

Programmazione ad Eventi

Eventi, Sorgenti, e *Listeners*

- **Una interfaccia utente deve gestire una moltitudine di eventi**
 - eventi da tastiera, del mouse, click su pulsanti, ...
- **Opportuno poter discriminare diversi eventi**
 - componenti specifiche devono poter essere programmate senza preoccuparsi che di eventi specifici
- **Necessario avere meccanismi per gestire eventi in modo selettivo**
 - Un meccanismo per dichiarare quali eventi si intendono gestire

Continua...

Eventi, Sorgenti, e *Listeners*

- **Listener:**
 - Oggetto che viene notificato dell'occorrere di un evento
 - I suoi metodi descrivono le azioni da eseguire in risposta ad un evento
- **Sorgente di eventi:**
 - Oggetto che origina eventi, o li emette,
 - Notifica l'evento ai propri listeners
- **Vari tipi di sorgenti**
 - Componenti visuali: `JButton`, `JTextField`, ...
 - `Timer`, ...

Continua...

Eventi, Sorgenti, e *Listeners*

- **Ogni sorgente genera un insieme ben definito di eventi**
 - determinato dal tipo della sorgente
- **Eventi di basso livello**
 - mouse-down/up/move, key-press/release, component-resize/move, ...
- **Eventi “semantici”**
 - **ActionEvent**: click su un `JButton`, doppio-click su un `JListItem`, tick di un `Timer`
 - **TextEvent**: `JTextField` modificato da uno o più mouse/key events

Continua...

Eventi, Sorgenti, e *Listeners*

- **Gestione mirata degli eventi**
 - Una applicazione può decidere quali eventi gestire tra quelli generati da ciascuna sorgente
- **Registra su ciascuna componente solo i listeners degli eventi che intende gestire**

Continua...

Eventi, Sorgenti, e *Listeners*

- **Esempio:** utilizziamo una componente `JButton` per realizzare un pulsante: associamo a ciascun pulsante un `ActionListener`

- **Ricordiamo:**

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

- **Dobbiamo fornire una classe il cui metodo `actionPerformed` contiene le istruzioni da eseguire quando il pulsante viene premuto**

Continua...

Eventi, Sorgenti, e *Listeners*

- **event** contiene informazioni sull'evento (ad esempio l'istante in cui è occorso)
- **Associamo il listener al pulsante, registrandolo sul pulsante:**

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

- **addActionListener() : un metodo di JButton**

```
void addActionListener(ActionListener l);
```

File ClickListener.java

```
01: import java.awt.event.ActionEvent;
02: import java.awt.event.ActionListener;
03:
04: /**
05:     Un action listener che stampa un messaggio
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 ClickTester.java

```
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

/**
 * Come installare un ActionListener.
 */
public class ClickApp
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();
        JButton button = new JButton("Click me!");
        frame.add(button);
    }
}
```

[Continua...](#)

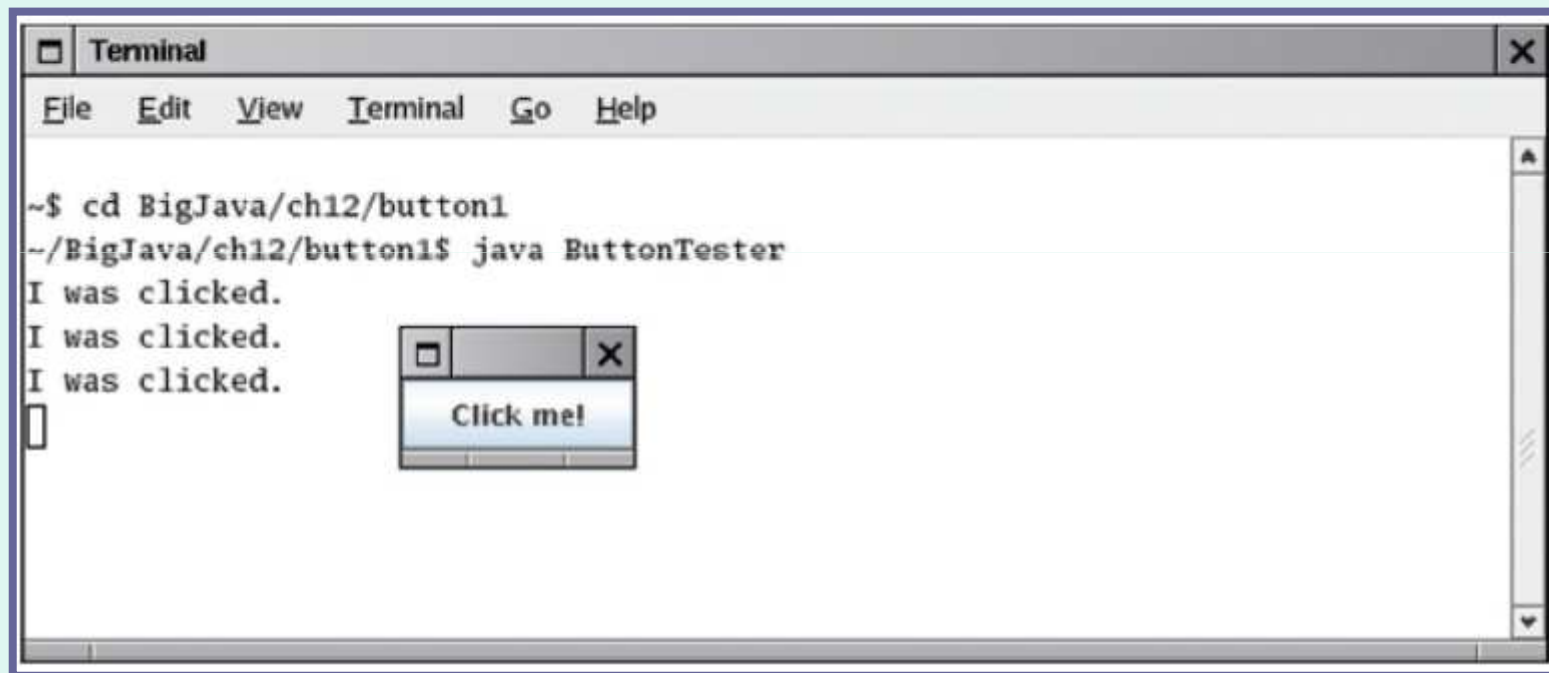
File ClickTester.java

```
    ActionListener listener =
        new ActionListener()
        {
            public void actionPerformed()
            {
                System.out.println("I was clicked");
            }
        }
    frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
}
private static final int FRAME_WIDTH = 100;
private static final int FRAME_HEIGHT = 60;
}
```

[Continua...](#)

File ClickListener.java

Output:



The image shows a terminal window titled "Terminal" with a menu bar containing "File", "Edit", "View", "Terminal", "Go", and "Help". The terminal output is as follows:

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

```

Overlaid on the terminal is a small GUI window with a title bar and a close button. The window contains a single button labeled "Click me!".

Applicazioni con pulsanti

- **Esempio: costruiamo una interfaccia grafica per calcolare gli interessi di un conto bancario**



- **Assumiamo dato il tasso di interesse**
- **Ad ogni click aggiungiamo al saldo gli interessi**

Continua...

Applicazioni con pulsanti

- **L'applicazione ha due componenti**
 - *Back-end*: il conto bancario con i suoi metodi
 - *Front-end* : l'interfaccia grafica che permette l'interazione con il backen
- **Elementi del front-end**
 - un `JButton` per il pulsante

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

- Una `JLabel` per visualizzare il saldo:

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

Continua...

Applicazioni con pulsanti

- Utilizziamo un `JPanel` come contenitore per raggruppare gli elementi della interfaccia all'interno della finestra di interazione

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

- Aggiungiamo prima il pulsante, poi l'etichetta
 - Vengono disposti da sinistra a destra, in ordine

Continua...

File BankAccountGUI.java

```
import java.awt.event.*;
import javax.swing.*;
/**
 * Interfaccia Grafica per un BanckAccount.
 */
public class BankAccountGUI
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();

        // L'etichetta per visualizzare il saldo
        final JLabel label = new JLabel(
            "balance=" + account.getBalance());
    }
}
```

[Continua...](#)

File BankAccountGUI.java

```
// Il pulsante che aggiorna il saldo
JButton button = new JButton("Add Interest");

ActionListener listener =
    new ActionListener() {
        public void actionPerformed(ActionEvent event)
        { // account è il conto su cui si agisce
            double interest = account.getBalance()
                * INTEREST_RATE / 100;
            account.deposit(interest);
            label.setText(
                "balance=" + account.getBalance());
        }
    };
button.addActionListener(listener);
```

[Continua...](#)

File BankAccountGUI.java

```
// Il JPanel che contiene le componenti dell'interfaccia
JPanel panel = new JPanel();
panel.add(button);
panel.add(label);
frame.add(panel);

frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
} // fine main()
```

Continua...

File BankAccountGUI.java

```
// Il conto bancario su cui agire
private static BankAccount account
    = new BankAccount(INITIAL_BALANCE);

private static final double INTEREST_RATE = 10;
private static final double INITIAL_BALANCE = 1000;

private static final int FRAME_WIDTH = 400;
private static final int FRAME_HEIGHT = 100;
}
```

Domanda

- Perché `label` è dichiarata `final` mentre `button` e `account` non lo sono?

Risposta

- **Perché**
 - **label** è una locale che viene riferita dalla classe interna mentre
 - **button** è anch'esso local ma non viene riferito
 - **account** è un campo e quindi si può riferire da una classe interna

Input con JTextField



Continua...

Input con JTextField

- **JTextField** la componente che permette di ottenere input

```
final int FIELD_WIDTH = 10; // numero di caratteri  
final JTextField rateField = new JTextField(FIELD_WIDTH);
```

- **Associamo ad ogni campo di testo una JLabel** per chiarire il senso dell'input

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

Continued...

Input con `JTextField`

- La lettura dell'input dal `JTextField` è la conseguenza di un evento:
- Due modi per ottenere l'evento:
 - diamo *enter* sul campo testo
 - segnaliamo la fine dell'input con un pulsante

Continua...

Input con JTextField

- **La lettura avviene notificando un action listener**

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

- **Registriamo il listener sia sul pulsante sia sul campo testo**

File BankAccountGUI2.java

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

/**
    Interfaccia Grafica per un BanckAccount
 */
public class BankAccountGUI2
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();
```

[Continua...](#)

File BankAccountGUI2.java

```
// L'etichetta e il campo di testo per l'input
JLabel rateLabel = new JLabel("Interest Rate: ");

final int FIELD_WIDTH = 10;
final JTextField rateField
    = new JTextField(FIELD_WIDTH);
rateField.setText("" + DEFAULT_RATE);

// il pulsante che esegue il calcolo
JButton button = new JButton("Add Interest");

// l'etichetta per la visualizzazione
final JLabel resultLabel = new JLabel(
    "balance=" + account.getBalance());
```

Continua...

File BankAccountGUI2.java

```
// il pannello che raccoglie le componenti GUI
JPanel panel = new JPanel();
panel.add(rateLabel);
panel.add(rateField);
panel.add(button);
panel.add(resultLabel);
frame.add(panel);

class AddInterestListener implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        double rate = Double.parseDouble(
            rateField.getText());
        double interest = account.getBalance()
            * rate / 100;
        account.deposit(interest);
```

[Continua...](#)

File BankAccountGUI2.java

```
        resultLabel.setText(
            "balance=" + account.getBalance());
    }
}
ActionListener listener = new AddInterestListener();
button.addActionListener(listener);
rateField.addActionListener(listener);
frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
}
// Il conto bancario su cui agire e le costanti
private static BankAccount account
    = new BankAccount(INITIAL_BALANCE);
private static final double DEFAULT_RATE = 10;
private static final double INITIAL_BALANCE = 1000;
private static final int FRAME_WIDTH = 500;
private static final int FRAME_HEIGHT = 200;
}
```

Are di testo

- **JTextArea**: utilizzabile per gestire più linee di testo nella stessa area
- **Specifichiamo il numero di righe e colonne al momento della creazione:**

```
final int ROWS = 10;  
final int COLUMNS = 30;  
JTextArea textArea = new JTextArea(ROWS, COLUMNS);
```

- **setText**: definisce il contenuto (JTextArea, e JTextField)
- **append**: aggiunti testo alla fine dell'area

Continua...

Are di testo

- **newline per separare le linee di testo**

```
textArea.append(account.getBalance() + "\n");
```

- **Are di testo di sola lettura**

```
textArea.setEditable(false);  
// modificabile solo dall'applicazione, non da utente
```


Aree di testo

Scrollbars

```
JTextArea textArea = new JTextArea(ROWS, COLUMNS);  
JScrollPane scrollPane = new JScrollPane(textArea);
```

Continua...

Esempio



The image shows a screenshot of a Windows application window. The window has a title bar with a close button (X) in the top right corner. Below the title bar, there is a label "Interest Rate:" followed by a text input field containing the value "10.0". To the right of the input field is a button labeled "Add Interest". Below these elements is a list box containing the following values:

- 1100.0
- 1210.0
- 1331.0
- 1464.1
- 1610.51

The list box has a vertical scrollbar on the right side, indicating that it can display more items than are currently visible.

Controllo del *layout*

- **Ovvero, controllo della disposizione dei componenti all'interno di una finestra**
- **Abbiamo visto applicazioni con una disposizione di componenti elementare**
 - I pannelli che abbiamo utilizzato dispongono le componenti in ordine, da sinistra verso destra
- **Per disporre in modo più strutturato utilizziamo pannelli con diverse politiche di disposizione (*layout*)**

Continua...

Controllo del *layout*

- Ogni contenitore è associato ad un *layout manager* che definisce la politica di disposizione delle componenti
- Tre tipologie standard di layout managers
 - border layout
 - flow layout
 - grid layout

Controllo del *layout*

- **JPanel** ha un manager di default:
 - **FlowLayout**: le componenti vengono disposte da sinistra a destra utilizzando una nuova riga quando necessario
- **È possibile modificare il layout manager con un metodo corrispondente**

```
panel.setLayout(new BorderLayout());
```

Grid Layout

- **Dispone le componenti in una griglia con un numero predeterminato di righe e colonne**
- **Tutte le componenti assumono le stesse dimensioni**
- **Espande ciascuna componente fino a riempire l'area ad essa assegnata**

Grid Layout

- **Le componenti si aggiungono per riga, da sinistra a destra:**

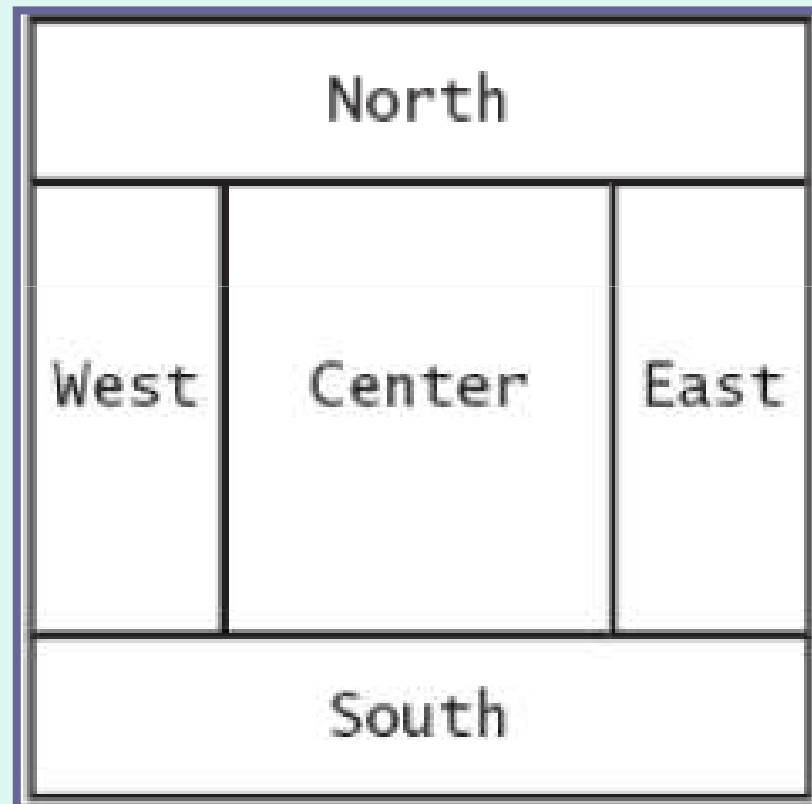
```
JPanel numberPanel = new JPanel();  
numberPanel.setLayout(new GridLayout(4, 3));  
numberPanel.add(button7);  
numberPanel.add(button8);  
numberPanel.add(button9);  
numberPanel.add(button4);  
• • •
```

Grid Layout



Border Layout

- **Dispone le componenti in aree:**



Continua...

Border Layout

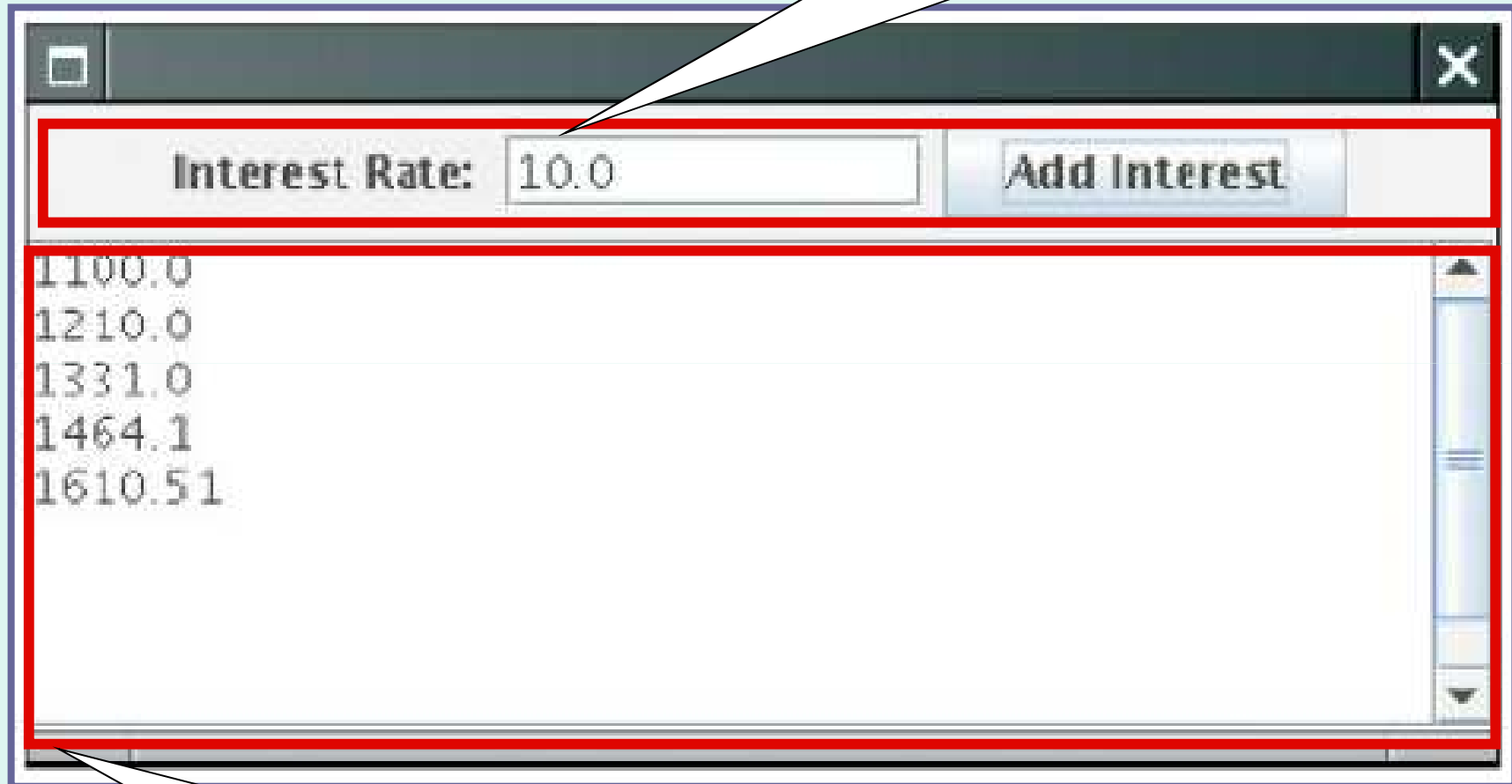
- È il layout di default per un `JFrame` (in realtà per il *content pane* del frame)
- Quando aggiungiamo una componente è necessario specificare la posizione:

```
panel.add(component, BorderLayout.NORTH);
```

- **Espande ciascuna componente fino a riempire l'are assegnata**
 - se non si vuole questo effetto è necessario includere la componente in un ulteriore pannello

Esempio

Pannello di input @ NORTH



ScrollPane @ CENTER

Eventi del mouse

- `mousePressed`, `mouseReleased`: invocati quando un pulsante del mouse viene premuto o rilasciato
- `mouseClicked`: invocato quando un pulsante del mouse viene *cliccato*
- `mouseEntered`, `mouseExited`: il mouse è entrato o è uscito dall'area della componente

Eventi del mouse

- **Catturati da MouseListeners**

```
public interface MouseListener
{
    // un metodo per ciascun mouse event su una componente
    void mousePressed(MouseEvent event);

    void mouseReleased(MouseEvent event);

    void mouseClicked(MouseEvent event);

    void mouseEntered(MouseEvent event);

    void mouseExited(MouseEvent event);
}
```

Continua...

Eventi del mouse

- **Se vogliamo che una componente reagisca ad eventi del mouse dobbiamo registrare un `MouseListener` sulla componente:**

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

Continua...

Eventi del mouse

- **Esempio: estendiamo l'applicazione che visualizza un rettangolo aggiungendo la possibilità di spostare il rettangolo attraverso un click del mouse**
- **Il rettangolo si sposta nella posizione sulla quale il mouse avviene il click del mouse**

Disegno di forme geometriche

- **JComponent**
 - La classe che definisce contenitori generici, al cui interno includiamo forme geometriche
- **Possiamo definire nuovi JComponent**

```
class ShapeComponent extends JComponent
{
    ...
}
```

Disegno di forme geometriche

- `paintComponent`
 - metodo che produce il disegno
 - invocato (automaticamente) tutte le volte che il componente per il quale è definito viene ridisegnato

Disegno di forme geometriche

- `paintComponent(Graphics g)`
 - `g` contesto grafico
 - `Graphics` classe che permette di manipolare lo stato del contesto grafico

```
class ShapeComponent extends JComponent
{
    public void paintComponent(Graphics g)
    {
        // Converti in Graphics2D
        Graphics2D g2 = (Graphics2D) g;
        ...
    }
}
```


File RectangleComponent.java

```
import java.awt.*;
import javax.swing.JComponent;
/** Sposta il rettangolo con un click del mouse. */
public class RectangleComponent extends JComponent
{
    public RectangleComponent()
    {
        box = new Rectangle(BOX_X, BOX_Y, BOX_WIDTH, BOX_HEIGHT);
    }
    public void moveTo(int x, int y)
    {
        box.setLocation(x, y);
    }
    public void paintComponent(Graphics g)
    {
        ((Graphics2D)g).draw(box);
    }
}
```

[Continua...](#)

File RectangleComponent.java

```
private Rectangle box;
```

```
private static final int BOX_X = 100;
```

```
private static final int BOX_Y = 100;
```

```
private static final int BOX_WIDTH = 20;
```

```
private static final int BOX_HEIGHT = 30;
```

```
}
```

Eventi del mouse

- **Ora definiamo il mouse listener: quando premiamo il mouse il rettangolo si sposta raggiungendo il mouse**

Continued...

Mouse Events

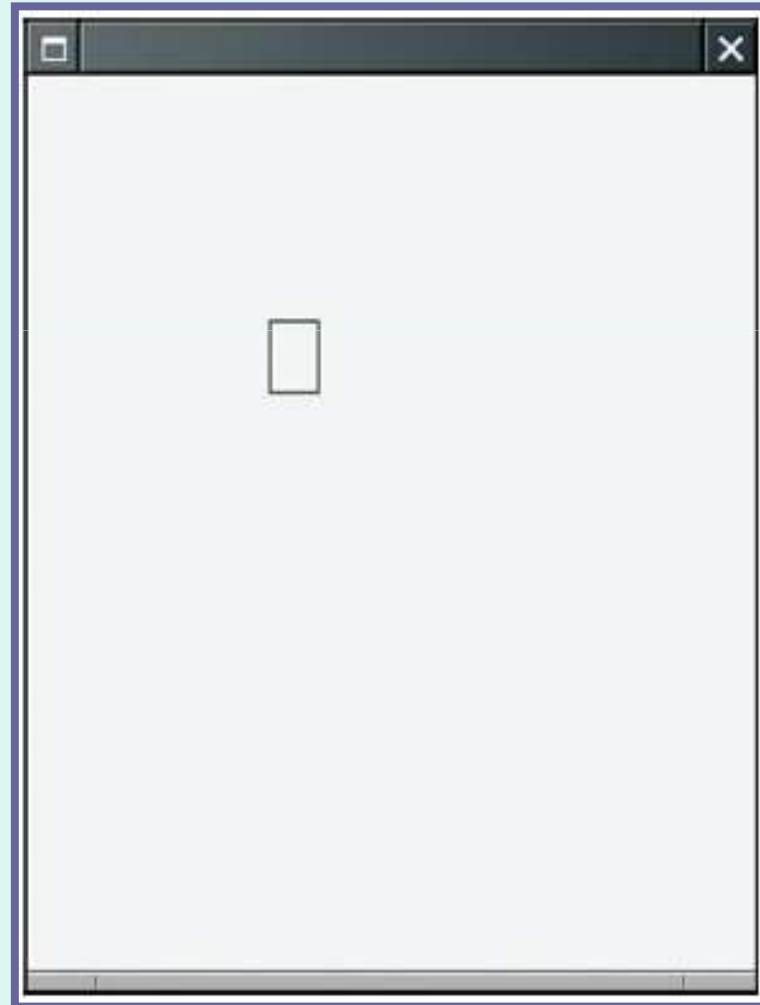
```
class MousePressListener extends MouseAdapter
{
    public void mousePressed(MouseEvent event)
    {
        int x = event.getX();
        int y = event.getY();
        component.moveTo(x, y);

        component.repaint();
    }
}
```

- **Tutti i metodi devono essere implementati; implementazione vuota se inutili (meglio usare un *Adapter*)**

RectangleMover

- Output



File RectangleMover.java

```
import java.awt.event.*;
import javax.swing.JFrame;
/**
    RectangleComponent interattivo
 */
public class RectangleMover
{
    public static void main(String[] args)
    {
        final RectangleComponent component
            = new RectangleComponent();
    }
}
```

[Continua...](#)

File RectangleMover.java

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

Continued...

File RectangleMover.java

```
// Registra il listener
MouseListener listener = new MousePressListener();
component.addMouseListener(listener);
// Crea e disponi il frame
JFrame frame = new JFrame();
frame.add(component);

frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
}

private static final int FRAME_WIDTH = 300;
private static final int FRAME_HEIGHT = 400;
}
```


Domande

7. Cosa succederebbe se omettessimo la chiamata a `repaint` nel listener?
8. Perché nella classe `MouseListener` dobbiamo definire tutti i metodi?

Risposte

7. Non avremmo l'effetto dell'animazione perché il rettangolo verrebbe ridisegnato solo in occasione di eventi del frame.
8. Perché implementa l'interfaccia `MouseListener` interface, che ha i cinque metodi