
/**  Serve a calcolare la media di
     un insieme di valori numerici,
     il minimo e il massimo
*/
public class DataSet {
   /**
     Default: insieme vuoto
   */
      public DataSet() {
            sum = 0;
            count = 0;
            minimum = 0;
            maximum=0;
      }
```

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

La classe DataSet

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

```
/**
    Restituisce il piu` piccolo dei
    valori
    @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.util.Scanner;

public class DataSetTest{
   public static void main(String[ ] args) {
          Scanner input = new  Scanner(System.in);
          DataSet ds = new DataSet();
          boolean done = false;
          while (!done){
                    String x = input.nextLine();
                    if (x.equalsIgnoreCase(“done”) )
                              done = true;
                    else
                              ds.add(Double.parseDouble(x));
          }
          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, il conto corrente di
     saldo massimo e quello di saldo
     minimo
*/

public class DataSet {
   /**
      Default: 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  e aggiorna i dati
   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;
}
```

La classe DataSetTest

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

      bankData.add(new BankAccount(0));
      bankData.add(new BankAccount(10000));
      bankData.add(new BankAccount(2000));

      System.out.println("Saldo medio = "+ bankData.getAverage());
      System.out.println("Saldo piu` alto = "+ bankData.getMaximun());
   }
}
```

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,
    la moneta di valore massimo e
    quella di valore minimo
*/

public class DataSet {
   /**
      Default: 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 massimo
 */
   public Coin getMaximum()
   {
      return maximum;
   }
```

La classe DataSet per le monete

```
 //   Restituisce una moneta con il valore minimo
   public Coin getMinimum() {  return minimum; }
//  Aggiunge una moneta e aggiorna i dati
   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;
}
```

La classe DataSetTest

```
/**
   Questo programma collauda la classe DataSet per le monete
*/
public class DataSetTest {
   public static void main(String[] args) {
      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(“Media dei valori delle monete = "+
            coinData.getAverage());
      System.out.println(“Moneta con valore piu` alto = "+ coinData.getMaximun());
   }
}
```

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

- L'implementazione di getMeasure deve variare a seconda di ciò che rappresenta realmente l'oggetto (double, conto, moneta,...)
  - non possiamo averne un'unica implementazione
- Ogni classe deve avere il suo metodo getMeasure
- Per avere un'unica implementazione di DataSet, tutte queste classi devono essere di uno stesso tipo
  - un tipo che ha getMeasure nella sua interfaccia pubblica
- Definiamo un tipo Measurable con metodo getMeasure

Interfaccia (Java interface)

- Measurable esprime una caratteristica comune a diversi concetti, non è un concetto in se
  - Measurable è una caratteristica di ogni cosa che può essere misurata (BankAccount, Rectangle, Coin, Double,etc.)
- Non serve una classe Measurable
  - un'implementazione unica di getMeasure significativa non è possibile (la definizione non è unica)
  - oltre a Measurable un oggetto può avere altre caratteristiche che può essere utile individuare con un tipo e che non sono in relazione con Measurable (ad es. Comparable, Cloneable)
- Si usa il costrutto Java "interface" (interfaccia)

Concetto di interfaccia Java

- Un'interfaccia dichiara una collezione di metodi elencandone le firme (con tipo del valore restituito) ma non fornisce alcuna implementazione
- Es.: l'interfaccia Measurable è

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

- Un'interfaccia può essere implementata da una o più classi
- Le classi che implementano un'interfaccia sono obbligate a fornire il codice per i metodi indicati nell'interfaccia

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 (non serve specificatore d'accesso)
- 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 con Measurable

```
/**
 Serve a computare la media delle
    misurazioni su un insieme di
    oggetti, l'oggetto con misura
    massima e quello con misura
    minima
*/

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

```
// Restituisce la media delle
     misurazioni

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

   /**Restituisce un oggetto
     Measurable di misura massima
 */
   public Measurable getMaximum()
   {
      return maximum;
   }
```

La classe DataSet con Measurable

```
 //   Restituisce un oggetto Measurable di misura minima
   public Measurable getMinimum() {  return minimum; }
//  Aggiunge un oggetto Measurable e aggiorna i dati
   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 dalle firme 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
}
```

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));

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

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

La classe DataSetTest

```
        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());
    }
}
```

Uso di parametri di tipo in DataSet

- getMinimum e getMaximum restituiscono oggetti di tipo Measurable
- In DataSetTest variabile max è di tipo Measurable
  - se serve BankAccount (primo caso) o Coin (secondo caso) occorre fare casting
- DataSet può elaborare statistiche che mischiano oggetti di tipo differente
  - ad es. Coin e BankAccount
- Possiamo risolvere i problemi usando i parametri di tipo (generics)

La classe DataSet parametrica

```
/**
 Serve a computare la media delle
    misurazioni su un insieme di
    oggetti, l'oggetto con misura
    massima e quello con misura
    minima
*/

public class DataSet
          <T extends Measurable> {
   /**
      Default: 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 di tipo T di
     misura massima
  */
   public T getMaximum()
   {
      return maximum;
   }
```

La classe DataSet parametrica

```
 //   Restituisce un oggetto di tipo T di misura minima
   public T getMinimum() {  return minimum; }
//  Aggiunge un oggetto di tipo T
   public void add(T 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 T minimum;
   private T maximum;
   private int count;
}
```

La classe DataSetParTest

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

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

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

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

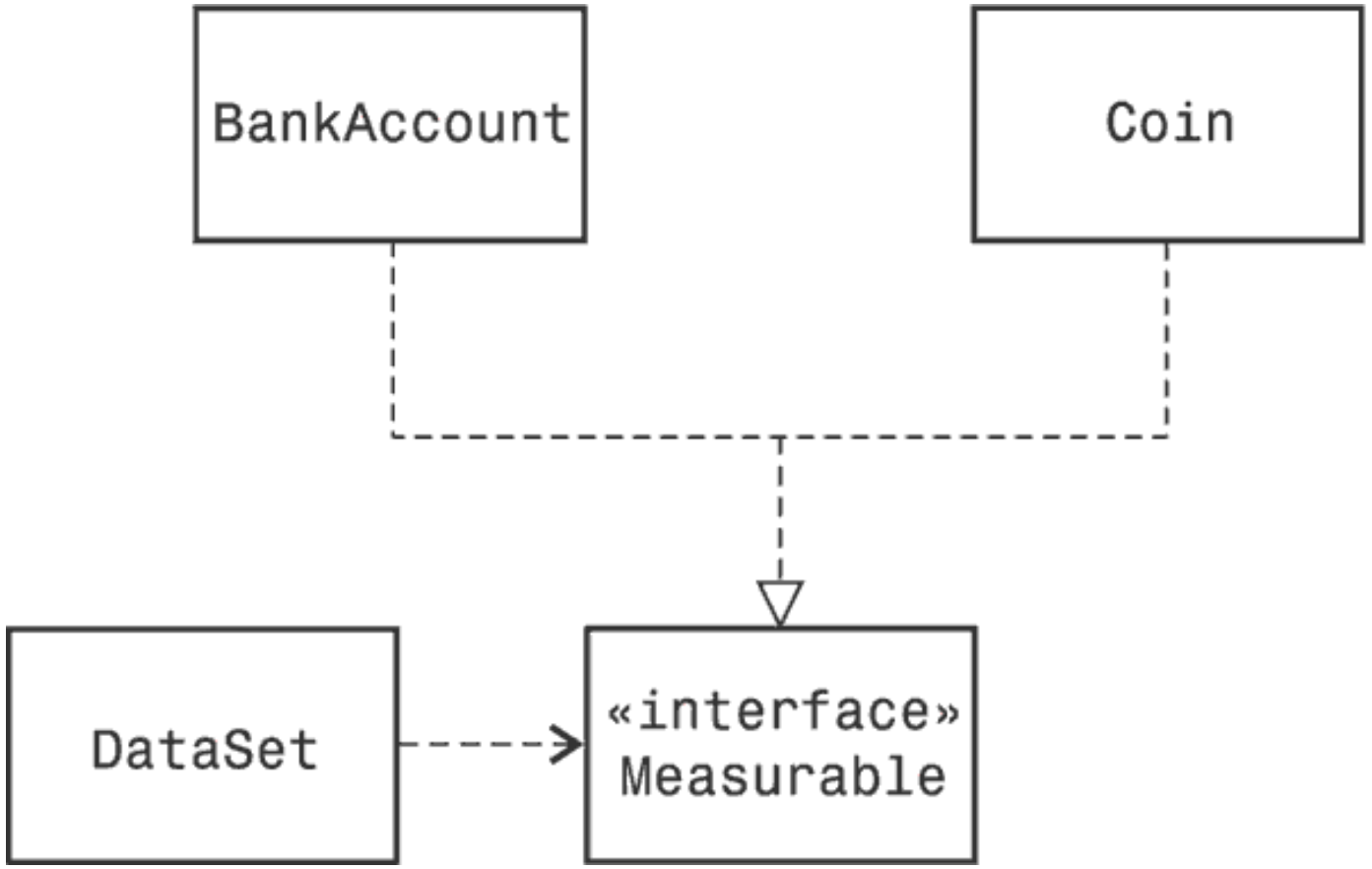
La classe DataSetParTest

```
        DataSet<Coin> coinData = new DataSet<Coin>();

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

Esercizio

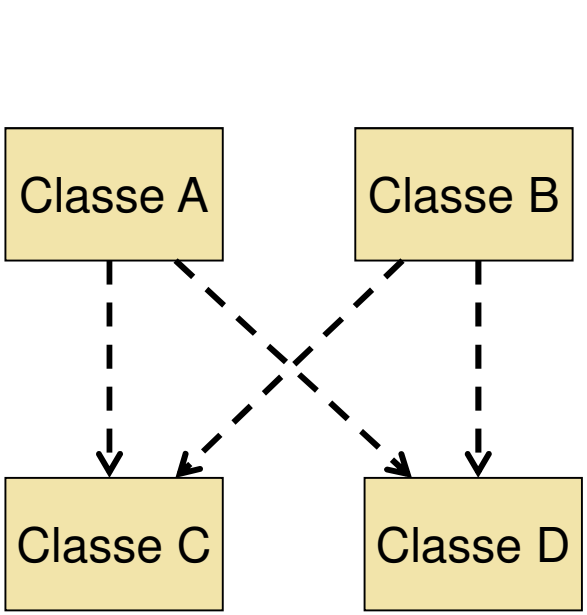
- Come agisce la type erasure su DataSet parametrico e DataSetParTest?
- Scrivere il codice sorgente delle due classi dopo che avviene la type erasure

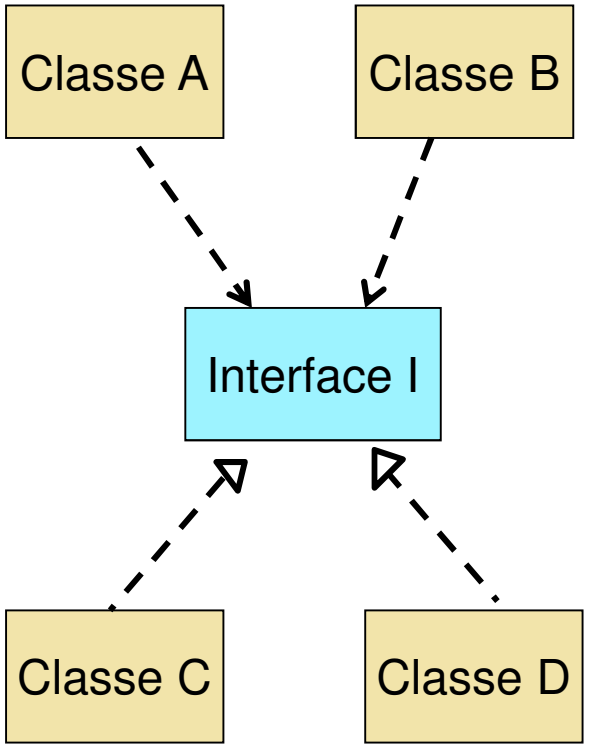
Schema UML della classe `DataSet` che dipende da `Measurable` e delle classi che implementano l'interfaccia `Measurable`



Le interfacce possono ridurre l'accoppiamento tra classi

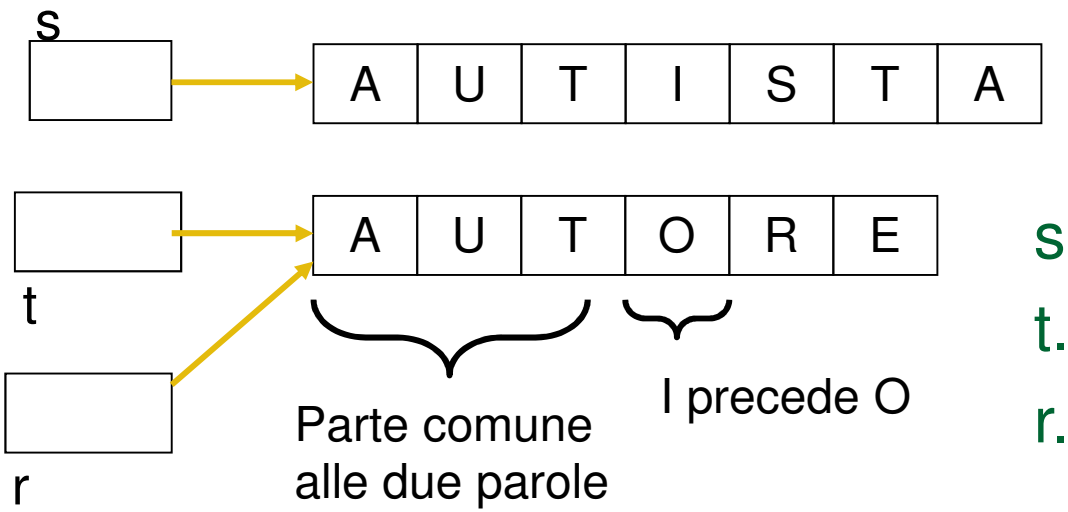
Riduzione livello di accoppiamento





L'interfaccia Comparable

- Il metodo compareTo della classe String



s.compareTo(t) è < 0

t.compareTo(s) è > 0

r.compareTo(t) è 0

L'interfaccia Comparable

- La classe String implementa l'interfaccia Comparable, appartenente alla libreria standard di Java

```
public interface Comparable <T>{
    int compareTo(T other);
}
```

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 è possibile effettuare una conversione dal tipo di una classe al tipo di un'interfaccia che NON è 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 casting
  - 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 restituito da getMaximum
// è già di 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 casting 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
- 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 (multiforme)
- Realizzato in Java attraverso:
  - Uso di interfacce
  - Ereditarietà -- Overriding (prossime lezioni)
- Altro caso di polimorfismo in senso lato
  - Overloading --- metodi sono distinti dai 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 standard: 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 parametri differenti?
  - un rettangolo con perimetro ed area
  - c/c bancario con saldo e tasso di interesse

Interfacce di smistamento

- Definiscono tipi di oggetti che possono estrarre informazioni su oggetti di altre classi
  - Con

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

    misurazione demandata all'oggetto stesso

  - Con

```
public interface Measurer<T>{
    double measure(T anObject);
// restituisce la misura dell'oggetto anObject
}
```

    misurazione implementata in una classe dedicata
    (Interfaccia di smistamento)

Misurazione dell'area dei rettangoli

```
class RectangleAreaMeasurer implements Measurer<Rectangle>
{
  public double measure(Rectangle aRectangle)
        {
          double area =
            aRectangle.getWidth() * aRectangle.getHeight();
          return area;
        }
}
```

Note su RectangleAreaMeasurer

- La firma del metodo measure della nostra classe deve essere lo stesso del metodo omonimo nell'interfaccia Measurer
  - measure ha un parametro del tipo del parametro (come indicato nell'interfaccia Measurer)

- Dopo Type erasure:

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

        public double measure(Object obj){
                return measure((Rectangle)obj); }
}
```

Soluzione ai problemi

- La nuova classe DataSet viene costruita con un oggetto di una classe che realizza l'interfaccia Measurer
- Tale oggetto viene memorizzato nella variabile di 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 DataSetMeasurer<T> {
   /**
      Costruisce un insieme vuoto
   */
   public DataSetMeasurer
                        (Measurer<T> 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 T getMaximum()
  {
     return maximum;
  }
```

La classe DataSet con l'oggetto Measurer

```
 //   Restituisce un oggetto con il valore più piccolo
   public T getMinimum() {  return minimum; }
//  Aggiunge un oggetto
   public void add(T 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 T minimum;
   private T maximum;                     private int count;
   private Measurer<T> measurer;
}
```

Misurare i rettangoli

- Costruiamo un oggetto di tipo RectangleAreaMeasurer e passiamolo al costruttore di DataSetMeasurer
  - Measurer<Rectangle> m = new RectangleAreaMeasurer();
  - DataSetMeasurer<Rectangle> data =
    new DataSetMeasurer<Rectangle>(m);
- Aggiungiamo rettangoli all'insieme dei dati
  - data.add(new Rectangle(5,10,20,30));
  - data.add(new Rectangle(10,20,30,40));
- Estraiamo misure:
  - Rectangle max = data.getMaximum();
  - Rectangle min = data.getMinimum();

Misurare i rettangoli

- RectangleMeasurer è una classe ausiliaria
  - Richiesta per l'utilizzo di codice che usa un Measurer
  - Implementa un concetto specifico di misura su oggetti Rectangle
  - Il concetto espresso può non essere rilevante al di fuori del contesto in cui è utilizzato
  - Possiamo dichiarare la classe all'interno del metodo che ne ha bisogno (classe interna)

Classi interne

- Classi definite all'interno di altre classi
  - fuori dai metodi: visibile in tutti i metodi
  - all'interno di un metodo: visibile solo nel metodo

- 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 definita)
    - se definite in un metodo statico accedono solo alle variabili statiche non alle variabili di istanza
  - possono accedere a variabili locali solo se sono state dichiarate final

Esempio

```
import java.awt.Rectangle;

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

```
    Measurer m = new RectangleMeasurer();
     DataSetMeasurer<Rectangle> data = new
      DataSetMeasurer<Rectangle>(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 =
                      data.getMaximum();

     System.out.println("L'area maggiore è = " +
                                        m.measure(max));
   }
}
```

Approfondimento su classi interne

- Il bytecode di una classe interna viene messo in un file separato
  - una classe interna è trattata come una classe standard da questo punto di vista
  - nel file system, il file .class delle classi interne ha un nome che combina il nome della classe ospitante e quello della classe interna (separati da $)
- Come si accede all'ambiente in cui è definita la classe interna?
  - Si usano variabili di istanza nascoste (aggiuntive alle variabili di istanza della classe interna)

Approfondimento su classi interne

- quando un oggetto della classe interna è istanziato, una variabile nascosta è assegnata con il riferimento all'oggetto in cui la classe interna è istanziata
  - in questo modo si ha un riferimento ai metodi, variabili di istanza e variabili statiche (visibili nell'ambiente in cui è definita la classe interna)
- se la classe interna è definita in un metodo, in aggiunta:
  - ogni variabile locale dell'ambiente esterno usata nella classe interna viene copiata in una variabile di istanza nascosta
  - stessa cosa per i parametri espliciti
  - dichiarare queste variabili final serve ad assicurare consistenza (queste variabili hanno diverse allocazioni in memoria)

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 in java.awt.event

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

- Implementare un ricevitore come classe non interna
  - Il ricevitore di eventi può aver bisogno di modificare lo stato di oggetti nel metodo actionPerformed
  - Occorre memorizzare questi oggetti nelle variabili di istanza della classe che implementa ActionListener
- In genere preferibile definire ricevitore come classi interne
  - Può accedere alle variabili dell'ambiente in cui è implementata la classe
  - Un ricevitore ha solitamente un uso locale (funzionalità specifiche dell'applicazione in cui viene usato)

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;
}
```