Scrivere codice riutilizzabile

Codice riutilizzabile

- Riutilizzo codice: unica soluzione algoritmica in diversi contesti
- Concetto chiave è il polimorfismo
  - Ereditarietà
  - Interfacce
- Maggiore facilità di uso con tipi differenti utilizzando i parametri di tipo

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

Soluzione già individuata

- Definiamo `DataSet` su oggetti di tipo dell'interfaccia `Measurable`

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

- Rendiamo gli oggetti su cui calcolare le statistiche `Measurable` implementando l'interfaccia
  - .... `BankAccount` **`implements`** `Measurable`....
  - .... `Coin` **`implements`** `Measurable`....

- Usiamo il polimorfismo di `getMeasure` per scrivere codice riutilizzabile

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;
}
```

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)

Tipi generici (generics)

- Programmazione generica:

  Creazione di costrutti che possono essere utilizzati con tipi di dati diversi
  - Es. ArrayList<String>, ArrayList<Rectangle>,etc.

- Semplifica l'uso di soluzioni generali
  - Possiamo usare tipi differenti senza ricorrere di continuo al casting

- Si può realizzare usando tipi generici cioè parametri
  - a cui si può assegnare un tipo non primitivo
  - che possono essere usati come tipi nelle dichiarazioni

Realizzazione di classi con tipi generici

- I tipi generici
  - sono dichiarati tra parentesi angolari “<“ e “>” dopo il nome della classe
  - di solito sono indicati con una lettera maiuscola
  - sono utilizzati come tipi per dichiarare le variabili, i parametri dei metodi e il valore di restituzione di un metodo nel codice della classe
- Es.: una classe generica Pair che contiene coppie di oggetti

Realizzazione di classi con parametri

```
public class Pair<S,T>{
    public Pair(S primoEl, T secondoEl) {
      primo = primoEl;
      secondo = secondoEl;
    }

  public S getFirst() { return primo; }
  public T getSecond() { return secondo; }
  private S primo;
  private T secondo;
}
```

- Pair<S,T> definisce un tipo generico
- Una classe può usare più variabili di tipo nella sua definizione, ma ogni tipo deve essere unico
  - Pair<T,T> genera errore sulla seconda T

Classe tester per Pair

```
public class PairTester {
    public static void main(String[] args) {
      Pair<Double,Integer> p =
                    new Pair<Double,Integer>(3.0,3);

      double x = p.getFirst();
      int y = p.getSecond();
    }
}
```

- L'effetto ottenuto è come se le variabili di tipo venissero assegnate con i tipi indicati al momento dell'istanziazione
  - in questo caso, S con Double e T con Integer

Metodi generici

- Possono appartenere anche a classi non generiche
- Considera l'esempio:

```
public class ArrayUtil{
  public static String print(String[] a){
        String s="";
        for(String e : a)
              s+=e+" ";
        s+='\n';
        return s;
   }
   ……
}
```

- Questo algoritmo può essere riutilizzato per convertire in String un array di oggetti di qualsiasi tipo

Metodi generici

- Stampa array di oggetti di tipo arbitrario:

```
public class ArrayUtil{
  public static <E> String print(E[] a){
       String s="";
       for(E e : a)
             s+=e+" ";
       s+='\n';
       return s;
   }
   ……
}
```

- Utilizzo metodo:

```
Rectangle[ ] rectangles= ……;
ArrayUtil.print(rectangles);
```

Osservazioni

- Per utilizzare metodo generico, non occorre specificare il tipo effettivo da assegnare alle variabili di tipo
- Il tipo del parametro è dedotto dal compilatore dall'uso che ne facciamo
  - Nell'esempio, il compilatore deduce che Rectangle è il tipo effettivo da usare per E

Limiti al tipo assegnabile ai parametri

- Può essere necessario limitare variabili di tipo
- Es: un metodo generico min va bene per oggetti che possono essere confrontati

```
public static <E> E min(E[] a) .........
```

non pone alcun vincolo sul tipo che possiamo assegnare ad E

Uso di extends per tipi generici

```
public static <E extends Comparable> E min(E[] a) {
   E smallest = a[0];
   for (int i=0; i<a.length;i++)
       if (a[i].compareTo(smallest)<0)
              smallest = a[i];
   return smallest;
}
```

- In questo caso E può essere assegnata con un qualsiasi tipo che è compatibile con Comparable (sotto-tipo di Comparable)
- Per esprimere più di un vincolo si usa “&”

```
public static <E extends Comparable & Cloneable> E
   min(E[] a) .........
```

Errori con uso variabili di tipo

```
public class MyClass<E> {
    public static void myMethod(E item) {
        if (item instanceof E) {
            //Compiler error ...
        }
        E item2 = new E(); //Compiler error
        E[ ] iArray = new E[10]; //Compiler error

    }
}
```

Invece è possibile…

```
public class MyClass<E> {
    public static void myMethod(E item) {
        E[ ] iArray = (E[ ]) new Object[10];
        ArrayList<E> x =
            (ArrayList<E>) new Arraylist();
        //Unchecked cast warning
    }
}
```

Utile per istanziare una collezione di tipo qualsiasi e si vuole evitare di fare il casting degli elementi

(casting dell'intera collezione genera ClassCastException)

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());
    }
}
```

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

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

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

Soluzione ai problemi

- La nuova classe DataSet viene costruita con un oggetto di una classe che realizza l'interfaccia Measurer
- L'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)

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));
   }
}
```