# Programmazione ad oggetti

# Layouts

A.A. 2022/2023

*Docente: Prof. Salvatore D'Angelo*
*Email: salvatore.dangelo@unicampania.it*

Università degli Studi della Campania *Luigi Vanvitelli*
*Dipartimento di Ingegneria*

# Layouts

Un layout manager è una modalità di posizionamento dei componenti in un pannello o in un altro contenitore .

In Java, ogni pannello ed qualunque componente GUI può essere un **Container**.

Un particolare layout viene impostato chiamando il metodo **setLayout** del container.

# Layouts

.

AWT/Swing supportano diversi managers. Ne considereremo 4:

- FlowLayout,
- GridLayout,
- BorderLayout,
- BoxLayout.

Tutte queste classi implementano l'interfaccia **java.awt.LayoutManager**.

# FlowLayout

Posiziona i componenti in linea fino a quando rientrano nelle dimensioni orizzontali della finestra, quindi comincia una nuova linea.

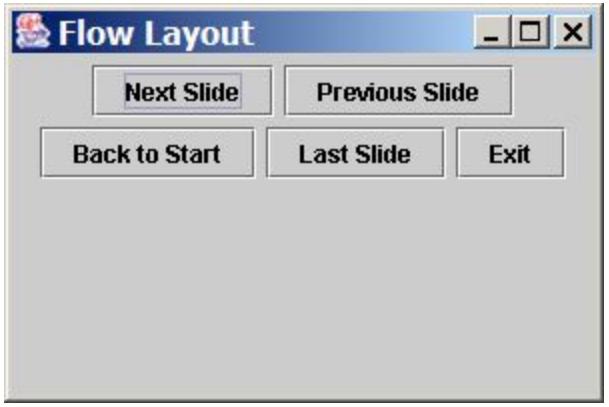
Cerca di ottimizzare lo spazio nel disporre i componenti. Per default utilizza un allineamento centrato.

Lascia che ogni componente conservi la dimensione naturale (preferred).

Spesso è utilizzato per inserire pulsanti in un pannello.

# FlowLayout

```
Container c = getContentPane();
c.setLayout (new FlowLayout() );
c.add (new JButton ("Next Slide"));
c.add (new JButton ("Previous Slide"));
c.add (new JButton ("Back to Start"));
c.add (new JButton ("Exit"));
```



# GridLayout

Divide il pannello in una griglia in un dato numero di righe e colonne.

Inserisce i componenti nelle celle della griglia.

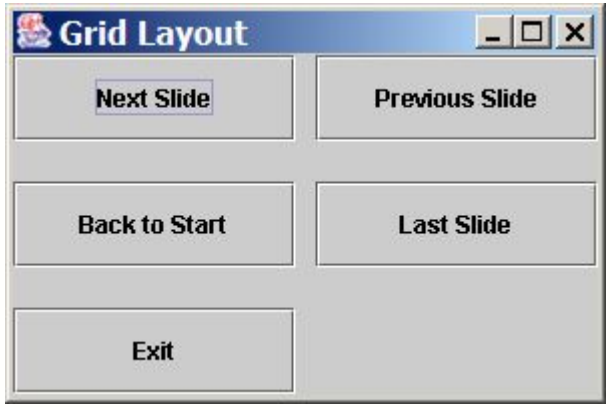
Forza le dimensioni del componente ad occupare l'intera cella.

Permette di inserire spazi addizionali tra le celle.

# GridLayout (cont'd)

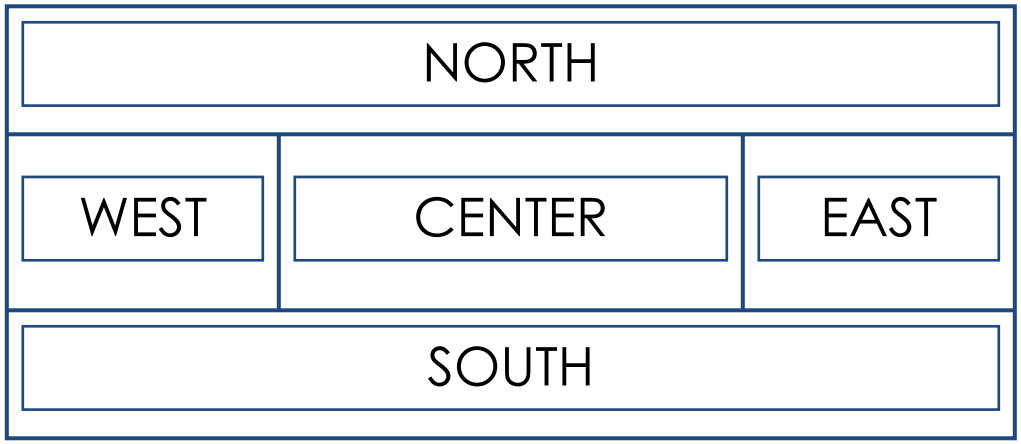
```
Container c = getContentPane();
c.setLayout (new GridLayout(3, 2,  10, 20 ));
c.add (new JButton ("Next Slide"));
c.add (new JButton ("Previous Slide"));
c.add (new JButton ("Back to Start"));
c.add (new JButton ("Last Slide"));
c.add (new JButton ("Exit"));
```

Extra space between the cells



# BorderLayout

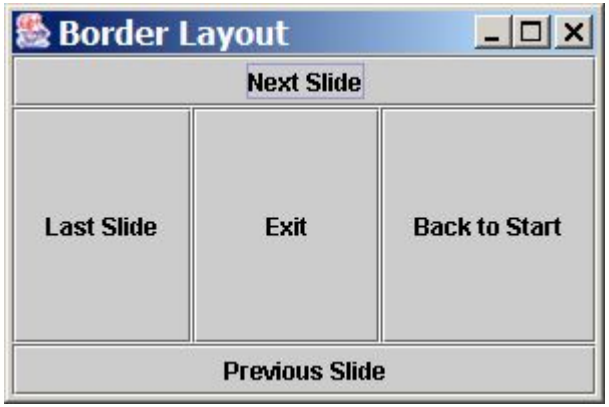
Divide l'area in 5 reggioni ed aggiunge un componente ad ogni regione.




Forza le dimensioni del componente ad occupare l'intera regione.

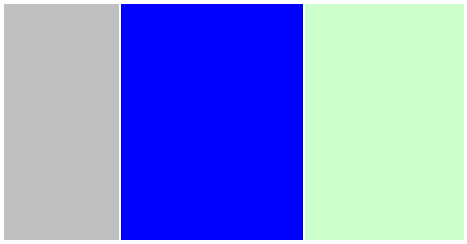
# BorderLayout (cont'd)

```
Container c = getContentPane();
c.setLayout(new BorderLayout() );
c.add (new JButton ("Next Slide"), BorderLayout.NORTH);
c.add (new JButton ("Previous Slide"), BorderLayout.SOUTH);
c.add (new JButton ("Back to Start"), BorderLayout.EAST);
c.add (new JButton ("Last Slide"), BorderLayout.WEST);
c.add (new JButton ("Exit"), BorderLayout.CENTER);
```

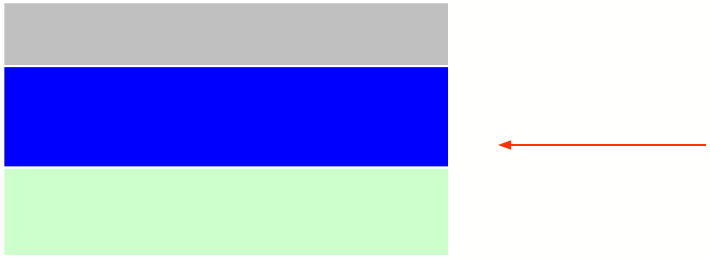


# BoxLayout

In una box orizzontaIe i componenti sono posizionati da sinistra a destra.



In una box verticale i componenti sono posizionati dall'alto verso il basso.



"Horizontal" o "vertical" non hanno niente a che fare con la forma della box.

# BoxLayout (cont'd)

BoxLayout è il layout di default per un container di classe Box.

Il linguaggio per utilizzare le Box e un po' differente:

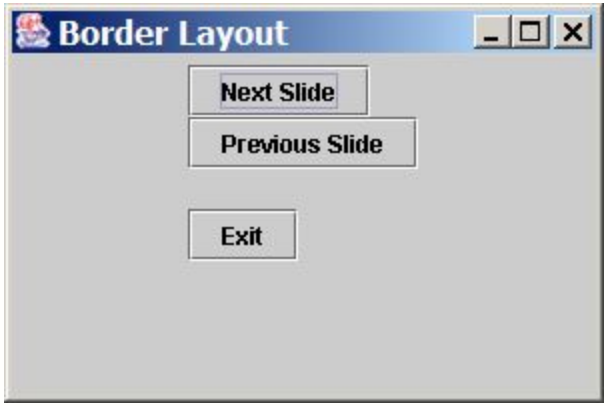
```
Box box1 = Box.createHorizontalBox();
box1. add (...);

 // add a spacer, 60 pixels:
 box1.add(Box.createHorizontalStrut (60);

 Box box2 = Box.createVerticalBox();
 ...
```

# BoxLayout

```
Container c = getContentPane();
c.setLayout(new FlowLayout() );
Box box = Box.createVerticalBox();
box.add (new JButton ("Next Slide"));
box.add (new JButton ("Previous Slide"));
box.add (Box.createVerticalStrut (20) );
box.add (new JButton ("Exit"));
c.add (box);
```



# Default Layouts*

Ogni componente ha un layout di default che rimane tale fino a che non viene utilizzato il metodo  setLayout.

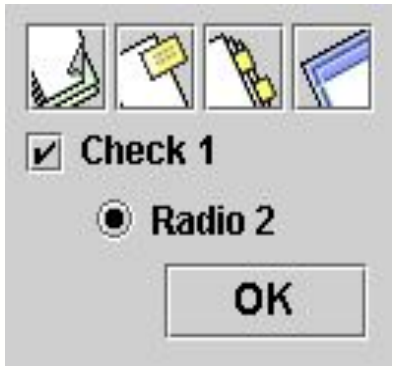
Alcuni layout di default sono:

| | |
|---|---|
| Content pane | ← BorderLayout |
| JPanel | ← FlowLayout |
| Box | ← BoxLayout |

# BUTTONS


Toggle
Enabled
Disabled

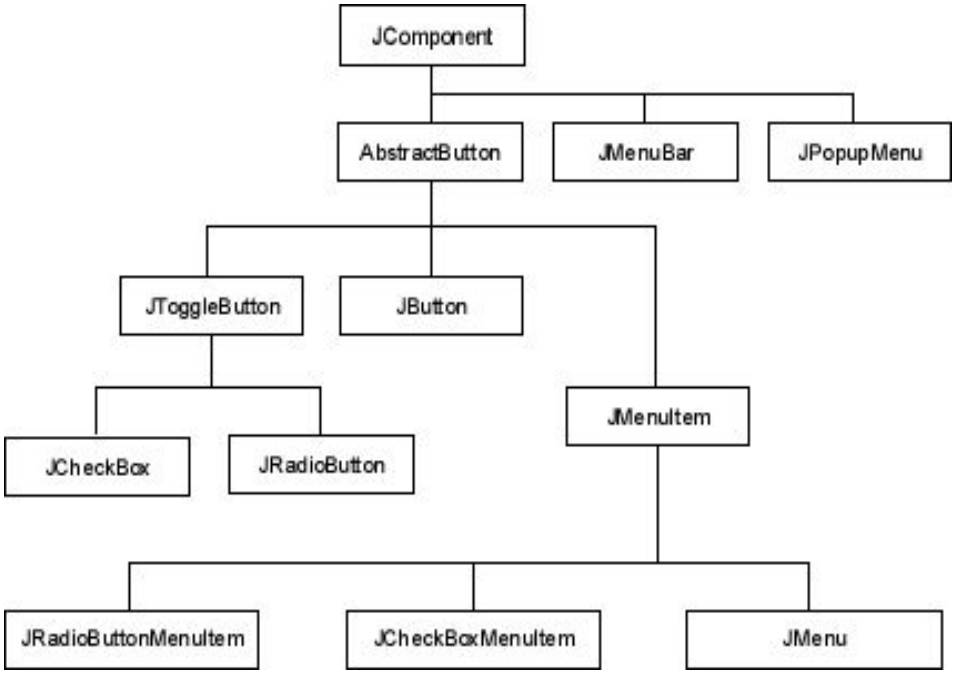

JComponent
AbstractButton
JToggleButton
JButton
JCheckBox
JRadioButton

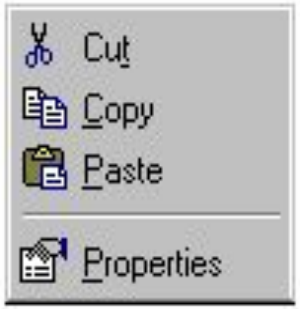

Check 1
Radio 2
OK


Radio 1
Radio 2
Radio 3
Radio 4

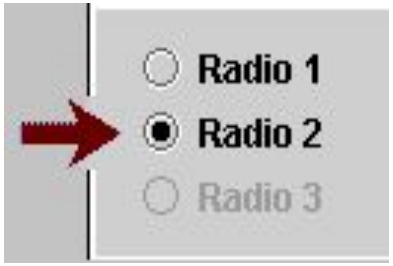

Check 1
Check 2
Check 3
Check 4

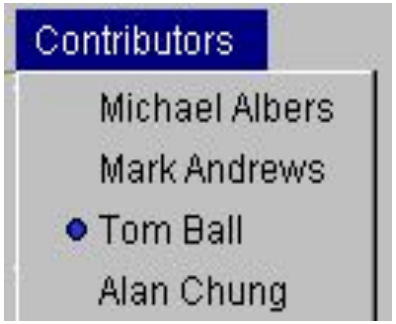
# MENUS


JComponent
AbstractButton
JMenuBar
JPopupMenu
JToggleButton
JButton
JCheckBox
JRadioButton
JMenuItem
JRadioButtonMenuItem
JCheckBoxMenuItem
JMenu


Cut
Copy
Paste
Properties

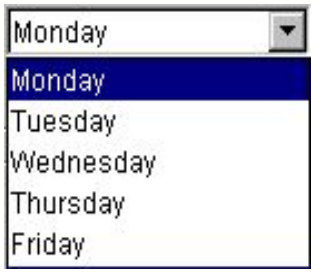

Text
Help
Settings
Font


Radio 1
Radio 2
Radio 3


Contributors
Michael Albers
Mark Andrews
Tom Ball
Alan Chung


Theme
Help
metal
ctrl-m
Organic
ctrl-o
metal2
ctrl-2

# OTHER COMPONENTS




JComboBox


JApplet


Border Interface

JColorChooser

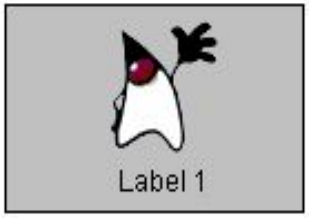

ImageIcon


JDialog


JFileChooser


JInternalFrame

# OTHER COMPONENTS



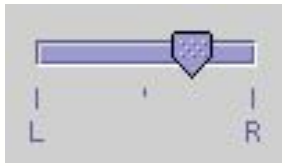

JLabel


JList

JScrollBar

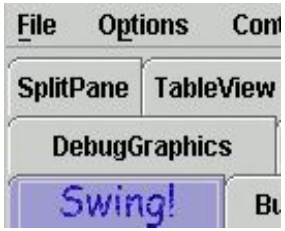

JScrollPane


JOptionPane



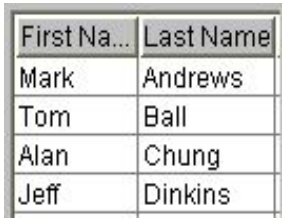

JSlider


JSplitPane

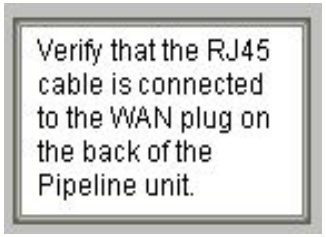



JTabbedPane

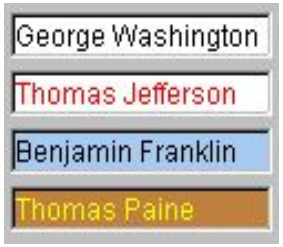
# OTHER COMPONENTS




**JTable**




**JTextArea**




**JTextField**

**JToolBar**


**JToolTip**


**JTree**

# Atomic components (3)

Impossibile spiegare il comportamento di tutti i componenti

Poche persone conoscono tutto!! – Swing è vasto.

Riferimenti:

- Java 2 API Documentation.
- http://java.sun.com/docs/books/tutorial/uiswing/components/components.html

# Pulsante con immagine

Al posto delle semplici scritte possiamo aggiungere grafica ai pulsanti con oggetti di tipo **ImageIcon** da incorporare nei bottoni stessi

```
ImageIcon img=new ImageIcon("c:\\...\\ img1.gif ");
// legge il file .gif e lo incorpora
// nell'oggetto img
JButton pulsanteUno = new Jbutton(img);
// passo img al costruttore del pulsante
```

# Label

Componente che contiene semplicemente del testo informativo (di solo output)
Può contenere anche solo un'icona o entrambi.
I costruttori di base sono :

- **JLabel(String);** Crea un'etichetta con la stringa passata come parametro
- **JLabel(String, int);** Crea un'etichetta con la stringa passata come parametro definendone l'allineamento a mezzo di un flag
- **JLabel(String, Icon, int);** Crea un'etichetta con la stringa passata come parametro e l'icona associata. definendone l'allineamento a mezzo di un flag

# Label: Allineamento

I tre flag disponibili sono:
**SwingConstants.LEFT**
**SwingConstants.RIGHT**
**SwingConstants.CENTER** (default)

# Campo testo: JTextField

Componente che gestisce una riga di input da tastiera

Costruttori :

- **JtextField();** Genera un campo di testo vuoto
- **JtextField(int);** Genera un campo di testo di larghezza specificata
- **JtextField(String, int);** Genera un campo di testo contenente una stringa e di larghezza specificata

# JTextField

**setEditable(boolean);** posto a “true” permette la modifica del testo contenuto, al contrario “false” non permette l'immissione di input da tastiera

**boolean isEditable();** dice se il componente è modificabile o meno

**setText(String);** permette di immettere testo nel componente

**String getText();** restituisce il contenuto in forma di stringa

**String getSelectedText();** restituisce il solo testo selezionato dall'utente

# Inserire passwords

Utile classe con la capacità di crittare l'output a video con un carattere a nostra scelta durante la digitazione.

Permette di definirne le dimensioni e mediante un opportuno metodo impostare il carattere per l'echo a video

**JPasswordField pf = new JPasswordField(10);**

- istanzia un oggetto campo password di dimensione 10

**pf.setEchoChar('*');**

- il classico asterisco

# JTextArea

Rappresenta un campo di testo in forma matriciale, con un certo numero di righe e di colonne. I costruttori sono

**JTextArea(int, int);**

- Crea un'area di testo di dimensioni righe, colonne

**JTextArea(String, int ,int);**

- Come sopra solo che inserisce una stringa da codice

I metodi sono **getText(); getSelectedText()** e **setText(String)** funzionanti come per i campi di testo

# JTextArea

Altre funzionalità di editing:

**append(String);**
- accoda del testo

**insert(String, int);**
- inserisce del testo alla posizione specificata

**setLineWrap( boolean);**
- specifica se “true” di andare a capo automaticamente a fine riga

**setWrapStyleWord( boolean );**
- va a capo con la parola se “true” o col singolo carattere se “false”

# Esempio

```
import javax.swing.*;
public class MyJLabelJTextFieldJFrame extends JFrame {
  JLabel etichetta = new JLabel("Etichetta");
  JTextField campoDiTesto = new JTextField("Scrivi qui", 30);
  JTextArea areaDiTesto = new JTextArea("Questa é un'area “ +“di
testo di\n6 righe e 20 colonne", 6, 20);

public MyJLabelJTextFieldJFrame() {
   super("Finestra con Etichette e Campi");
   setSize(350, 300);
   setDefaultCloseOperation(EXIT_ON_CLOSE);
   JPanel pannello = new JPanel();
   // impostiamo le proprietà dei componenti
   campoDiTesto.setEditable(true);
   areaDiTesto.setLineWrap(true);
   areaDiTesto.setWrapStyleWord(true);
```

# Esempio

```
// ora aggiungiamo i componenti al pannello
pannello.add(etichetta);
pannello.add(campoDiTesto);
pannello.add(areaDiTesto);
// rendiamo il pannello parte del nostro frame
setContentPane(pannello);
// Visualizziamo il tutto!
show();
}
public static void main(String argv[]) {
    MyJLabelJTextFieldJFrame ec =
            new MyJLabelJTextFieldJFrame();
}
}
```

# Esempi

Esempi:

- Pulsanti
- PulsantiGrid
- PulsantiBorder
- PulsantiVBox
- JCompositeFrame
- Semplice Login

# Esempio

Realizzare la finestra sottostante formata da un pannello ed un pulsante

# Realizzare la finestra sottostante formata da più pannelli


Realizzare la finestra a lato costituita da tre pannelli ed un pulsante

# Gestione eventi

Realizzazione di una interfaccia grafica:

- Disegno grafico: CAD
- Gestione degli eventi Scrittura codice

Ogni componente Swing è in grado di generare degli eventi generici o particolari (Ad esempio per il pulsante non siamo particolarmente interessati al fatto che sia attraversato dal mouse, ma solo al click)

Quindi cominceremo col gestire per ogni componente gli eventi principali

# JButton

Per registrare un gestore degli eventi per un pulsante occorre utilizzare il metodo:

**addActionListener(ActionListener a);**

del componente JButton.

Tale metodo si aspetta come parametro una classe in grado ascoltare gli eventi generati dal pulsanti e di gestirli

Tale classe per poter essere utilizzata da JButton deve implementare l'interfaccia ActionListener

# ActionListener interface

```
class ButtonListener implements ActionListener {
   public void actionPerformed(ActionEvent e) {
      ……
   }

}
```

Quando viene pigiato il pulsante automaticamente viene invocato il metodo ActionPerformed

# Esempio

```
public class Button2 extends JFrame {
   private JButton b1 = new JButton("Button 1");
   JButton b2 = new JButton("Button 2");
   private JTextField txt = new JTextField(10);

  public Button2(){
       super(“2 pulsanti”);
       setSize(400,300);
       JPanel p=new JPanel();
       b1.addActionListener(new ButtonListener());
       b2.addActionListener(new ButtonListener());
       p.add(b1); p.add(b2); p.add(txt);}

public static void main(String[] args){
  JFrame frame=new Button2();
  frame.show();
 }}
```

# Esempio

```
class ButtonListener implements ActionListener {
   JTextField txt;

   //public ButtonListener(JTextField t){txt=t;}

   public void actionPerformed(ActionEvent e) {
         String name =
((JButton)e.getSource()).getText();
         txt.setText(name);
   }
}
```

# Generalizziamo

In conclusione per gestire un evento occorre:

1. Definire una classe che implementa una interfaccia Listener
2. Ridefinire tutti i metodi dell'interfaccia
3. Aggiungere il Listener al componente

# Tipi di eventi

| Event, listener interface and add- and remove-methods | Components supporting this event |
|---|---|
| **ActionEvent**<br>**ActionListener**<br>**addActionListener( )**<br>**removeActionListener( )** | **JButton, JList, JTextField, JMenuItem** and its derivatives including **JCheckBoxMenuItem, JMenu,** and **JpopupMenu.** |
| **AdjustmentEvent**<br>**AdjustmentListener**<br>**addAdjustmentListener( )**<br>**removeAdjustmentListener( )** | **JScrollbar**<br>and anything you create that implements the **Adjustable interface.** |
| **ComponentEvent**<br>**ComponentListener**<br>**addComponentListener( )**<br>**removeComponentListener( )** | ***Component** and its derivatives, including **JButton, JCheckBox, JComboBox, Container, JPanel, JApplet, JScrollPane, Window, JDialog, JFileDialog, JFrame, JLabel, JList, JScrollbar, JTextArea,** and **JTextField.** |

# Tipi di eventi

| | |
|---|---|
| **ContainerEvent**<br>**ContainerListener**<br>**addContainerListener( )**<br>**removeContainerListener( )** | **Container** and its derivatives, including **JPanel, JApplet, JScrollPane, Window, JDialog, JFileDialog,** and **JFrame.** |
| **FocusEvent**<br>**FocusListener**<br>**addFocusListener( )**<br>**removeFocusListener( )** | **Component** and **derivatives*.** |
| **KeyEvent**<br>**KeyListener** | **Component** and **derivatives*.** |

# Tipi di eventi

| Event, listener interface and add- and remove-methods | Components supporting this event |
|---|---|
| **addKeyListener( )**<br>**removeKeyListener( )** | |
| **MouseEvent** (for both clicks and motion)<br>**MouseListener**<br>**addMouseListener( )**<br>**removeMouseListener( )** | **Component** and **derivatives*.** |
| **MouseEvent**[8] (for both clicks and motion)<br>**MouseMotionListener**<br>**addMouseMotionListener( )**<br>**removeMouseMotionListener( )** | **Component** and **derivatives*.** |
| **WindowEvent**<br>**WindowListener**<br>**addWindowListener( )**<br>**removeWindowListener( )** | **Window** and its derivatives, including **JDialog, JFileDialog, and JFrame.** |
| **ItemEvent**<br>**ItemListener**<br>**addItemListener( )**<br>**removeItemListener( )** | **JCheckBox, JCheckBoxMenuItem, JComboBox, JList,** and anything that implements the **ItemSelectable interface.** |
| **TextEvent**<br>**TextListener**<br>**addTextListener( )**<br>**removeTextListener( )** | Anything derived from **JTextComponent,** including **JTextArea** and **JTextField.** |

# Listeners e interfacce

| Listener interface w/ adapter | Methods in interface |
|---|---|
| ActionListener | actionPerformed(ActionEvent) |
| AdjustmentListener | adjustmentValueChanged( AdjustmentEvent) |
| ComponentListener<br>ComponentAdapter | componentHidden(ComponentEvent)<br>componentShown(ComponentEvent)<br>componentMoved(ComponentEvent)<br>componentResized(ComponentEvent) |
| ContainerListener<br>ContainerAdapter | componentAdded(ContainerEvent)<br>componentRemoved(ContainerEvent) |
| FocusListener<br>FocusAdapter | focusGained(FocusEvent)<br>focusLost(FocusEvent) |
| KeyListener<br>KeyAdapter | keyPressed(KeyEvent)<br>keyReleased(KeyEvent)<br>keyTyped(KeyEvent) |

# Tipi di eventi

| | |
|---|---|
| MouseListener<br>MouseAdapter | mouseClicked(MouseEvent)<br>mouseEntered(MouseEvent)<br>mouseExited(MouseEvent)<br>mousePressed(MouseEvent)<br>mouseReleased(MouseEvent) |
| MouseMotionListener<br>MouseMotionAdapter | mouseDragged(MouseEvent)<br>mouseMoved(MouseEvent) |
| WindowListener<br>WindowAdapter | windowOpened(WindowEvent)<br>windowClosing(WindowEvent)<br>windowClosed(WindowEvent)<br>windowActivated(WindowEvent)<br>windowDeactivated(WindowEvent)<br>windowIconified(WindowEvent)<br>windowDeiconified(WindowEvent) |
| ItemListener | itemStateChanged(ItemEvent) |

# Utilizzare listener adapters

Se volessi implementare una interfaccia listener dovrei impazzire perché non dovrei dimenticare nessun metodo

Per ogni listener esiste un adapter che implementa l'interfaccia corrispondente con tutti metodi vuoti

Esempio:

```
class MyMouseListener extends MouseAdapter {
    public void MouseClicked(MouseEvent e) {
    // Respond to mouse click...
}
```