

Gestione degli eventi

Input di un programma

■ Da console

- ❑ gestito rigidamente da programma
- ❑ Sequenza di input definita dal programma

■ Con interfaccia grafica

- ❑ Utente ha più libertà
- ❑ Sequenza di input in qualsiasi ordine
- ❑ Gestione delle finestre in Java utilizza il concetto di evento

Eventi

- Ogni volta che l'utente esegue un'azione (su elementi dell'interfaccia utente grafica)
 - un clic del mouse
 - la pressione di un tasto sulla tastiera
 - la modifica di una finestra
 - la selezione di un elemento da un menu...

viene generato un evento

Eventi: sorgenti e ricevitori

- Sorgente di eventi (**source**): componente di un'interfaccia grafica che può generare eventi
- Ricevitore di eventi (**listener**): reagisce al verificarsi di eventi attraverso le azioni descritte nei suoi metodi
- Meccanismo di funzionamento:
 - ❑ si implementa un ricevitore per ogni evento che si vuole gestire
 - ❑ le istanze di ricevitori vengono collegate alle istanze delle sorgenti di interesse
 - ❑ quando viene generato un evento, tutti ricevitori per quel tipo di evento che sono collegati alla sorgente vengono attivati

Package java.awt.event

- Ogni tipo di evento è descritto da una classe
 - Il package `java.awt.event` contiene
 - Le classi per i diversi tipi di eventi
 - Le interfacce relative ai ricevitori di eventi (`Listener`)
 - alcune classi di adattatori, gli `Adapter`, che implementano le interfacce `Listener`
-

Tipi di eventi

- **MouseEvent** (eventi del mouse)
 - click del mouse, spostamento del mouse, etc
 - **ActionEvent** (eventi di azione)
 - azioni di specifiche componenti, ad es. pressione di un pulsante
 - può essere generato anche pressando la barra spaziatrice col mouse posizionato su un pulsante
-

Tipi di eventi

- **AdjustmentEvent** (eventi di modifica)
 - eventi emessi da oggetti Adjustable (dell'interfaccia) che hanno un valore numerico modificabile (ad es. JScrollBar)
- **FocusEvent** (guadagnare/perdere input focus)
 - generato da componenti quali JTextField
- **ItemEvent** (selezione/deselezione elemento)
 - generato da elementi di tipo ItemSelectable (ad es. JButton)

Tipi di eventi

- **KeyEvent** (eventi di tastiera)
 - pressione di un tasto su una componente grafica (ad es. JTextField)
- **WindowEvent** (eventi relativi a finestre)
 - ad es. una finestra dell'interfaccia grafica ha cambiato il suo stato
- etc...

Interfacce Listener

- Sono i **ricevitori di eventi**
 - Esiste una interfaccia **Listener** per ciascun tipo di evento
 - Definiscono i metodi che devono essere implementati da un ricevitore di eventi
-

Interfacce Listener

- `MouseListener`
 - `ActionListener`
 - `AdjustmentListener`
 - `FocusListener`
 - `ItemListener`
 - `KeyListener`
 - `WindowListener`
 - `etc...`
-

ActionListener e JButton

■ Esempio:

- ❑ usare oggetti `JButton` per definire pulsanti
- ❑ collegare un `ActionListener` a ogni pulsante

■ Interfaccia `ActionListener`:

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

- Serve una classe che implementi l'interfaccia
 - ❑ l'implementazione di `actionPerformed` contiene le istruzioni da eseguire quando il pulsante viene premuto

ActionListener e JButton

- il parametro `event` contiene dettagli relativi all'evento, quali ad esempio il tempo al quale si è manifestato
- per collegare una sorgente di eventi ad un ricevitore occorre aggiungere il ricevitore alla sorgente di eventi:

```
ActionListener listener = new ClickListener();  
button.addActionListener(listener);
```

- la variabile `button` contiene un riferimento ad un oggetto JButton
- JButton (javax.swing) è una sottoclasse di JComponent
 - Un JComponent ha una variabile di istanza che contiene una lista di `listener`

File ClickListener.java

Output:



```
Terminal
File Edit View Terminal Go Help

~$ cd BigJava/ch12/button1
~/BigJava/ch12/button1$ java ButtonTester
I was clicked.
I was clicked.
I was clicked.
█
```

The terminal window displays the execution of the `ButtonTester` Java program. The output shows three instances of "I was clicked." followed by a cursor. A small GUI window titled "Click me!" is also visible, featuring a blue button.

File ClickListener.java

```
01: import java.awt.event.ActionEvent;
02: import java.awt.event.ActionListener;
03:
04: /**
05:     An action listener that prints a message.
06: */
07: public class ClickListener implements ActionListener
08: {
09:     public void actionPerformed(ActionEvent event)
10:     {
11:         System.out.println("I was clicked.");
12:     }
13: }
```

File ButtonTester.java

```
01: import java.awt.event.ActionListener;
02: import javax.swing.JButton;
03: import javax.swing.JFrame;
04:
05: //Esempio di come installare un ActionListener
08: public class ButtonTester
09: {
10:     public static void main(String[] args)
11:     {
12:         JFrame frame = new JFrame();
13:         JButton button = new JButton("Click me!");
14:         frame.add(button);
15:
16:         ActionListener listener = new ClickListener();
17:         button.addActionListener(listener);
18:
19:         frame.setSize(FRAME_WIDTH,FRAME_HEIGHT);
20:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21:         frame.setVisible(true);
22:     }
23:
24:     private static final int FRAME_WIDTH = 100;
25:     private static final int FRAME_HEIGHT = 60;
26: }
```

Applicazioni con pulsanti

- Esempio: programma visualizzatore di un investimento
 - ogni volta che il pulsante è pressato, l'interesse viene aggiunto al saldo e il nuovo saldo viene visualizzato



Costruiamo una soluzione

- Occorre istanziare un oggetto di `JButton`:

```
JButton button = new JButton("Add Interest");
```

- Una componente dell'interfaccia utente deve visualizzare un messaggio:

```
JLabel label = new JLabel("balance=" + account.getBalance());
```

- `JLabel` (pacchetto `javax.swing`) sottoclasse di `JComponent`
 - rappresenta un'area del display per visualizzare un piccolo testo e/o immagine

JPanel (pacchetto javax.swing)

- JPanel è un contenitore
 - estende JComponent
 - serve quando vogliamo aggiungere più componenti ad un frame
 - aggiungere le singole componenti al frame le sovrapporrebbe

```
JPanel panel = new JPanel();  
panel.add(button);  
panel.add(label);  
frame.add(panel);
```

- L'ordine in cui vengono aggiunte le componenti rispecchia l'ordine di visualizzazione nel frame

Implementazione del Listener

- La classe `AddInterestListener` aggiunge l'interesse e aggiorna la label con il nuovo saldo

```
class AddInterestListener implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        double interest = account.getBalance() * INTEREST_RATE / 100;
        account.deposit(interest);
        label.setText("balance=" + account.getBalance());
    }
}
```

- Viene aggiunta come classe interna per poter utilizzare le variabili (`final`) `account` e `label`

File InvestmentViewer1.java

```
01: import java.awt.event.ActionEvent;
02: import java.awt.event.ActionListener;
03: import javax.swing.JButton;
04: import javax.swing.JFrame;
05: import javax.swing.JLabel;
06: import javax.swing.JPanel;
07:
08:
09: /**
10:     This program displays the growth of an investment.
11: */
12: public class InvestmentViewer1
13: {
14:     public static void main(String[] args)
15:     {
16:         JFrame frame = new JFrame();
17:
```

File InvestmentViewer1.java

```
18:         // The button to trigger the calculation
19:         JButton button = new JButton("Add Interest");
20:
21:         // The application adds interest to this bank account
22:         final BankAccount account
           = new BankAccount(INITIAL_BALANCE);
23:
24:         // The label for displaying the results
25:         final JLabel label = new JLabel(
26:             "balance=" + account.getBalance());
27:
28:         // The panel that holds the user interface components
29:         JPanel panel = new JPanel();
30:         panel.add(button);
31:         panel.add(label);
32:         frame.add(panel);
33:
```

File InvestmentViewer1.java

```
34:      class AddInterestListener implements ActionListener
35:      {
36:          public void actionPerformed(ActionEvent event)
37:          {
38:              double interest = account.getBalance()
39:                  * INTEREST_RATE / 100;
40:              account.deposit(interest);
41:              label.setText(
42:                  "balance=" + account.getBalance());
43:          }
44:      }
45:
46:      ActionListener listener = new AddInterestListener();
47:      button.addActionListener(listener);
48:
49:      frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
50:      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
51:      frame.setVisible(true);
52:  }
```

File InvestmentViewer1.java

```
53:
54:     private static final double INTEREST_RATE = 10;
55:     private static final double INITIAL_BALANCE = 1000;
56:
57:     private static final int FRAME_WIDTH = 400;
58:     private static final int FRAME_HEIGHT = 100;
59: }
```

Elaborare testo in input

- Si usano componenti `JTextField` per riservare spazio per l'input dell'utente (javax.swing)

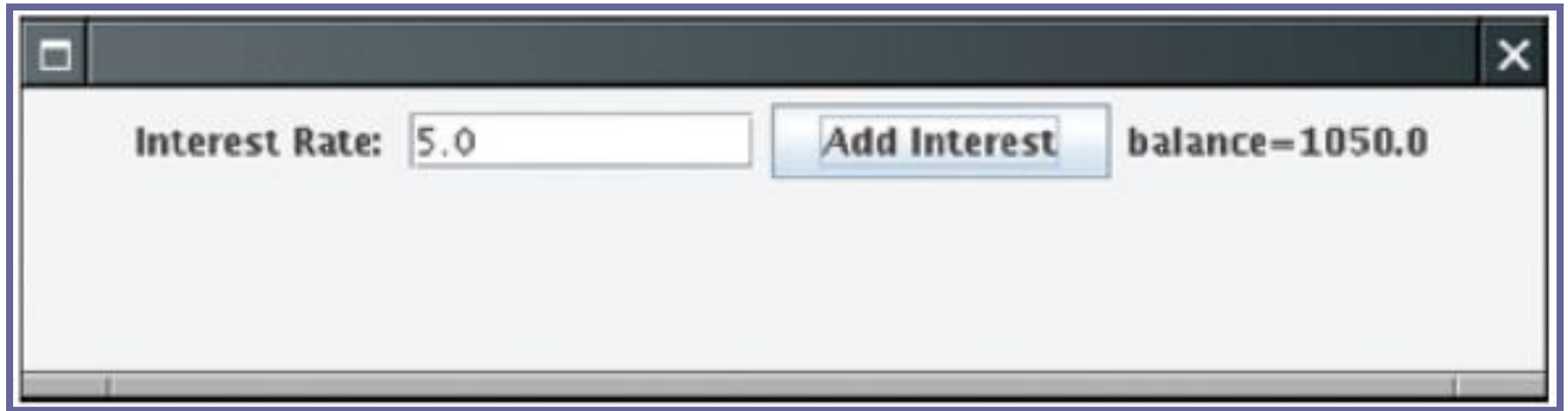
```
final int FIELD_WIDTH = 10; // caratteri in ingresso  
final JTextField rateField = new JTextField(FIELD_WIDTH);
```

- E' una sottoclasse di `JComponent`
- E' buona norma
 - usare un `JLabel` per descrivere un `JTextField`

```
JLabel rateLabel = new JLabel("Interest Rate: ");
```

- predisporre un pulsante per permettere all'utente di segnalare quando l'input è pronto per l'elaborazione

Esempio



A screenshot of a Java Swing window. The window has a title bar with a close button (X) on the right. The main content area contains the text "Interest Rate:" followed by a text input field containing the value "5.0". To the right of the input field is a button labeled "Add Interest". Further to the right is a label displaying "balance=1050.0".

Elaborazione testo in input

- Il metodo `actionPerformed` collegato alla pressione del pulsante legge il testo di input dell'utente dal campo di testo usando il metodo `getText`

```
class AddInterestListener implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        double rate = Double.parseDouble(rateField.getText());
        . . .
    }
}
```

- Si può mettere un testo di default con il metodo `setText`

File InvestmentViewer2.java

```
01: import java.awt.event.ActionEvent;
02: import java.awt.event.ActionListener;
03: import javax.swing.JButton;
04: import javax.swing.JFrame;
05: import javax.swing.JLabel;
06: import javax.swing.JPanel;
07: import javax.swing.JTextField;
08:
09: /**
10:     This program displays the growth of an investment.
11: */
12: public class InvestmentViewer2
13: {
14:     public static void main(String[] args)
15:     {
16:         JFrame frame = new JFrame();
17:
```

File InvestmentViewer2.java

```
18:         // The label and text field for entering the
           //interest rate
19:         JLabel rateLabel = new JLabel("Interest Rate: ");
20:
21:         final int FIELD_WIDTH = 10;
22:         final JTextField rateField
           = new JTextField(FIELD_WIDTH);
23:         rateField.setText("" + DEFAULT_RATE);
24:
25:         // The button to trigger the calculation
26:         JButton button = new JButton("Add Interest");
27:
28:         // The application adds interest to this bank account
29:         final BankAccount account
           = new BankAccount(INITIAL_BALANCE);
30:
31:         // The label for displaying the results
32:         final JLabel resultLabel = new JLabel(
33:             "balance=" + account.getBalance());
34:
```

File InvestmentViewer2.java

```
35:         // The panel that holds the user interface components
36:         JPanel panel = new JPanel();
37:         panel.add(rateLabel);
38:         panel.add(rateField);
39:         panel.add(button);
40:         panel.add(resultLabel);
41:         frame.add(panel);
42:
43:         class AddInterestListener implements ActionListener
44:         {
45:             public void actionPerformed(ActionEvent event)
46:             {
47:                 double rate = Double.parseDouble(
48:                     rateField.getText());
49:                 double interest = account.getBalance()
50:                     * rate / 100;
51:                 account.deposit(interest);
```

File InvestmentViewer2.java

```
52:         resultLabel.setText(  
53:             "balance=" + account.getBalance());  
54:     }  
55: }  
56:  
57: ActionListener listener = new AddInterestListener();  
58: button.addActionListener(listener);  
59:  
60: frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);  
61: frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
62: frame.setVisible(true);  
63: }  
64:  
65: private static final double DEFAULT_RATE = 10;  
66: private static final double INITIAL_BALANCE = 1000;  
67:  
68: private static final int FRAME_WIDTH = 500;  
69: private static final int FRAME_HEIGHT = 200;  
70: }
```

Eventi del mouse

- Per catturare eventi del mouse si usa un `MouseListener`
- L'interfaccia `MouseListener`:

```
public interface MouseListener
{
    void mousePressed(MouseEvent event);
    void mouseReleased(MouseEvent event);
    void mouseClicked(MouseEvent event);
    void mouseEntered(MouseEvent event);
    void mouseExited(MouseEvent event);
}
```

Eventi del mouse

- `mousePressed`: se un pulsante del mouse è pressato su una componente
- `mouseReleased`: pulsante rilasciato
- `mouseClicked`: se un pulsante del mouse è pressato e rilasciato su un componente e il mouse non si è spostato
- `mouseEntered`: il mouse entra nell'area di una componente
- `mouseExited`: ne esce

Eventi del mouse

- Si aggiunge un `MouseListener` ad una componente con il metodo `addMouseListener`:

```
public class MyMouseListener implements MouseListener
{
    // Implementa i cinque metodi
}
MouseListener listener = new MyMouseListener();
component.addMouseListener(listener);
```

- Esempio programma: miglioramento del programma `RectangleComponentViewer` (cap. 5)
 - Se un utente clicca su un frame, allora sposta il rettangolo

File RectangleComponent.java

[illegible]

File RectangleComponent.java

```
19:     public void paintComponent(Graphics g)
20:     {
21:         super.paintComponent(g);
22:         Graphics2D g2 = (Graphics2D) g;
23:
24:         g2.draw(box);
25:     }
26:
27:     /**
28:      Moves the rectangle to the given location.
29:      @param x the x-position of the new location
30:      @param y the y-position of the new location
31:     */
32:     public void moveTo(int x, int y)
33:     {
34:         box.setLocation(x, y);
35:         repaint(); //invoca di nuovo paintComponent non
36:     }              // appena possibile
```

File RectangleComponent.java

```
37:
38:     private Rectangle box;
39:
40:     private static final int BOX_X = 100;
41:     private static final int BOX_Y = 100;
42:     private static final int BOX_WIDTH = 20;
43:     private static final int BOX_HEIGHT = 30;
44: }
```

Implementazione MouseListener

```
class MousePressListener implements MouseListener
{
    public void mousePressed(MouseEvent event)
    {
        int x = event.getX();
        int y = event.getY();
        component.moveTo(x, y);
    }
    // Do-nothing methods
    public void mouseReleased(MouseEvent event) {}
    public void mouseClicked(MouseEvent event) {}
    public void mouseEntered(MouseEvent event) {}
    public void mouseExited(MouseEvent event) {}
}
```

- **Tutti i metodi devono essere implementati**
- **Metodi inutilizzati possono restare vuoti**

File RectangleComponentViewer2.java

```
01: import java.awt.event.MouseListener;
02: import java.awt.event.MouseEvent;
03: import javax.swing.JFrame;
04:
05: /**
06:     This program displays a RectangleComponent.
07: */
08: public class RectangleComponentViewer2
09: {
10:     public static void main(String[] args)
11:     {
12:         final RectangleComponent component
13:             = new RectangleComponent();
14:
15:         // Add mouse press listener
16:
17:         class MousePressListener implements MouseListener
18:         {
```

File RectangleComponentViewer2.java

```
18:         public void mousePressed(MouseEvent event)
19:         {
20:             int x = event.getX();
21:             int y = event.getY();
22:             component.moveTo(x, y);
23:         }
24:
25:         // Do-nothing methods
26:         public void mouseReleased(MouseEvent event) {}
27:         public void mouseClicked(MouseEvent event) {}
28:         public void mouseEntered(MouseEvent event) {}
29:         public void mouseExited(MouseEvent event) {}
30:     }
31:
32:     MouseListener listener = new MousePressListener();
33:     component.addMouseListener(listener);
34:
```

File RectangleComponentViewer2.java

```
35:         JFrame frame = new JFrame();
36:         frame.add(component);
37:
38:         frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
39:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
40:         frame.setVisible(true);
41:     }
42:
43:     private static final int FRAME_WIDTH = 300;
44:     private static final int FRAME_HEIGHT = 400;
45: }
```