Programmazione II

A.A. 2022-23
Prof. Maria Tortorella

Callbacks ed eventi temporali

- Callbacks
- Inner Classes
- Eventi temporali e listener

Callbacks

Limiti dell'interfaccia `Measurable`:

- Può essere aggiunta solo a classi su cui si ha il controllo
- Realizza un solo modo per misurare un oggetto
  - Esempio: non può "misurare" un insieme di conti correnti sia per giacenza che per massimo scoperto, oppure non si può misurare uno studente sia per la media degli esami superati che per l'età
- Callback: consente ad una classe di delegare un apposito metodo per ottenere informazioni diverse dagli oggetti di una stessa classe o di classi diverse, in modo neutro

Interfacce e Callbacks

- Misurazione demandata all'oggetto
  - Solo una misurazione è possibile

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

- Misurazione implementata in una classe dedicata
  - È possibile implementare più tipi di misurazioni

```
public interface Measurer
{
   double measure(Object anObject);
}
```

Interfacce e Callbacks

- Il metodo `add` potrebbe invocare un measurer per effettuare la misurazione dell'oggetto che deve essere aggiunto
- Non l'oggetto da aggiungere

```
public void add(Object x)
{
   sum = sum + measurer.measure(x);
   if (count == 0 ||
         measurer.measure(maximum) < measurer.measure(x))
      maximum = x;
   count++;
}
```

Interfacce e Callbacks

- Se l'oggetto di tipo DataSet vuole misurare la media ed il massimo del media dei voto di oggetti Student

```
public class StudentMeasurerByVote implements Measurer {
   public double measure(Object anObject)    {
      Student aStudent = (Sudent) anObject;
       return aStudent.getVote();
   }
}
```

- Se lo si vuole misurare in base all'anno di iscrizione

```
public class StudentMeasurerByYear implements Measurer {
   public double measure(Object anObject)    {
      Student aStudent = (Sudent) anObject;
       return aStudent.getYear();
   }
}
```

```
Measurer m = new StudentMeasurerByVote();
DataSet data = new DataSet(m);
data.add(new Student(Mario,Rossi,25,2,false));
data.add(new Student(Paolo,Verdi,27.5,3,true));
. . .
```

Interfacce e Callbacks

- Le classi che implementano l'interfaccia measurer possono ovviamente realizzare diverse tipologie di misurazione
  - Anche per classi che non è possibile modificare perché non si possiede il codice Java

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

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

Interfacce e Callbacks

```
public class RectangleMeasurerByPerimeter implements Measurer{
   public double measure(Object anObject){
      Rectangle aRectangle = (Rectangle) anObject;
      double perimeter = 2*aRectangle.getWidth() +
                                 2 * aRectangle.getHeight();
      return perimeter;
   }
}
```

```
Measurer m = new RectangleMeasurerByPerimeter();
DataSet data = new DataSet(m);
data.add(new Rectangle(5, 10, 20, 30));
data.add(new Rectangle(10, 20, 30, 40));
System.out.println(data.getMaximun());
m = new RectangleMeasurerByArea;
data = new DataSet(m);
data.add(new Rectangle(5, 10, 20, 30));
data.add(new Rectangle(10, 20, 30, 40));
System.out.println(data.getMaximun());
```

Interfacce e Callbacks

- Notare il casting da `Object` a `Rectangle`

```
Rectangle aRectangle = (Rectangle) anObject;
```

- La classe DataSet deve conoscere quale measurer utilizzare
- Passaggio in fase di istanziazione

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

File DataSet.java

```
01: /**
02:    Computes the average of a set of data values.
03: */
04: public class DataSet
05: {
06:    /**
07:       Constructs an empty data set with a given measurer
08:       @param aMeasurer the measurer that is used to
              // measure data values
09:    */
10:    public DataSet(Measurer aMeasurer)
11:    {
12:       sum = 0;
13:       count = 0;
14:       maximum = null;
15:       measurer = aMeasurer;
16:    }
17:
```

File DataSet.java

```
18:    /**
19:       Adds a data value to the data set.
20:       @param x a data value
21:    */
22:    public void add(Object x)
23:    {
24:       sum = sum + measurer.measure(x);
25:       if (count == 0
26:       || measurer.measure(maximum)<measurer.measure(x))
27:          maximum = x;
28:       count++;
29:    }
30:
31:    /**
32:       Gets the average of the added data.
33:       @return the average or 0 if no data has been added
34:    */
```

File DataSet.java

```
35:    public double getAverage()
36:    {
37:       if (count == 0) return 0;
38:       else return sum / count;
39:    }
40:
41:    /**
42:       Gets the largest of the added data.
43:       @return the maximum or 0 if no data has been adde
44:    */
45:    public Object getMaximum()
46:    {
47:       return maximum;
48:    }
49:
```

File DataSet.java

```
50:     private double sum;
51:     private Object maximum;
52:     private int count;
53:     private Measurer measurer;
54: }
```

File DataSetTester2.java

```
01: import java.awt.Rectangle;
02:
03: /**
04:    This program demonstrates the use of a Measurer.
05: */
06: public class DataSetTester2
07: {
08:    public static void main(String[] args)
09:    {
10:       Measurer m = new RectangleMeasurerByArea();
11:
12:       DataSet data = new DataSet(m);
13:
14:       data.add(new Rectangle(5, 10, 20, 30));
15:       data.add(new Rectangle(10, 20, 30, 40));
16:       data.add(new Rectangle(20, 30, 5, 10));
17:
```

File DataSetTester2.java

```
18:         System.out.println(
19:                 "Average area = " + data.getAverage());
20:         Rectangle max = (Rectangle) data.getMaximum();
21:         System.out.println(
22:                         "Maximum area rectangle = " + max);
23:     }
24: }
```

File Measurer.java

```
01: /**
02:    Describes any class whose objects can measure
03:    other objects.   */
04: public interface Measurer
05: {
06:    /**
07:       Computes the measure of an object.
08:       @param anObject the object to be measured
09:       @return the measure
10:    */
11:    double measure(Object anObject);
12: }
```

File RectangleMeasurer.java

```
01: import java.awt.Rectangle;
02:
03: /**
04:     Objects of this class measure rectangles by area.
05: */
06: public class RectangleMeasurerByArea
07:                                     implements Measurer
08: {
09:    public double measure(Object anObject)
10:    {
11:       Rectangle aRectangle = (Rectangle) anObject;
12:       double area = aRectangle.getWidth()
                  * aRectangle.getHeight();
13:       return area;
14:    }
15: }
16:
```

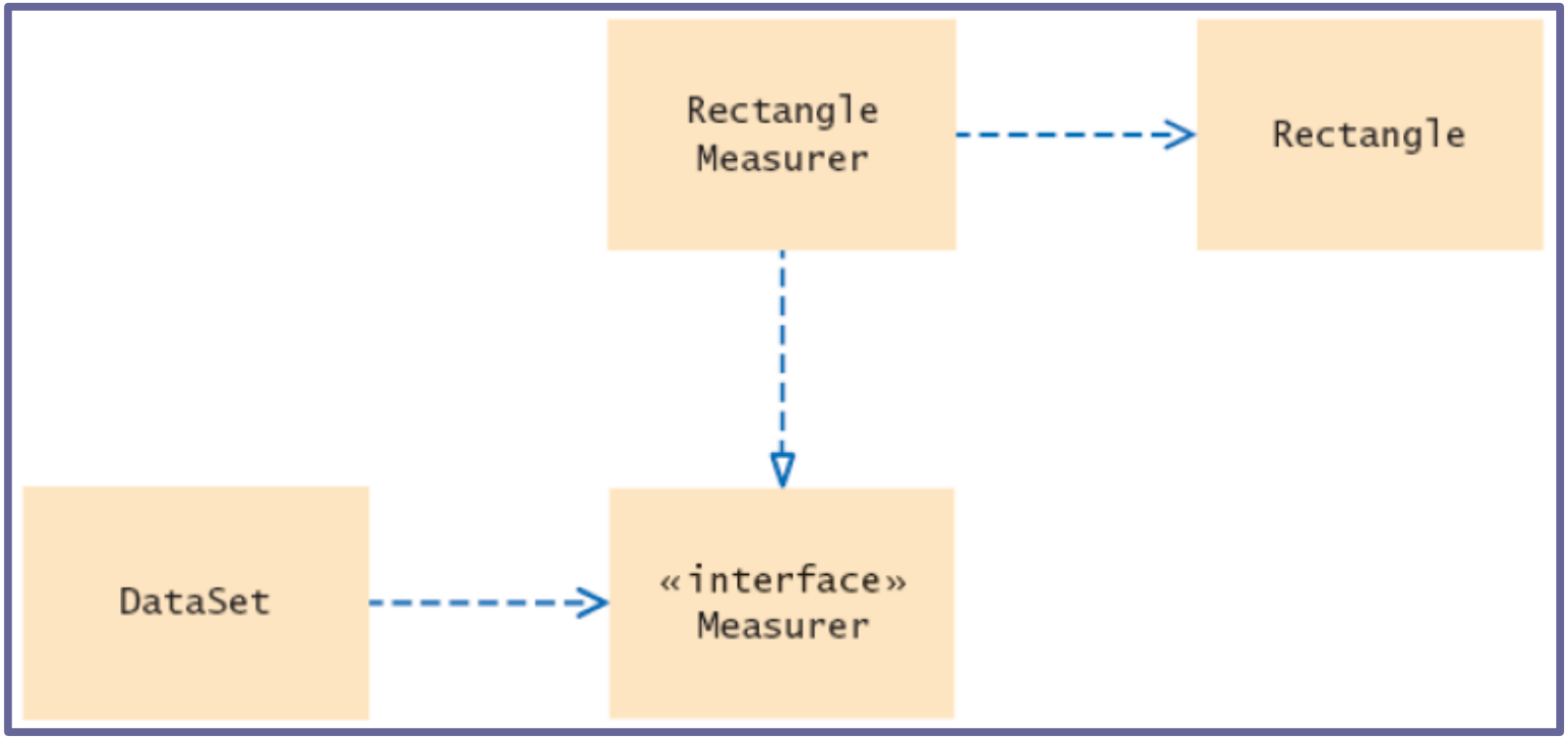
File RectangleMeasurer.java

Output:

```
Average area = 616.6666666666666
Maximum area rectangle = java.awt.Rectangle[x=10,y=20,
   // width=30,height=40]
```

UML Diagram

- La classe `Rectangle` è decoupled dall'interfaccia `Measurer`



Inner Classes

- E' possibile definire classi all'interno di un metodo

```
public class DataSetTester3
{
   public static void main(String[] args)
   {
      class RectangleMeasurer implements Measurer
      {
         . . .
      }
      Measurer m = new RectangleMeasurer();
      DataSet data = new DataSet(m); . . .
   }
}
```

Inner Classes

- Una classe definita in un metodo
- O, più frequentemente, all'interno di una classe, ma al di fuori dei metodi
  - Accessibile a tutti i metodi della classe
- Nota: il compilatore in realtà trasforma una inner class in un normale class file:

```
DataSetTester3$1$RectangleMeasurer.class
```

Nota sintattica

Declared inside a method

```
class OuterClassName
{
   method signature
   {
      . . .
      class InnerClassName
      {
         // methods
         // fields
      }
      . . .
   }
   . . .
}
```

Declared inside the class

```
class OuterClassName
   {
      // methods
      // fields
      accessSpecifier class
         InnerClassName
      {
         // methods
         // fields
      }
   . . .
}
```

File FileTester3.java

```
01: import java.awt.Rectangle;
02:
03: /**
04:    This program demonstrates the use of a Measurer.
05: */
06: public class DataSetTester3
07: {
08:    public static void main(String[] args)
09:    {
10:       class RectangleMeasurerByArea implements Measurer
11:       {
12:          public double measure(Object anObject)
13:          {
14:            Rectangle aRectangle = (Rectangle) anObject;
15:            double area = aRectangle.getWidth()
16:                                * aRectangle.getHeight();
17:             return area;
```

File FileTester3.java

```
18:          }
19:       }
20:
21:       Measurer m = new RectangleMeasurerByArea();
22:
23:       DataSet data = new DataSet(m);
24:
25:       data.add(new Rectangle(5, 10, 20, 30));
26:       data.add(new Rectangle(10, 20, 30, 40));
27:       data.add(new Rectangle(20, 30, 5, 10));
28:
29:       System.out.println("Average area = "
30:                                  + data.getAverage());
31:       Rectangle max = (Rectangle) data.getMaximum();
32:       System.out.println("Maximum area rectangle = "
33:                                                max);
34:    }
35: }
```

Gestire eventi temporali

- `javax.swing.Timer` genera una sequenza di eventi ad intervalli di tempo prefissati
- Utile per gestire aggiornamenti periodici dello stato di un oggetto
- Gli eventi sono gestiti mediante un opportuno listener

```
public interface ActionListener
{
   void actionPerformed(ActionEvent event);
}
```

Gestire eventi temporali

- Un esempio di classe che implementa l'interfaccia `ActionListener`

```
class MyListener implements ActionListener
{
      void actionPerformed(ActionEvent event)
    {
       // This action will be executed at each timer event
       Place listener action here
    }
}
```

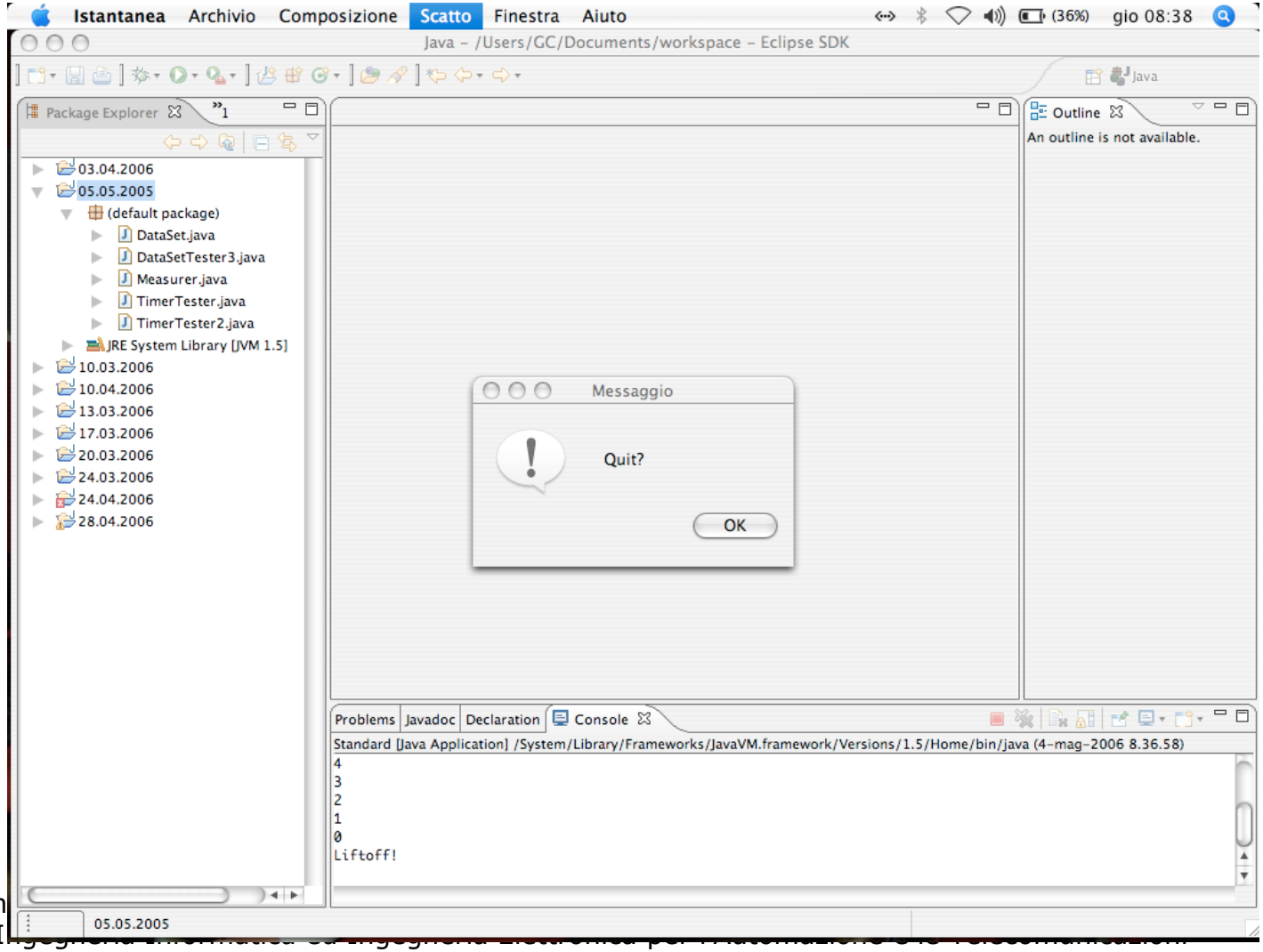
Processing Timer Events

- Add listener to timer

```
MyListener listener = new MyListener();
Timer t = new Timer(interval, listener);
t.start();
```

Example: Countdown

- Example: a timer that counts down to zero



File TimeTester.java

```
01: import java.awt.event.ActionEvent;
02: import java.awt.event.ActionListener;
03: import javax.swing.JOptionPane;
04: import javax.swing.Timer;
05:
06: /**
07:    This program tests the Timer class.
08: */
09: public class TimerTester
10: {
11:    public static void main(String[] args)
12:    {
13:       class CountDown implements ActionListener
14:       {
15:          public CountDown(int initialCount)
16:          {
17:             count = initialCount;
18:          }
```

File TimeTester.java

```
19:
20:          public void actionPerformed(ActionEvent event)
21:          {
22:             if (count >= 0)
23:                System.out.println(count);
24:             if (count == 0)
25:                System.out.println("Liftoff!");
26:             count--;
27:          }
28:
29:          private int count;
30:       }
31:
32:       CountDown listener = new CountDown(10);
33:
34:       final int DELAY = 1000; // Milliseconds between
             // timer ticks
```

File TimeTester.java

```
35:         Timer t = new Timer(DELAY, listener);
36:         t.start();
37:
38:         JOptionPane.showMessageDialog(null, "Quit?");
39:         System.exit(0);
40:     }
41: }
```

Accesso alle variabili della classe contenitore

- I metodi di una inner class possono accedere le variabili definite nei blocchi in cui sono contenute
- Utile per la realizzazione di gestori di eventi (event handlers)
- Esempio: un'animazione
  Spostare un rettangolo

Variabili della classe contenitore

```
class Mover implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
      // Move the rectangle
   }
}

ActionListener listener = new Mover();
final int DELAY = 100;
// Milliseconds between timer ticks
Timer t = new Timer(DELAY, listener);
t.start();
```

Variabili della classe contenitore

- Il metodo `actionPerformed` può accedere le variabili del contenitore:

```
public static void main(String[] args) {
   . . .
   final Rectangle box = new Rectangle(5, 10, 20, 30);

   class Mover implements ActionListener {
      public void actionPerformed(ActionEvent event){
         // Move the rectangle
         box.translate(1, 1);
      }
   }
   ActionListener listener = new Mover();
   final int DELAY = 100;
   // Milliseconds between timer ticks
   Timer t = new Timer(DELAY, listener);
   t.start();
}
```

Variabili della classe contenitore

- Le variabili locali accedute da un metodo di una inner-class devono necessariamente essere dichiarate come final
- Una inner-class può sempre accedere i campi della classe contenitore relativamente all'oggetto che ha costruito la specifica istanza dell'oggetto del tipo della inner-class
- Un oggetto di una inner class creata in un metodo statico può accedere solo i campi statici della classe contenitore

File TimeTester2.java

```
01: import java.awt.Rectangle;
02: import java.awt.event.ActionEvent;
03: import java.awt.event.ActionListener;
04: import javax.swing.JOptionPane;
05: import javax.swing.Timer;
06:
07: /**
08:    This program uses a timer to move a rectangle
09:    once per second. */
10: public class TimerTester2
11: {
12:    public static void main(String[] args)
13:    {
14:     final Rectangle box = new Rectangle(5, 10, 20, 30);
15:
16:     class Mover implements ActionListener
17:       {
```

File TimeTester2.java

```
18:          public void actionPerformed(ActionEvent event){
19:              box.translate(1, 1);
20:              System.out.println(box);
21:          }
22:       }
23:
24:       ActionListener listener = new Mover();
25:
26:       final int DELAY = 100;
27:                 // Milliseconds between timer ticks
28:       Timer t = new Timer(DELAY, listener);
29:       t.start();
30:
31:       JOptionPane.showMessageDialog(null, "Quit?");
32:       System.out.println("Last box position: " + box);
33:       System.exit(0);
34:    }
35: }
```

File `TimeTester2.java`

Output:

```
java.awt.Rectangle[x=6,y=11,width=20,height=30]
java.awt.Rectangle[x=7,y=12,width=20,height=30]
java.awt.Rectangle[x=8,y=13,width=20,height=30] . . .
java.awt.Rectangle[x=28,y=33,width=20,height=30]
java.awt.Rectangle[x=29,y=34,width=20,height=30]
Last box position: java.awt.Rectangle[x=29,y=34,width=20,height=30]
```

Esercizio

- Simulazione di una diga
  - Il programma deve consentire di variare p1 e p2 durante la simulazione, e deve calcolare il valore di L nel tempo