



# Interfacce grafiche

---

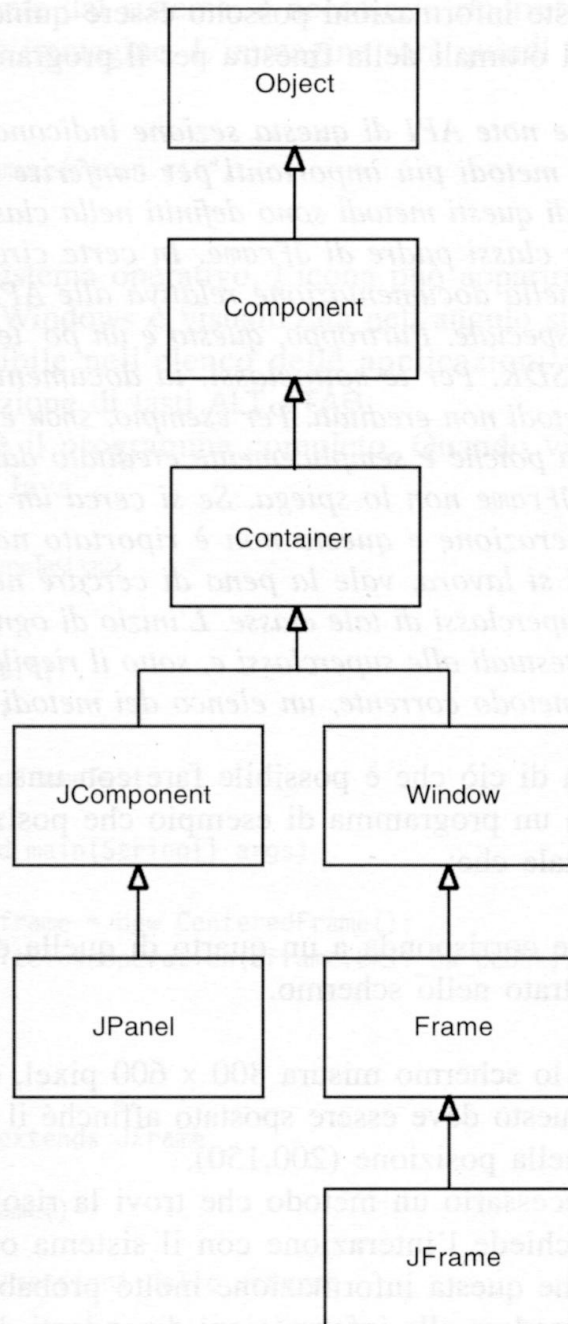


# Ereditarietà per personalizzare i frame

---

- Usare l'ereditarietà per scomporre frames complessi in unità facilmente comprensibili
- Progettare sottoclassi di **JFrame**
- Memorizzare le componenti come variabili di istanza
- Inizializzare le variabili nei costruttori delle sottoclassi
- Se l'inizializzazione diventa complessa utilizza alcuni metodi di servizio

# Gerarchia di ereditarietà



- La classe **JFrame** in sé contiene solo alcuni metodi per modificare l'aspetto dei frame
- Quasi tutti i metodi per lavorare con le dimensioni e con la posizione di un frame provengono dalle varie superclassi di **JFrame**
  - Le classi **Component** e **Window** contengono i metodi per modificare le dimensioni e la forma dei frame

# Esempio: programma visualizzazione investimento

---

- Progettare sottoclassi di **JFrame** e memorizzare componenti nelle variabili di istanza, inizializzandole nel costruttore della vostra sottoclasse
- Usando anche metodi ausiliari se il codice del costruttore diventa troppo complesso
- Ad Esempio:



# Esempio: programma visualizzazione investimento

---

```
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 InvestmentFrame extends JFrame
13: {
14:     public InvestmentFrame()
15:     {
16:         account = new BankAccount(INITIAL_BALANCE);
17:
```

# Esempio: continuazione

---

```
18:         // Use instance fields for components
19:         resultLabel = new JLabel(
20:             "balance=" + account.getBalance());
21:
22:         // Use helper methods
23:         createRateField();
24:         createButton();
25:         createPanel();
26:
27:         setSize(FRAME_WIDTH, FRAME_HEIGHT);
28:     }
29:
30:     public void createRateField()
31:     {
32:         rateLabel = new JLabel("Interest Rate: ");
33:         final int FIELD_WIDTH = 10;
34:         rateField = new JTextField(FIELD_WIDTH);
```



# Esempio: continuazione

---

```
53:         }
54:     }
55:
56:     ActionListener listener = new AddInterestListener();
57:     button.addActionListener(listener);
58: }
59:
60: public void createPanel()
61: {
62:     JPanel panel = new JPanel();
63:     panel.add(rateLabel);
64:     panel.add(rateField);
65:     panel.add(button);
66:     panel.add(resultLabel);
67:     add(panel);
68: }
69:
```



# Esempio: continuazione

---

```
70:     private JLabel rateLabel;
71:     private JTextField rateField;
72:     private JButton button;
73:     private JLabel resultLabel;
74:     private BankAccount account;
75:
76:     private static final double DEFAULT_RATE = 10;
77:     private static final double INITIAL_BALANCE = 1000;
78:
79:     private static final int FRAME_WIDTH = 500;
80:     private static final int FRAME_HEIGHT = 200;
81: }
```

# Esempio: continuazione

---

Classe con metodo `main`:

```
01: import javax.swing.JFrame;
02:
03: /**
04:     This program tests the InvestmentFrame.
05: */
06: public class InvestmentFrameViewer
07: {
08:     public static void main(String[] args)
09:     {
10:         JFrame frame = new InvestmentFrame();
11:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12:         frame.setVisible(true);
13:     }
14: }
15:
```

# Gestione del layout delle componenti

---

- Le componenti di un'interfaccia utente sono organizzate mettendole all'interno di un contenitore
  - Ad esempio **JPanel**
- Ogni contenitore ha un *layout manager* (*gestore di layout*) che si occupa del posizionamento delle sue componenti
  - Le componenti in un **JPanel** sono inserite da sinistra a destra
- Tre gestori di layout più diffusi (**java.awt**):
  - gestore di layout a bordi (**BorderLayout**)
  - gestore di layout a scorrimento (**FlowLayout**)
  - gestore di layout a griglia (**GridLayout**)

# Gestione del layout

---

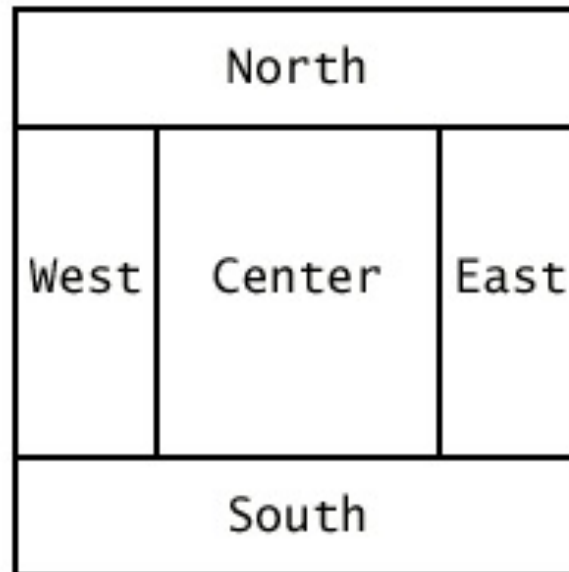
- Per default, **JPanel** organizza le componenti da sinistra a destra e comincia una nuova riga se necessario
- Il layout di **JPanel** è gestito da **FlowLayout**
  - gestore di layout a scorrimento
- Possiamo richiedere altri gestori di layout

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

# Layout a bordi (BorderLayout)

---

- Il contenitore è diviso in 5 aree:  
**center, north, west, south e east**



# BorderLayout

---

- E' il layout manager di default di un frame
- Quando si aggiunge una componente si specifica una posizione (CENTER di default):

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

- Ogni componente è estesa per coprire l'intera area allocata. Se si ha l'esigenza di condividere l'area con altri componenti, si possono inserire i componenti in un **JPanel**



# Layout a griglia

---

- Posiziona le componenti in una griglia con un numero fissato di righe e colonne
- La taglia di ogni componente viene opportunamente modificata in modo che tutte le componenti hanno la stessa taglia
- Ogni componente viene espansa in modo che occupi tutta l'area allocata

# GridLayout

---

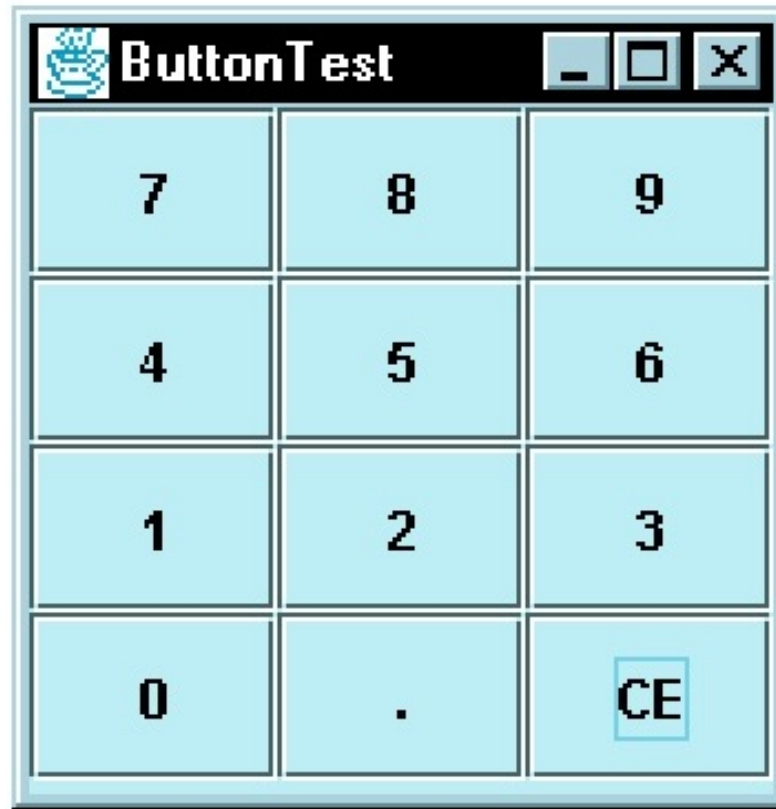
- Aggiungere le componenti, riga per riga, da sinistra a destra:

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



# GridLayout

---





# Altro tipo di layout a griglia

---

- **GridBagLayout:**

- le componenti sono disposte in una tabella
- le colonne possono avere taglie differenti
- le componenti possono ricoprire colonne multiple

- Difficile da usare

- Possiamo ovviare con **JPanel** annidati

- Ogni oggetto **JPanel** ha un gestore appropriato
- Oggetti **JPanel** non hanno bordi visibili
- Si possono usare tanti **JPanel** quanti ne servono per organizzare le componenti

# Scelte

---

- Caselle combinate
- Caselle di controllo
- Pulsanti radio



# Pulsanti radio

---

- In un insieme di pulsanti radio uno solo alla volta può essere selezionato
- Adatto ad un insieme di scelte mutuamente esclusive
- Se un pulsante è selezionato, tutti gli altri nell'insieme sono automaticamente deselezionati
- Ogni pulsante è un oggetto di **JRadioButton**
  - pacchetto **javax.swing**
  - sottoclasse di **JComponent**

# ButtonGroup

---

- Nell'esempio la taglia del font è realizzata con pulsanti radio:

```
JRadioButton smallButton = new JRadioButton("Small");  
JRadioButton mediumButton = new JRadioButton("Medium");  
JRadioButton largeButton = new JRadioButton("Large");  
// Aggiungi pulsanti radio in un ButtonGroup così  
// soltanto un pulsante nel gruppo può essere selezionato  
ButtonGroup group = new ButtonGroup();  
group.add(smallButton);  
group.add(mediumButton);  
group.add(largeButton);
```

# ButtonGroup (javax.swing)

---

- Un gruppo di pulsanti non piazza i pulsanti insieme in un contenitore (non è di tipo **JComponent**), serve solo a stabilire tra quali pulsanti la scelta deve essere mutualmente esclusiva
- **isSelected**: invocato per verificare se un pulsante è selezionato o no

```
if (largeButton.isSelected()) size = LARGE_SIZE;
```

- Prima di visualizzare un frame che contiene pulsanti radio, **setSelected(true)** deve essere invocato su un pulsante di ogni gruppo di pulsanti radio (uno dei pulsanti deve essere selezionato)

# Bordi

---

- Per default, pannelli non hanno bordi visibili
- Può essere utile aggiungere un bordo visibile
- EtchedBorder: un bordo con effetto tri-dimensionale
- Si può aggiungere un bordo a ogni componente:

```
JPanel panel = new JPanel();  
panel.setBorder(new EtchedBorder());
```

- TitledBorder: bordo con titolo

```
panel.setBorder(new TitledBorder(new EtchedBorder(), "Size"));
```

# Caselle di controllo

---

- Ogni casella ha due stati: selezionata e non selezionata
- Nei gruppi di caselle di controllo la scelta non è mutuamente esclusiva
- Esempio: "bold" e "italic" nella figura precedente
- Si costruiscono dandone il nome nel costruttore:

```
JCheckBox italicCheckBox = new JCheckBox("Italic");
```

- Non si devono mettere in un gruppo di pulsanti
- **JCheckBox** è nel pacchetto `javax.swing` ed è una sottoclasse di **JComponent**



# Caselle combinate

---

- Per grandi insiemi di scelte mutuamente esclusive
  - usa meno spazio dei pulsanti radio
- Combinazione di una lista e un campo di testo
  - Il campo di testo visualizza il nome della selezione corrente



# Caselle combinate

---

- Se la casella combinata è editabile, allora l'utente può digitare la sua selezione
- Si usa il metodo **setEditable(true)** per rendere editabile il campo di testo
- Si aggiungono le stringhe di testo con il metodo **addItem**:

```
JComboBox facenameCombo = new JComboBox();  
facenameCombo.addItem("Serif");  
facenameCombo.addItem("SansSerif");  
...
```

- **JComboBox** è nel pacchetto **javax.swing** ed è una sottoclasse di **JComponent**

# Caselle combinate

---

- La selezione dell'utente si prende con **getSelectedItem** (restituisce un **Object**)

```
String selectedString =  
    (String) facenameCombo.getSelectedItem();
```

- Seleziona un elemento della lista da visualizzare all'inizio con **setSelectedItem** (**anObject**)

# Nota

---

- Pulsanti radio, caselle di controllo e caselle combinate di un frame generano un **ActionEvent** ogni volta che l'utente seleziona un elemento
- Nel programma che segue:
  - Tutte le componenti notificano l'evento allo stesso listener
  - Quando un utente clicca su una componente, si chiede alla componente il suo contenuto corrente
  - Quindi riscriviamo il testo campione con la scelta corrente

# I componenti di choiceFrame

---

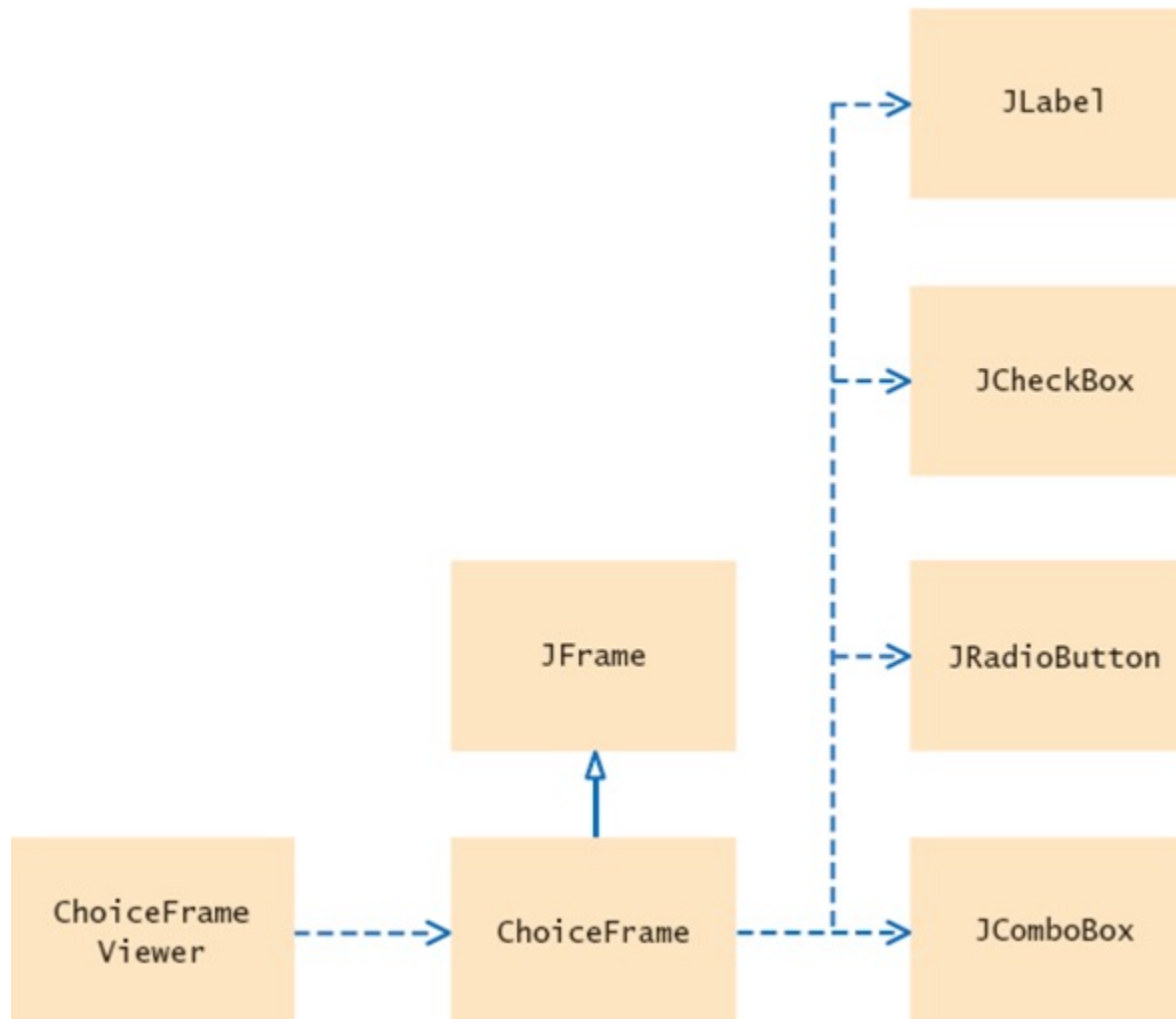
JLabel in posizione  
Center

JPanel in posizione  
South gestito da  
GridLayout



# Classi del programma scelta font

---



# File ChoiceFrameViewer.java

---

```
01: import javax.swing.JFrame;
02:
03: /**
04:     This program tests the ChoiceFrame.
05: */
06: public class ChoiceFrameViewer
07: {
08:     public static void main(String[] args)
09:     {
10:         JFrame frame = new ChoiceFrame();
11:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12:         frame.setVisible(true);
13:     }
14: }
15:
```

# File ChoiceFrame.java

---

```
001: import java.awt.BorderLayout;
002: import java.awt.Font;
003: import java.awt.GridLayout;
004: import java.awt.event.ActionEvent;
005: import java.awt.event.ActionListener;
006: import javax.swing.ButtonGroup;
007: import javax.swing.JButton;
008: import javax.swing.JCheckBox;
009: import javax.swing.JComboBox;
010: import javax.swing.JFrame;
011: import javax.swing.JLabel;
012: import javax.swing.JPanel;
013: import javax.swing.JRadioButton;
014: import javax.swing.border.EtchedBorder;
015: import javax.swing.border.TitledBorder;
016:
```



# File ChoiceFrame.java

---

```
017: /**
018:     This frame contains a text field and a control panel
019:     to change the font of the text.
020: */
021: public class ChoiceFrame extends JFrame
022: {
023:     /**
024:         Constructs the frame.
025:     */
026:     public ChoiceFrame()
027:     {
028:         // Construct text sample
029:         sampleField = new JLabel("Big Java");
030:         add(sampleField, BorderLayout.CENTER);
031:
```

# File ChoiceFrame.java

---

```
032:         // This listener is shared among all components
033:         class ChoiceListener implements ActionListener
034:         {
035:             public void actionPerformed(ActionEvent event)
036:             {
037:                 setSampleFont();
038:             }
039:         }
040:
041:         listener = new ChoiceListener();
042:
043:         createControlPanel();
044:         setSampleFont();
045:         setSize(FRAME_WIDTH, FRAME_HEIGHT);
046:     }
047:
```

# File ChoiceFrame.java

---

```
048:    /**
049:        Creates the control panel to change the font.
050:    */
051:    public void createControlPanel()
052:    {
053:        JPanel facenamePanel = createComboBox();
054:        JPanel sizeGroupPanel = createCheckBoxes();
055:        JPanel styleGroupPanel = createRadioButtons();
056:
057:        // Line up component panels
058:
059:        JPanel controlPanel = new JPanel();
060:        controlPanel.setLayout(new GridLayout(3, 1));
061:        controlPanel.add(facenamePanel);
062:        controlPanel.add(sizeGroupPanel);
063:        controlPanel.add(styleGroupPanel);
064:
```

# File ChoiceFrame.java

---

```
065:         // Add panels to content pane
066:
067:         add(controlPanel, BorderLayout.SOUTH);
068:     }
069:
070:     /**
071:      * Creates the combo box with the font style choices.
072:      * @return the panel containing the combo box
073:      */
074:     public JPanel createComboBox()
075:     {
076:         facenameCombo = new JComboBox();
077:         facenameCombo.addItem("Serif");
078:         facenameCombo.addItem("SansSerif");
079:         facenameCombo.addItem("Monospaced");
080:         facenameCombo.setEditable(true);
081:         facenameCombo.addActionListener(listener);
082:
```

# File ChoiceFrame.java

---

```
083:         JPanel panel = new JPanel();
084:         panel.add(facenameCombo);
085:         return panel;
086:     }
087:
088:     /**
089:         Creates the check boxes for selecting bold and
            // italic styles.
090:         @return the panel containing the check boxes
091:     */
092:     public JPanel createCheckBoxes()
093:     {
094:         italicCheckBox = new JCheckBox("Italic");
095:         italicCheckBox.addActionListener(listener);
096:
097:         boldCheckBox = new JCheckBox("Bold");
098:         boldCheckBox.addActionListener(listener);
099:
```

# File ChoiceFrame.java

---

```
100:         JPanel panel = new JPanel();
101:         panel.add(italicCheckBox);
102:         panel.add(boldCheckBox);
103:         panel.setBorder
104:             (new TitledBorder(new EtchedBorder(), "Style"));
105:
106:         return panel;
107:     }
108:
109:     /**
110:      * Creates the radio buttons to select the font size
111:      * @return the panel containing the radio buttons
112:      */
113:     public JPanel createRadioButtons()
114:     {
115:         smallButton = new JRadioButton("Small");
116:         smallButton.addActionListener(listener);
```

# File ChoiceFrame.java

---

```
117:
118:     mediumButton = new JRadioButton("Medium");
119:     mediumButton.addActionListener(listener);
120:
121:     largeButton = new JRadioButton("Large");
122:     largeButton.addActionListener(listener);
123:     largeButton.setSelected(true);
124:
125:     // Add radio buttons to button group
126:
127:     ButtonGroup group = new ButtonGroup();
128:     group.add(smallButton);
129:     group.add(mediumButton);
130:     group.add(largeButton);
131:
```

# File ChoiceFrame.java

---

```
132:         JPanel panel = new JPanel();
133:         panel.add(smallButton);
134:         panel.add(mediumButton);
135:         panel.add(largeButton);
136:         panel.setBorder
137:             (new TitledBorder(new EtchedBorder(), "Size"));
138:
139:         return panel;
140:     }
141:
142:     /**
143:         Gets user choice for font name, style, and size
144:         and sets the font of the text sample.
145:     */
146:     public void setSampleFont()
147:     {
```



# File ChoiceFrame.java

---

```
148:         // Get font name
149:         String facename
150:             = (String) facenameCombo.getSelectedItem();
151:
152:         // Get font style
153:
154:         int style = 0;
155:         if (italicCheckBox.isSelected())
156:             style = style + Font.ITALIC;
157:         if (boldCheckBox.isSelected())
158:             style = style + Font.BOLD;
159:
160:         // Get font size
161:
162:         int size = 0;
163:
```

# File ChoiceFrame.java

---

```
164:         final int SMALL_SIZE = 24;
165:         final int MEDIUM_SIZE = 36;
166:         final int LARGE_SIZE = 48;
167:
168:         if (smallButton.isSelected())
169:             size = SMALL_SIZE;
170:         else if (mediumButton.isSelected())
171:             size = MEDIUM_SIZE;
172:         else if (largeButton.isSelected())
173:             size = LARGE_SIZE;
174:
175:         // Set font of text field
176:
177:         sampleField.setFont(new Font(facename, style, size));
178:         sampleField.repaint();
179:     }
```

# File ChoiceFrame.java

---

```
180:
181:     private JLabel sampleField;
182:     private JCheckBox italicCheckBox;
183:     private JCheckBox boldCheckBox;
184:     private JRadioButton smallButton;
185:     private JRadioButton mediumButton;
186:     private JRadioButton largeButton;
187:     private JComboBox facenameCombo;
188:     private ActionListener listener;
189:
190:     private static final int FRAME_WIDTH = 300;
191:     private static final int FRAME_HEIGHT = 400;
192: }
```

# Osservazioni sulla gestione del layout

---

- Passo 1: Annotare su un foglio il layout delle componenti desiderate

Size

☒ Small

☐ Medium

☐ Large

☒ Pepperoni

☒ Anchovies

Your Price:

# Osservazioni sulla gestione del layout

---

- Passo 2: Raggruppare componenti adiacenti con lo stesso layout

The diagram illustrates a pizza order form layout. It features a large outer container with a purple border. Inside, there are three distinct groups of components, each enclosed in a red border:

- Size Group:** Located in the top-left, it contains the label "Size" followed by three radio button options: "Small" (selected), "Medium", and "Large".
- Toppings Group:** Located in the top-right, it contains two checkbox options: "Pepperoni" and "Anchovies", both of which are currently unchecked.
- Price Group:** Located at the bottom, it contains the text "Your Price:" followed by an empty rectangular input field.

# Osservazioni sulla gestione del layout

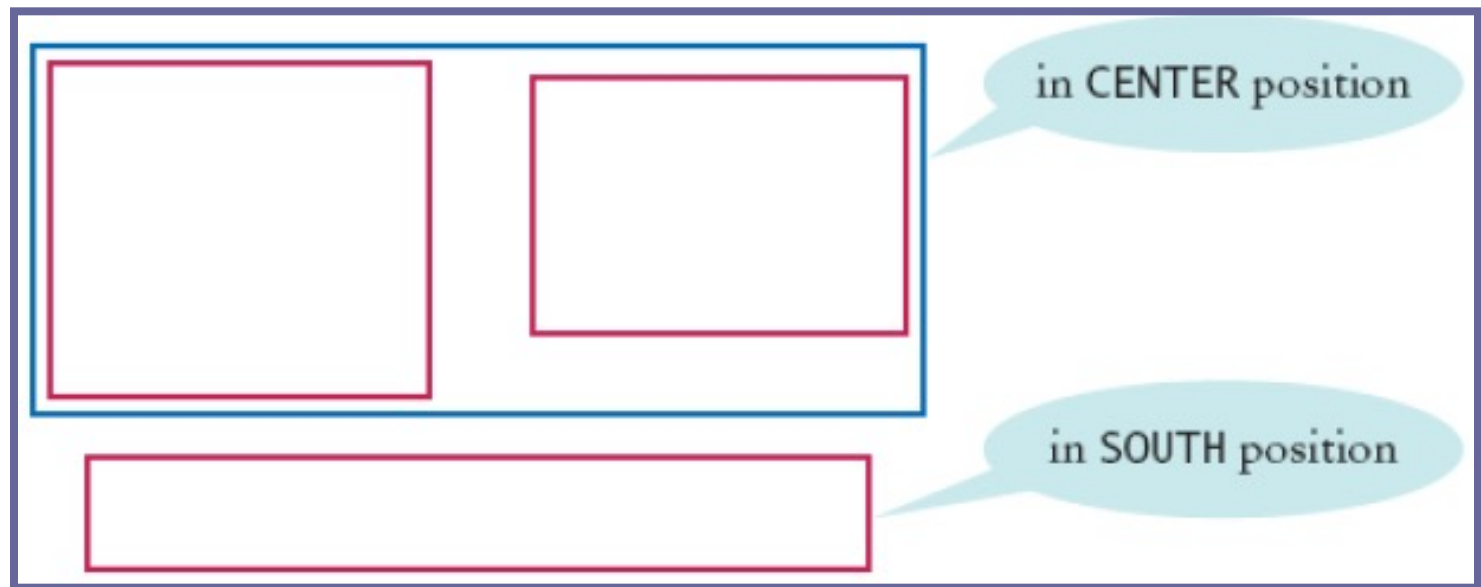
---

- Passo 3: determina un layout per ogni gruppo (i.e., identificare i gestori della disposizione per ciascun gruppo)
  - Quando i componenti sono disposti orizzontalmente, scegliete un gestore a scorrimento
  - Quando i componenti sono disposti verticalmente, usate un gestore a griglia, che abbia tante righe quanti sono i componenti e una sola colonna
- Nell'esempio
  - Un gestore a griglia (3,1) per i pulsanti radio
  - Un gestore a griglia (2,1) per le caselle di testo
  - Un gestore a scorrimento per l'etichetta e il campo di testo

# Osservazioni sulla gestione del layout

---

- Passo 4: raggruppa i gruppi



Avete terminato quando tutti i gruppi si trovano in un unico contenitore

- Passo 5: Scrivi il codice per generare il layout

```
JPanel radioButtonPanel = new JPanel();
radioButtonPanel.setLayout(new GridLayout(3,1));
radioButtonPanel.setBorder(new TitleBorder(new
                                EtcheBorder(),"Size"));
radioButtonPanel.add(smallButton);
radioButtonPanel.add(mediumButton);
radioButtonPanel.add(largeButton);
```

```
JPanel checkBoxPanel = new JPanel();
checkBoxPanel.setLayout(new GridLayout(2,1));
checkBoxPanel.add(pepperoniButton);
checkBoxPanel.add(anchovies);
```

```
JPanel pricePanel = new JPanel(); // usa FlowLayout
pricePanel.add(new JLabel("Your Price:"));
pricePanel.add(priceTextField);
```

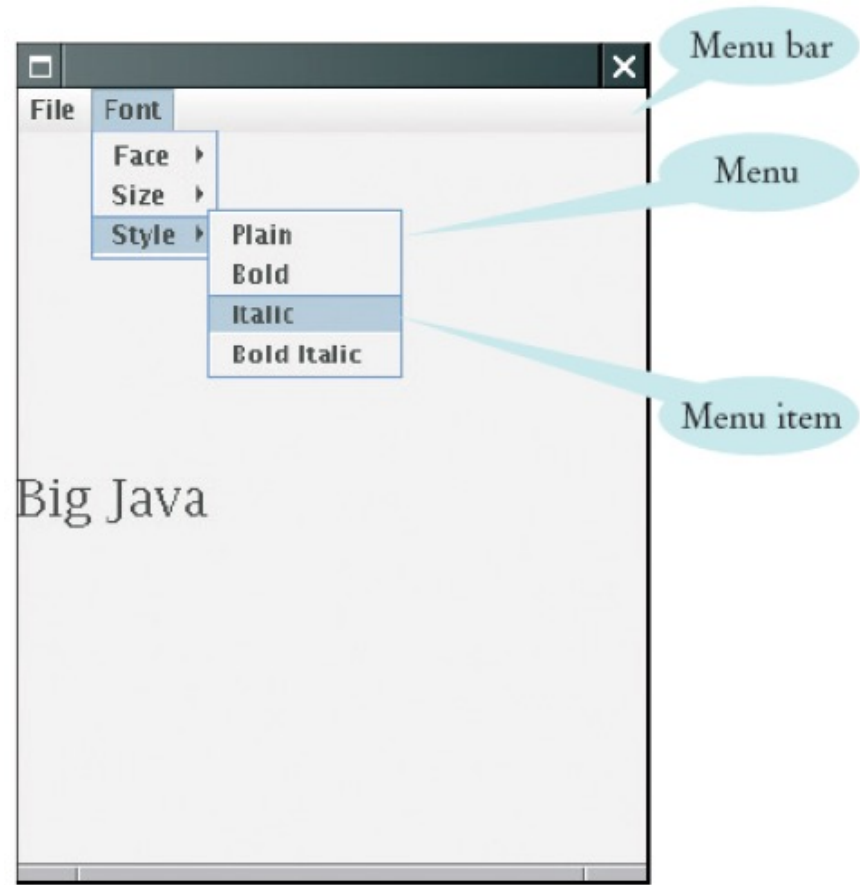
```
JPanel centerPanel = new JPanel(); // usa FlowLayout
centerPanel.add(radioButtonPanel);
centerPanel.add(checkBoxPanel);
```

```
// il frame è gestito in modo predefinito da un BorderLayout
Add(centerPanel, BorderLayout.CENTER)
Add(pricePanel, BorderLayout.SOUTH)
```



# Menu

- Ogni frame contiene una barra dei menu
- La barra contiene dei menu
- Un menu contiene sub-menu e item del menu



# Nuove classi javax.swing utilizzate

---

- **JMenuBar**
  - implementazione della barra dei menu di una finestra
- **JMenuItem**
  - implementazione di un elemento di un menu
  - praticamente un pulsante posizionato in una lista
- **JMenu** (sottoclasse di **JMenuItem**)
  - implementazione di un menu
  - essenzialmente un pulsante con un pop-up menu associato
  - contiene lista di oggetti di tipo **JMenuItem** e **Jseparator**
- **JComponent** è un supertipo per ciascuna di queste classi

# Voci del menu

---

- Gli item e i sub-menu si aggiungono con il metodo add:

```
JMenuItem fileExitItem = new JMenuItem("Exit");  
fileMenu.add(fileExitItem);
```

- Un item non ha ulteriori sub-menu
- Gli item generano eventi del tipo **ActionEvent**
- Si può aggiungere un ascoltatore ad ogni item:

```
fileExitItem.addActionListener(listener);
```

- In genere si aggiungono ascoltatori di azioni solo agli item e non ai menu o alla barra dei menu

# Programma esempio

---

- Costruire un menu tipo
- Catturare gli eventi generati dai menu item
- Per una migliore leggibilità, scrivere un metodo per ogni menu o insieme di menu correlati
  - **createFaceItem**: crea item per cambiare il font
  - **createSizeItem**
  - **createStyleItem**

# File MenuFrameViewer.java

---

```
01: import javax.swing.JFrame;
02:
03: /**
04:     This program tests the MenuFrame.
05: */
06: public class MenuFrameViewer
07: {
08:     public static void main(String[] args)
09:     {
10:         JFrame frame = new MenuFrame();
11:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12:         frame.setVisible(true);
13:     }
14: }
15:
```

# File MenuFrame.java

---

```
001: import java.awt.BorderLayout;
002: import java.awt.Font;
003: import java.awt.GridLayout;
004: import java.awt.event.ActionEvent;
005: import java.awt.event.ActionListener;
006: import javax.swing.ButtonGroup;
007: import javax.swing.JButton;
008: import javax.swing.JCheckBox;
009: import javax.swing.JComboBox;
010: import javax.swing.JFrame;
011: import javax.swing.JLabel;
012: import javax.swing.JMenu;
013: import javax.swing.JMenuBar;
014: import javax.swing.JMenuItem;
015: import javax.swing.JPanel;
016: import javax.swing.JRadioButton;
```

# File MenuFrame.java

---

```
017: import javax.swing.border.EtchedBorder;
018: import javax.swing.border.TitledBorder;
019:
020: /**
021:     This frame has a menu with commands to change the font
022:     of a text sample.
023: */
024: public class MenuFrame extends JFrame
025: {
026:     /**
027:         Constructs the frame.
028:     */
029:     public MenuFrame()
030:     {
031:         // Construct text sample
032:         sampleField = new JLabel("Big Java");
033:         add(sampleField, BorderLayout.CENTER);
034:
```

**Nota:**

Il gestore di layout  
si può usare  
anche  
per JFrame

# File MenuFrame.java

---

```
035:         // Construct menu
036:         JMenuBar menuBar = new JMenuBar();
037:         setJMenuBar(menuBar);
038:         menuBar.add(createFileMenu());
039:         menuBar.add(createFontMenu());
040:
041:         facename = "Serif";
042:         fontsize = 24;
043:         fontstyle = Font.PLAIN;
044:
045:         setSampleFont();
046:         setSize(FRAME_WIDTH, FRAME_HEIGHT);
047:     }
048:
049:     /**
050:      Creates the File menu.
051:      @return the menu
052:     */
```



# File MenuFrame.java

---

```
053:     public JMenu createFileMenu()
054:     {
055:         JMenu menu = new JMenu("File");
056:         menu.add(createFileExitItem());
057:         return menu;
058:     }
059:
060:     /**
061:         Creates the File->Exit menu item and sets its
            // action listener.
062:         @return the menu item
063:     */
064:     public JMenuItem createFileExitItem()
065:     {
066:         JMenuItem item = new JMenuItem("Exit");
067:         class MenuItemListener implements ActionListener
068:         {
069:             public void actionPerformed(ActionEvent event)
```

# File MenuFrame.java

---

```
070:         {
071:             System.exit(0);
072:         }
073:     }
074:     ActionListener listener = new MenuItemListener();
075:     item.addActionListener(listener);
076:     return item;
077: }
078:
079: /**
080:     Creates the Font submenu.
081:     @return the menu
082: */
083: public JMenu createFontMenu()
084: {
085:     JMenu menu = new JMenu("Font");
086:     menu.add(createFaceMenu());
```

# File MenuFrame.java

---

```
087:         menu.add(createSizeMenu());
088:         menu.add(createStyleMenu());
089:         return menu;
090:     }
091:
092:     /**
093:         Creates the Face submenu.
094:         @return the menu
095:     */
096:     public JMenu createFaceMenu()
097:     {
098:         JMenu menu = new JMenu("Face");
099:         menu.add(createFaceItem("Serif"));
100:         menu.add(createFaceItem("SansSerif"));
101:         menu.add(createFaceItem("Monospaced"));
102:         return menu;
103:     }
104:
```

# File MenuFrame.java

---

```
105:    /**
106:        Creates the Size submenu.
107:        @return the menu
108:    */
109:    public JMenu createSizeMenu()
110:    {
111:        JMenu menu = new JMenu("Size");
112:        menu.add(createSizeItem("Smaller", -1));
113:        menu.add(createSizeItem("Larger", 1));
114:        return menu;
115:    }
116:
117:    /**
118:        Creates the Style submenu.
119:        @return the menu
120:    */
121:    public JMenu createStyleMenu()
122:    {
```

# File MenuFrame.java

---

```
123:         JMenu menu = new JMenu("Style");
124:         menu.add(createStyleItem("Plain", Font.PLAIN));
125:         menu.add(createStyleItem("Bold", Font.BOLD));
126:         menu.add(createStyleItem("Italic", Font.ITALIC));
127:         menu.add(createStyleItem("Bold Italic", Font.BOLD
128:             + Font.ITALIC));
129:         return menu;
130:     }
131:
132:
133:     /**
134:         Creates a menu item to change the font face and
            // set its action listener.
135:         @param name the name of the font face
136:         @return the menu item
137:     */
138:     public JMenuItem createFaceItem(final String name)
139:     {
```

# File MenuFrame.java

---

```
140:         JMenuItem item = new JMenuItem(name);
141:         class MenuItemListener implements ActionListener
142:         {
143:             public void actionPerformed(ActionEvent event)
144:             {
145:                 facename = name;
146:                 setSampleFont();
147:             }
148:         }
149:         ActionListener listener = new MenuItemListener();
150:         item.addActionListener(listener);
151:         return item;
152:     }
153:
```

# File MenuFrame.java

---

```
154:    /**
155:        Creates a menu item to change the font size
156:        and set its action listener.
157:        @param name the name of the menu item
158:        @param ds the amount by which to change the size
159:        @return the menu item
160:    */
161:    public JMenuItem createSizeItem(String name, final int ds)
162:    {
163:        JMenuItem item = new JMenuItem(name);
164:        class MenuItemListener implements ActionListener
165:        {
166:            public void actionPerformed(ActionEvent event)
167:            {
168:                fontsize = fontsize + ds;
169:                setSampleFont();
170:            }
171:        }
```

# File MenuFrame.java

---

```
172:         ActionListener listener = new MenuItemListener();
173:         item.addActionListener(listener);
174:         return item;
175:     }
176:
177:     /**
178:         Creates a menu item to change the font style
179:         and set its action listener.
180:         @param name the name of the menu item
181:         @param style the new font style
182:         @return the menu item
183:     */
184:     public JMenuItem createStyleItem(String name,
        final int style)
185:     {
186:         JMenuItem item = new JMenuItem(name);
187:         class MenuItemListener implements ActionListener
188:         {
```



# File MenuFrame.java

---

```
189:         public void actionPerformed(ActionEvent event)
190:         {
191:             fontstyle = style;
192:             setSampleFont();
193:         }
194:     }
195:     ActionListener listener = new MenuItemListener();
196:     item.addActionListener(listener);
197:     return item;
198: }
199:
200: /**
201:     Sets the font of the text sample.
202: */
203: public void setSampleFont()
204: {
```

# File MenuFrame.java

---

```
205:         Font f = new Font(facename, fontstyle, fontsize);
206:         sampleField.setFont(f);
207:         sampleField.repaint();
208:     }
209:
210:     private JLabel sampleField;
211:     private String facename;
212:     private int fontstyle;
213:     private int fontsize;
214:
215:     private static final int FRAME_WIDTH = 300;
216:     private static final int FRAME_HEIGHT = 400;
217: }
218:
219:
```

# Area di testo

---

- Si usa **JTextArea** per mostrare linee di testo multiple
- Si possono specificare numero di righe e colonne:

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

- **setText**: per impostare il testo di un campo o un'area di testo
- **append**: per aggiungere testo alla fine di un'area di testo

# Area di testo

---

- Si usa “\n” carattere per separare righe:

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

- Se si vuole usare un'area di testo solo per visualizzare un testo:

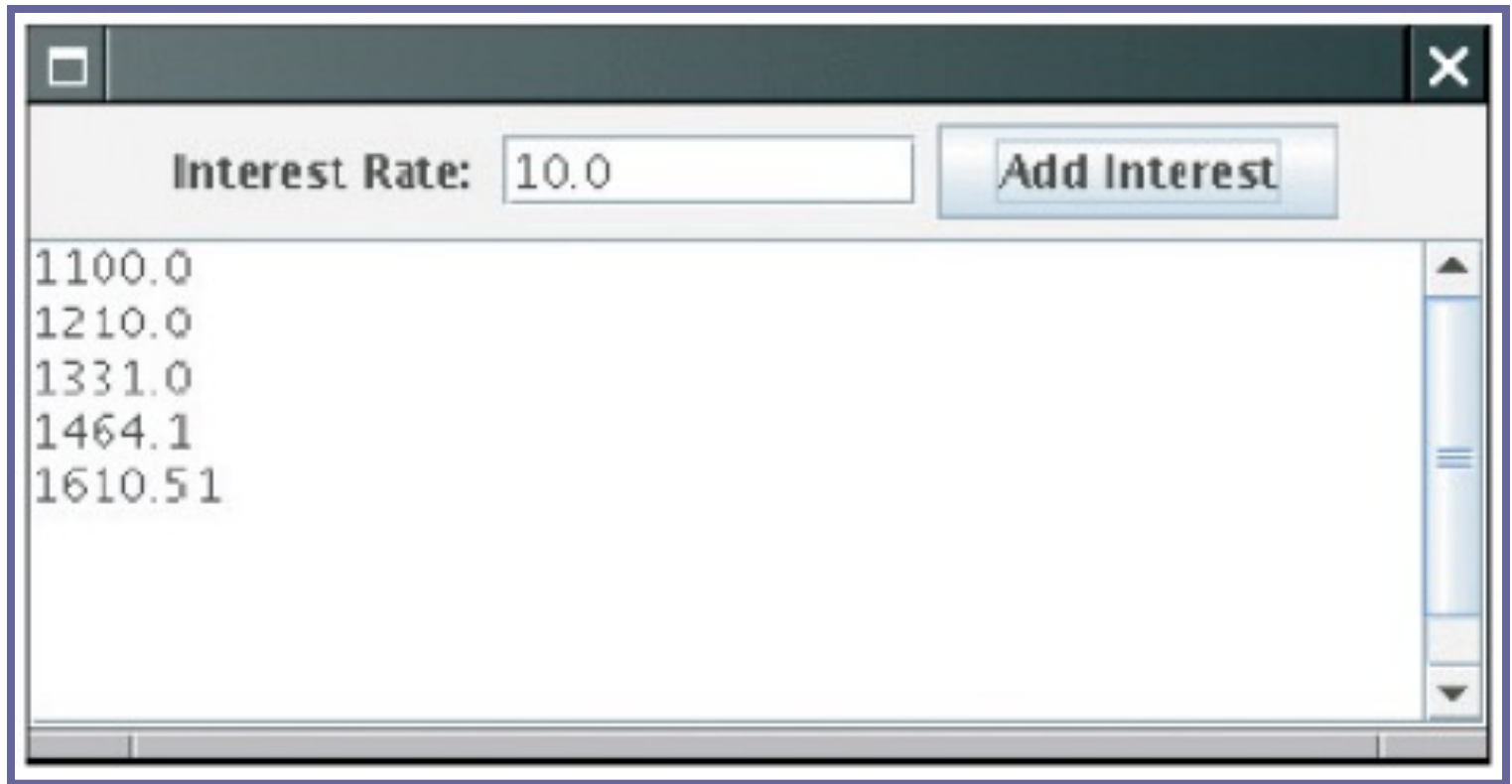
```
textArea.setEditable(false);  
// program can call setText and append to change it
```

- Si possono aggiungere barre di scorrimento:

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

# Area di testo

---



The screenshot shows a Java Swing window with a title bar containing a maximize button and a close button. The window contains a text area with the following text:

1100.0  
1210.0  
1331.0  
1464.1  
1610.51

Below the text area is a button labeled "Add Interest". Above the button is a text field containing the value "10.0", which is preceded by the label "Interest Rate:". The window has a standard Mac OS X-style title bar.

# File TextAreaViewer.java

---

```
01: import java.awt.BorderLayout;
02: import java.awt.event.ActionEvent;
03: import java.awt.event.ActionListener;
04: import javax.swing.JButton;
05: import javax.swing.JFrame;
06: import javax.swing.JLabel;
07: import javax.swing.JPanel;
08: import javax.swing.JScrollPane;
09: import javax.swing.JTextArea;
10: import javax.swing.JTextField;
11:
12: /**
13:     This program shows a frame with a text area that
14:     displays the growth of an investment.
15: */
16: public class TextAreaViewer
17: {
```

# File TextAreaViewer.java

---

```
18:    public static void main(String[] args)
19:    {
20:        JFrame frame = new JFrame();
21:
22:        // The application adds interest to this bank account
23:        final BankAccount account =
            new BankAccount(INITIAL_BALANCE);
24:        // The text area for displaying the results
25:        final int AREA_ROWS = 10;
26:        final int AREA_COLUMNS = 30;
27:
28:        final JTextArea textArea = new JTextArea(
29:            AREA_ROWS, AREA_COLUMNS);
30:        textArea.setEditable(false);
31:        JScrollPane scrollPane = new JScrollPane(textArea);
32:
33:        // The label and text field for entering the
            // interest rate
```

# File TextAreaViewer.java

---

```
34:         JLabel rateLabel = new JLabel("Interest Rate: ");
35:
36:         final int FIELD_WIDTH = 10;
37:         final JTextField rateField =
38:             new JTextField(FIELD_WIDTH);
39:         rateField.setText("" + DEFAULT_RATE);
40:
41:         // The button to trigger the calculation
42:         JButton calculateButton = new JButton("Add Interest");
43:
44:         // The panel that holds the input components
45:         JPanel northPanel = new JPanel();
46:         northPanel.add(rateLabel);
47:         northPanel.add(rateField);
48:         northPanel.add(calculateButton);
49:
50:         frame.add(northPanel, BorderLayout.NORTH);
51:         frame.add(scrollPane);
```



# File TextAreaViewer.java

---

```
52:         class CalculateListener implements ActionListener
53:         {
54:             public void actionPerformed(ActionEvent event)
55:             {
56:                 double rate = Double.parseDouble(
57:                     rateField.getText());
58:                 double interest = account.getBalance()
59:                     * rate / 100;
60:                 account.deposit(interest);
61:                 textArea.append(account.getBalance() + "\n");
62:             }
63:         }
64:
65:         ActionListener listener = new CalculateListener();
66:         calculateButton.addActionListener(listener);
67:
```

# File TextAreaViewer.java

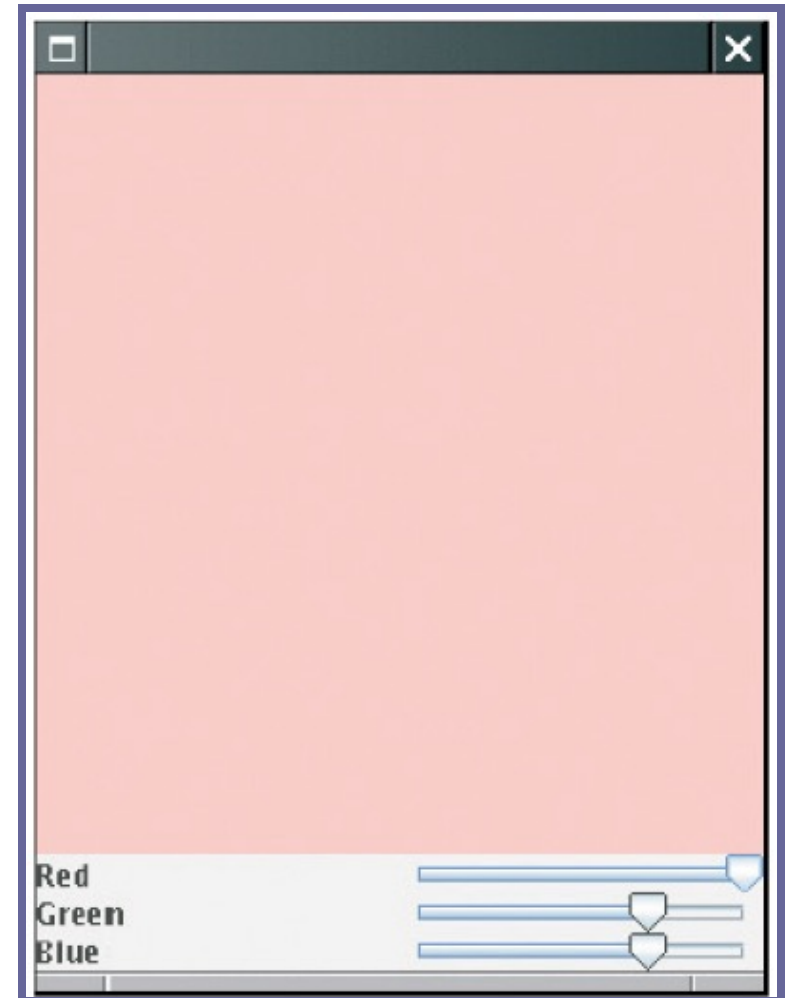
---

```
68:         frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
69:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
70:         frame.setVisible(true);
71:     }
72:
73:     private static final double DEFAULT_RATE = 10;
74:     private static final double INITIAL_BALANCE = 1000;
75:
76:     private static final int FRAME_WIDTH = 400;
77:     private static final int FRAME_HEIGHT = 200;
78: }
```

# Consultare la documentazione di Swing

---

- Per ottenere effetti grafici particolari si può consultare la documentazione del pacchetto Swing
- Consideriamo il problema di realizzare un mixer dei colori Rosso, Verde, Blu per visualizzare ogni possibile colore



# Esempio

---

- Come facciamo a stabilire se c'è un cursore a scorrimento (slider) in Swing?
  - Consulta la documentazione e controlla tutti i nomi che cominciano per J
    - **JSlider** può essere un buon candidato
- Domande successive:
  - Come costruisco un **JSlider**?
  - Come ricevo la notifica quando l'utente lo muove?
  - Come posso determinare i valori corrispondenti alla posizione del cursore?
- Dopo aver trovato la risposta a queste domande possiamo migliorare altri aspetti grafici del mixer.



# Esempio: osservazioni

---

- Ci sono più di 50 metodi descritti direttamente nella documentazione di **JSlider** e oltre 250 metodi ereditati
- Alcune descrizioni non sono di facile comprensione
- Bisogna sviluppare l'attitudine a sorvolare dettagli meno significativi

# Come si istanzia uno JSlider?

---

- Nella documentazione di Java 5.0 ci sono sei costruttori per **JSlider**
- Scegli il più appropriato
  - `public JSlider()`  
Crea uno slider orizzontale con intervallo da 0 a 100 e valore iniziale 50
  - `public JSlider(BoundedRangeModel brm)`  
Crea uno slider orizzontale usando lo specificato **BoundedRangeModel**
  - `public JSlider(int min, int max, int value)`  
Crea uno slider orizzontale usando min, max e value specificati

# Come riceviamo la notifica che uno `JSlider` è stato mosso?

---

- Non c'è un metodo `addActionListener`
- C'è un metodo

```
public void addChangeListener(ChangeListener l)
```

- Segui il link su `ChangeListener`
  - ha un solo metodo:

```
void stateChanged(ChangeEvent e)
```

# Come riceviamo la notifica che uno **JSlider** è stato mosso?

---

- Sembra che il metodo venga invocato ogni volta che l'utente muove lo slider
- Cos'è un **ChangeEvent**?
  - Eredita **getSource** da **EventObject**
  - **getSource**: ci dice quale componente ha generato l'evento



# Come possiamo stabilire il valore impostato dall'utente con gli `JSlider`?

---

- Aggiungi un **`ChangeListener`** a ogni slider
- Quando uno slider cambia stato, si chiama il metodo **`stateChanged`**
- Determina il nuovo valore dello slider
- Ricalcola il valore del colore
- Ridisegna il pannello del colore

# Come possiamo stabilire il valore impostato dall'utente con gli `JSlider`?

---

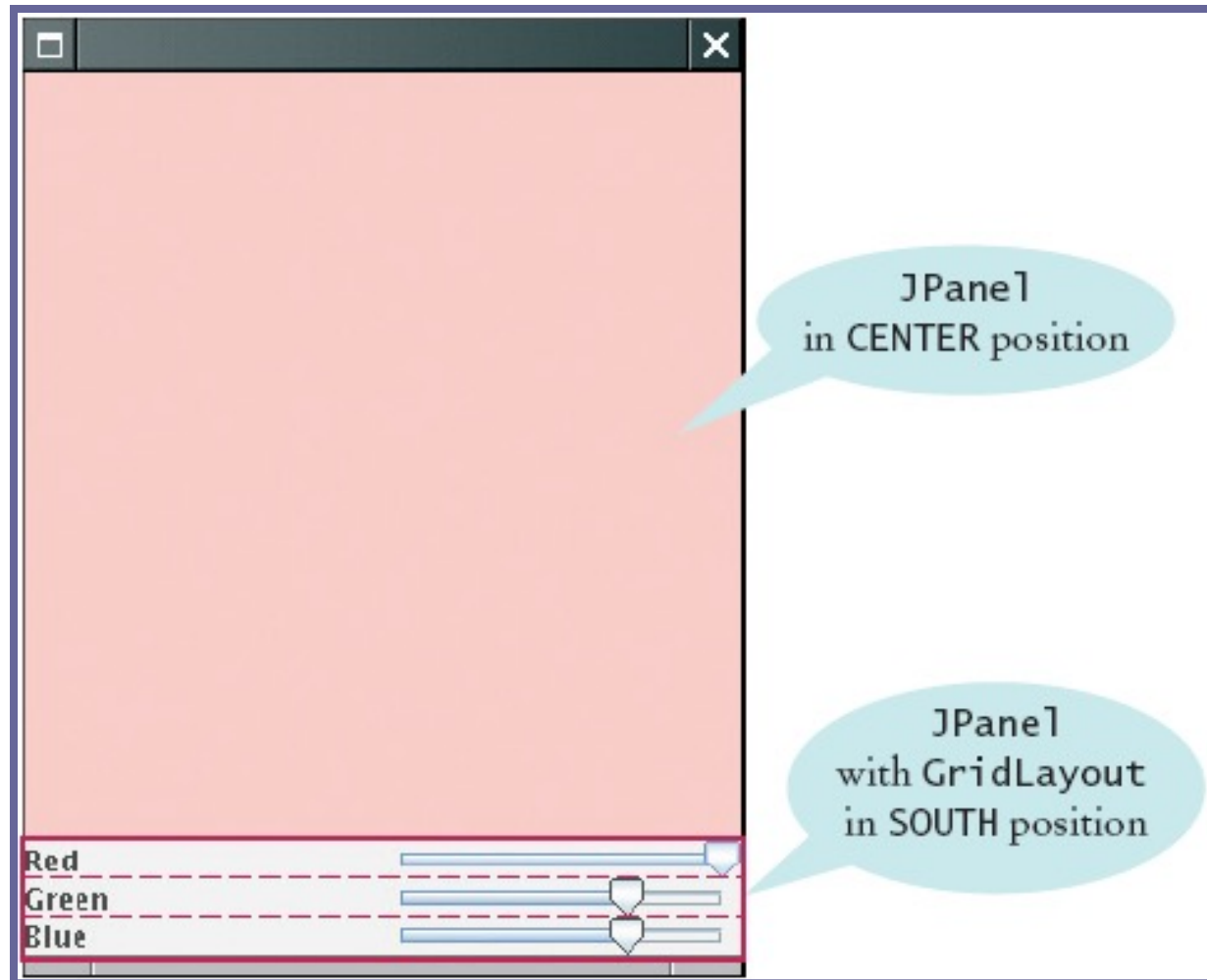
- Dobbiamo recuperare il valore corrente dallo slider
- Controlla tra i metodi che cominciano con `get`:

```
public int getValue()
```

- Restituisce il valore dello slider.

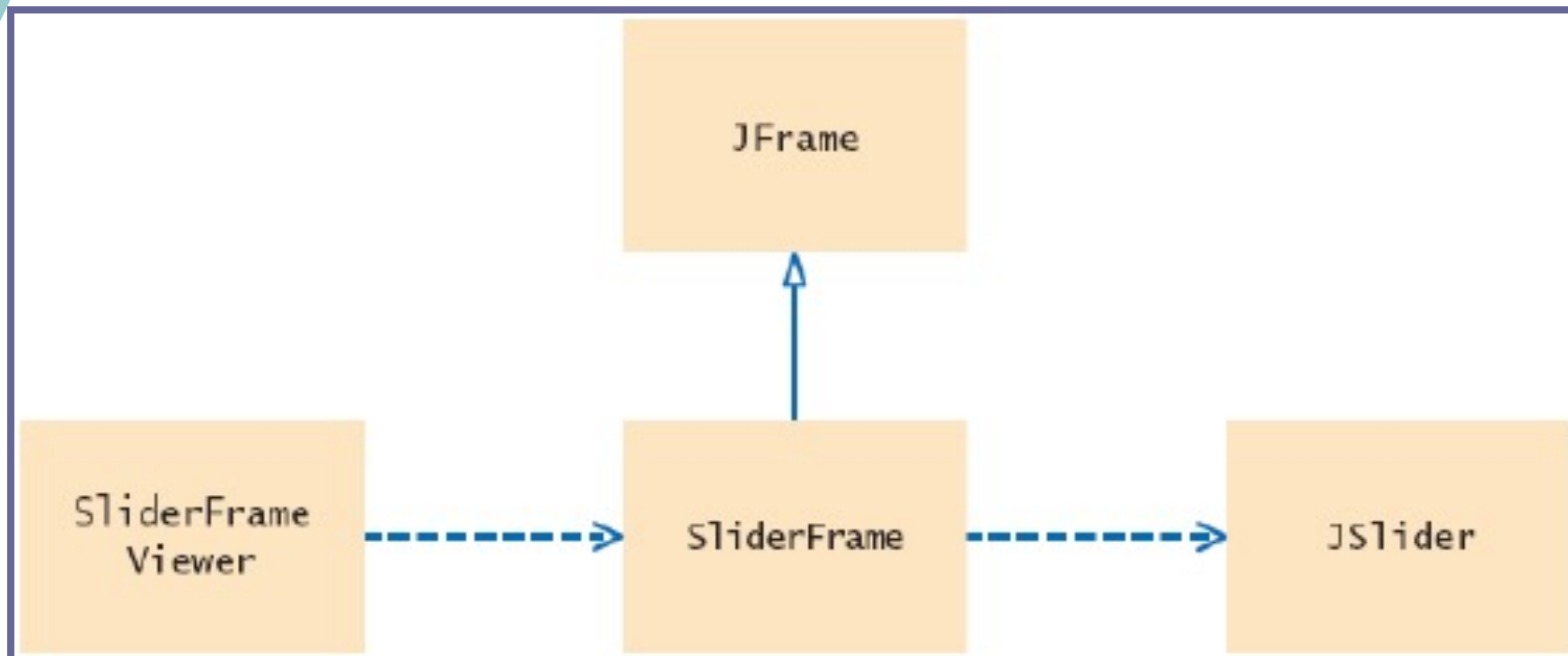
# I componenti dello `SliderFrame`

---



# Classi del programma

---



# File SliderFrameViewer.java

---

```
01: import javax.swing.JFrame;
02:
03: public class SliderFrameViewer
04: {
05:     public static void main(String[] args)
06:     {
07:         SliderFrame frame = new SliderFrame();
08:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
09:         frame.setVisible(true);
10:     }
11: }
12:
```

# File SliderFrame.java

---

```
01: import java.awt.BorderLayout;
02: import java.awt.Color;
03: import java.awt.GridLayout;
04: import javax.swing.JFrame;
05: import javax.swing.JLabel;
06: import javax.swing.JPanel;
07: import javax.swing.JSlider;
08: import javax.swing.event.ChangeListener;
09: import javax.swing.event.ChangeEvent;
10:
11: public class SliderFrame extends JFrame
12: {
13:     public SliderFrame()
14:     {
15:         colorPanel = new JPanel();
16:
```

# File SliderFrame.java

---

```
17:         add(colorPanel, BorderLayout.CENTER);
18:         createControlPanel();
19:         setSampleColor();
20:         setSize(FRAME_WIDTH, FRAME_HEIGHT);
21:     }
22:
23:     public void createControlPanel()
24:     {
25:         class ColorListener implements ChangeListener
26:         {
27:             public void stateChanged(ChangeEvent event)
28:             {
29:                 setSampleColor();
30:             }
31:         }
32:
```

# File SliderFrame.java

---

```
33:         ChangeListener listener = new ColorListener();
34:
35:         redSlider = new JSlider(0, 100, 100);
36:         redSlider.addChangeListener(listener);
37:
38:         greenSlider = new JSlider(0, 100, 70);
39:         greenSlider.addChangeListener(listener);
40:
41:         blueSlider = new JSlider(0, 100, 70);
42:         blueSlider.addChangeListener(listener);
43:
44:         JPanel controlPanel = new JPanel();
45:         controlPanel.setLayout(new GridLayout(3, 2));
46:
47:         controlPanel.add(new JLabel("Red"));
48:         controlPanel.add(redSlider);
49:
```



## File SliderFrame.java

---

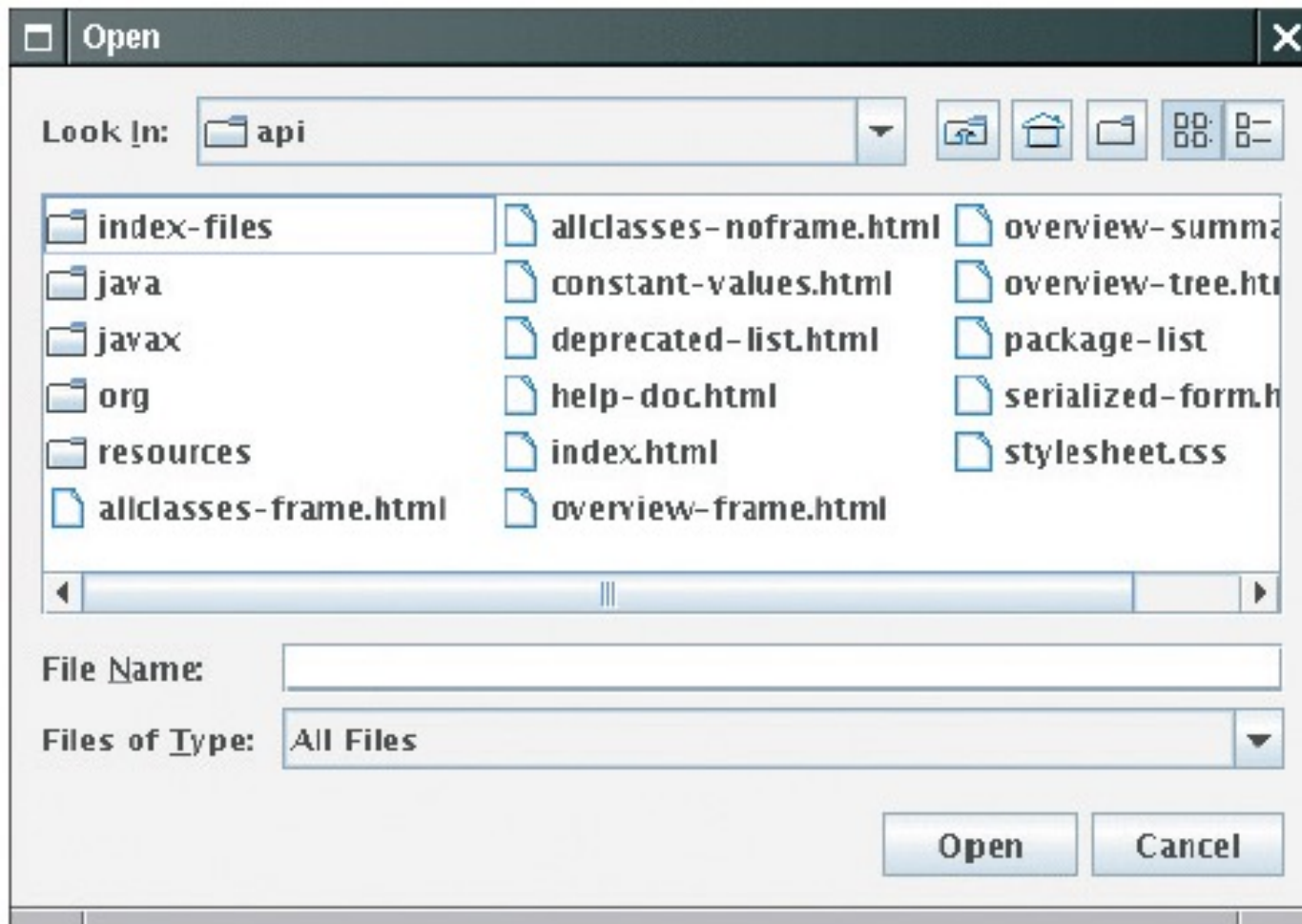
```
50:         controlPanel.add(new JLabel("Green"));
51:         controlPanel.add(greenSlider);
52:
53:         controlPanel.add(new JLabel("Blue"));
54:         controlPanel.add(blueSlider);
55:
56:         add(controlPanel, BorderLayout.SOUTH);
57:     }
58:
59:     /**
60:         Reads the slider values and sets the panel to
61:         the selected color.
62:     */
63:     public void setSampleColor()
64:     {
65:         // Read slider values
66:
```

# File SliderFrame.java

---

```
67:         float red = 0.01F * redSlider.getValue();
68:         float green = 0.01F * greenSlider.getValue();
69:         float blue = 0.01F * blueSlider.getValue();
70:
71:         // Set panel background to selected color
72:
73:         colorPanel.setBackground(new Color(red, green, blue));
74:         colorPanel.repaint();
75:     }
76:
77:     private JPanel colorPanel;
78:     private JSlider redSlider;
79:     private JSlider greenSlider;
80:     private JSlider blueSlider;
81:
82:     private static final int FRAME_WIDTH = 300;
83:     private static final int FRAME_HEIGHT = 400;
84: }
```

# File Dialog Boxes



A JFileChooser Dialog Box

# File Dialog Boxes

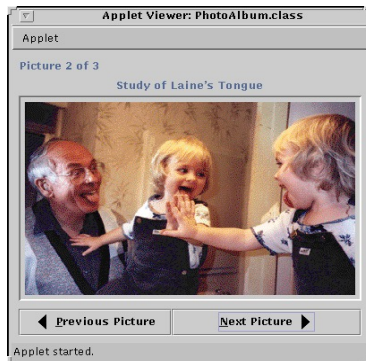
---

```
JFileChooser chooser = new JFileChooser();
FileReader in = null;
if (chooser.showOpenDialog(null) ==
    JFileChooser.APPROVE_OPTION)
{
    File selectedFile = chooser.getSelectedFile();
    reader = new FileReader(selectedFile);
    . . .
}
```

# Container

- Top-level container

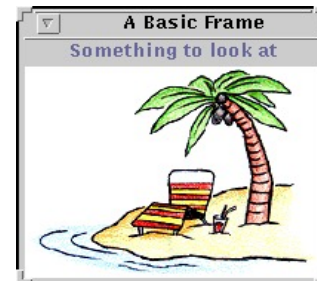
**JApplet**



**JDialog**



**JFrame**

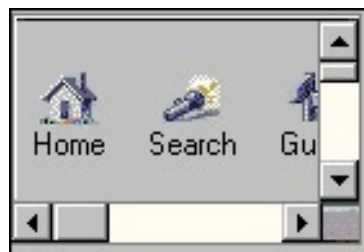


- General-purpose container

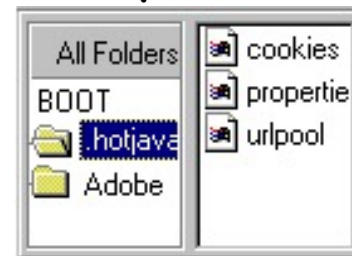
**JPanel**



**JScrollPane**



**JSplitPane**



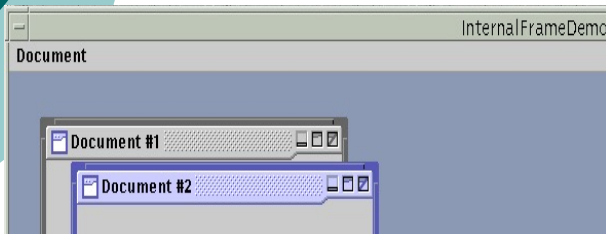
**JTabbedPane**



# Container

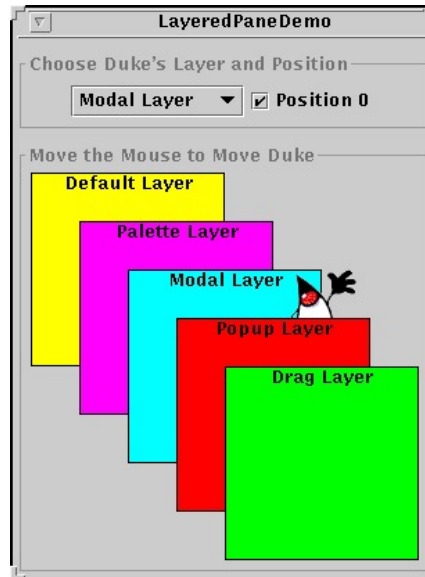
- Special-purpose container

## JInternalFrame



Permette di inserire frame dentro altri frame

## JLayeredPane



Permette di inserire componenti a vari livelli di profondità

## JToolBar



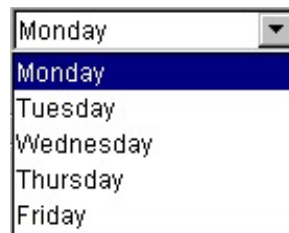
Permette di semplificare l'attivazione di determinate funzioni per mezzo di semplici pulsanti

# Controlli di base

**JButtons**



**JComboBox**



**JList**

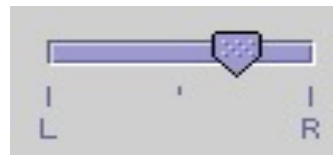


**JMenu**

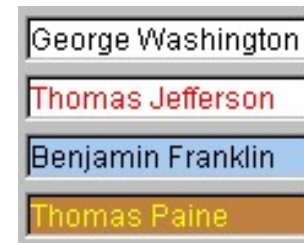


Include buttons,  
radioButtons, checkbox,  
MenuItem, ToggleButton

**JSlider**



**TextField**



Include  
JPasswordField,  
JTextArea

# Visualizzatori di informazioni non editabili

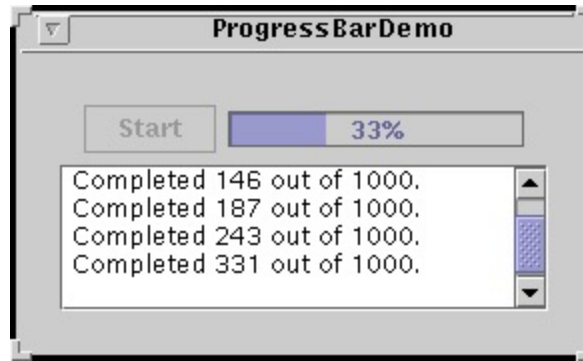
---

## JLabel



**Può includere  
immagini e/o testo**

## JProgressBar



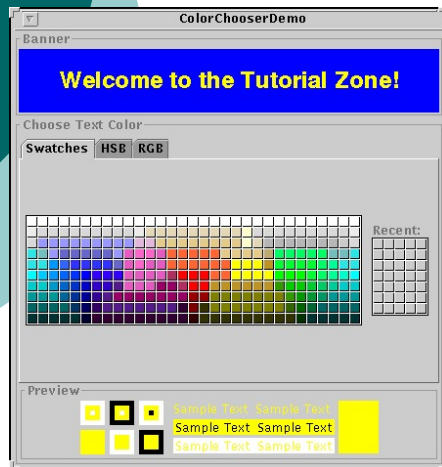
## Jcomponent.setToolTipText(String)





# Visualizzatori di informazioni formattate editabili

## JColorChooser



## JFileChooser



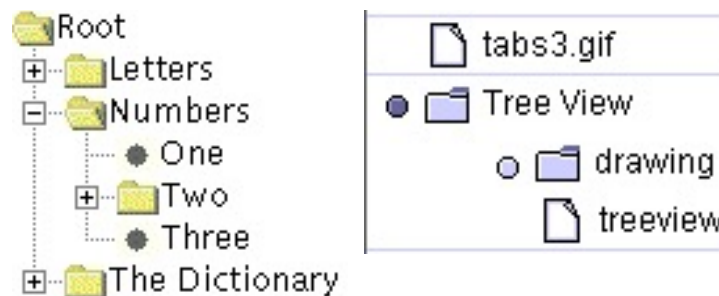
## JTable

TableDemo				
First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	<input type="checkbox"/>
Alison	Huml	Rowing	3	<input checked="" type="checkbox"/>
Kathy	Walrath	Chasing toddl...	2	<input type="checkbox"/>
Mark	Andrews	Speed reading	20	<input checked="" type="checkbox"/>

## JTextFieldComponent

Verify that the RJ45 cable is connected to the WAN plug on the back of the Pipeline unit.

## JTree



**JTextField,**  
**JPasswordField,**  
**JTextArea, JEditorPane,**  
**JTextPane**