

# Planejamento

- Aula 1
  - Introdução
  - Conceitos Básicos
    - Classe, Objeto, Método, Herança, polimorfismo, Encapsulamento
  - Introdução a linguagem Java
- Aula 2
  - Introdução a Ambientes Integrados de Desenvolvimento
  - Introdução ao Ambiente Eclipse
  - Desenvolvimento de Programas básicos (modo texto)
    - Comandos de controle: if, while, forClasses: String, Integer, DateBibliotecas Básicas: System, java,
- Aula 3
  - Programando Interfaces Gráficas com Java – Parte I
- **Aula 4**
  - **Programando Interfaces Gráficas com Java – Parte II**
- Aula 5
  - 5.1. Introdução a Design Patterns
  - 5.2. Programação concorrente (Threads)

# Aula 4 – Programando Interfaces Gráficas Orientadas a Objeto - Parte 2

## Sumário

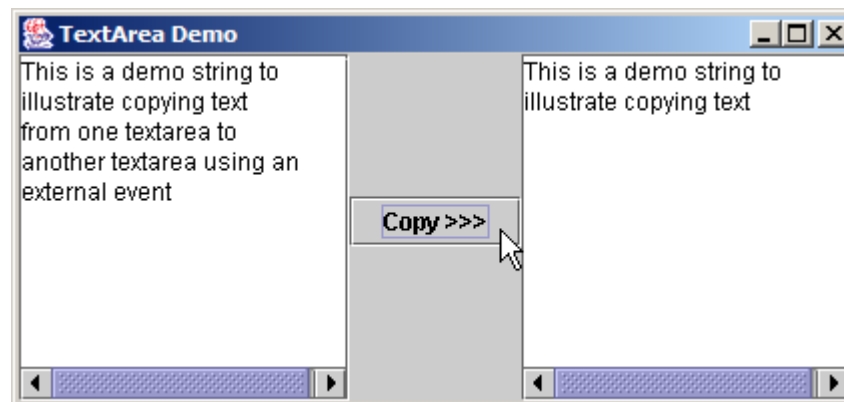
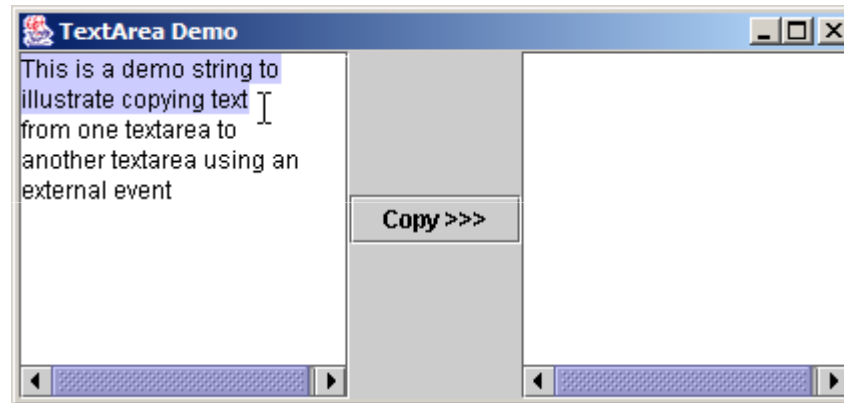
- 4.1 **Introdução**
- 4.2 **Componente JTextArea**
- 4.3 **Criando uma classe customizada de JPanel**
- 4.4 **Uma subclasse JPanel que trata seus próprios eventos**
- 4.5 **JSlider**
- 4.6 **Windows: Additional Notes**
- 4.7 **Usando Menus com Frames**
- 4.8 **JPopupMenu**
- 4.9 **Pluggable Look-and-Feel**
- 4.10 **JDesktopPane and JInternalFrame**
- 4.11 **JTabbedPane**
- 4.12 **Layout Managers: BorderLayout and GridBagLayout**

## 4.1 Introdução

- Componentes GUI Avançados
  - Text areas
  - Sliders
  - Menus
- Multiple Document Interface (MDI)
- Layout Managers Avançados
  - BorderLayout
  - GridBagLayout

## 4.2 JTextArea

- JTextArea
  - Area for manipulating multiple lines of text
  - extends JTextComponent



```

1 // Fig. 4.1: TextAreaDemo.java
2 // Copying selected text from one textarea to another.
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6
7 public class TextAreaDemo extends JFrame {
8     private JTextArea textArea1, textArea2;
9     private JButton copyButton;
10
11 // set up GUI
12 public TextAreaDemo()
13 {
14     super( "TextArea Demo" );
15
16     Box box = Box.createHorizontalBox();
17
18     String string = "This is a demo string to\n" +
19         "illustrate copying text\nfrom one textarea to \n" +
20         "another textarea using an\nexternal event\n";
21
22 // set up textArea1
23 textArea1 = new JTextArea( string, 10, 15 );
24 box.add( new JScrollPane( textArea1 ) );
25

```

Create BOX container for organizing GUI components

Populate JTextArea with String, then add to BOX

```

26 // set up copyButton
27 copyButton = new JButton( "Copy >>>" );
28 box.add( copyButton );
29 copyButton.addActionListener(
30
31     new ActionListener() { // anonymous inner class
32
33         // set text in textArea2 to selected text from textArea1
34         public void actionPerformed( ActionEvent event )
35         {
36             textArea2.setText( textArea1.getSelectedText() );
37         }
38     } // end anonymous inner class
39 ); // end call to addActionListener
40
41 // set up textArea2
42 textArea2 = new JTextArea( 10, 15 );
43 textArea2.setEditable( false );
44 box.add( new JScrollPane( textArea2 ) );
45
46 // add box to content pane
47 Container container = getContentPane();
48 container.add( box ); // place in BorderLayout.CENTER
49
50
51

```

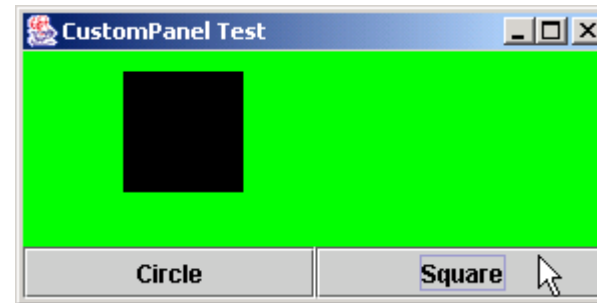
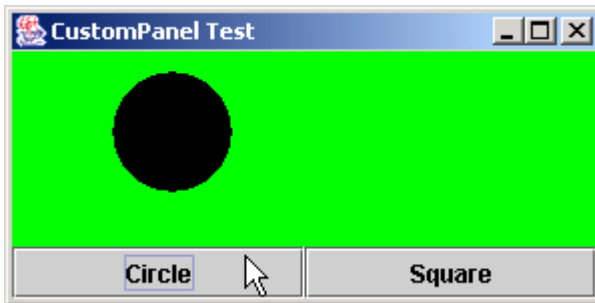
When user presses JButton, textArea1's highlighted text is copied into textArea2

Instantiate uneditable JTextArea

```
52     setSize( 425, 200 );
53     setVisible( true );
54
55 } // end constructor TextAreaDemo
56
57 public static void main( String args[] )
58 {
59     TextAreaDemo application = new TextAreaDemo();
60     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
61 }
62
63 } // end class TextAreaDemo
```

## 4.3 Criando uma classe customizada de JPanel

- Extend `JPanel` to create new components
  - Dedicated drawing area
    - Method `paintComponent` of class `JComponent`



```

1 // Fig. 4.2: CustomPanel.java
2 // A customized JPanel class.
3 import java.awt.*;
4 import javax.swing.*;
5
6 public class CustomPanel extends JPanel {
7     public final static int CIRCLE = 1, SQUARE = 2;
8     private int shape;
9
10    // use shape to draw an oval or rectangle
11    public void paintComponent( Graphics g )
12    {
13        super.paintComponent( g );
14
15        if ( shape == CIRCLE )
16            g.fillOval( 50, 10, 60, 60 );
17        else if ( shape == SQUARE )
18            g.fillRect( 50, 10, 60, 60 );
19    }
20
21    // set shape value and repaint CustomPanel
22    public void draw( int shapeToDraw )
23    {
24        shape = shapeToDraw;
25        repaint();
26 }
27
28 } // end class CustomPanel

```

Store integer representing shape to draw

Override method paintComponent of class JComponent to draw oval or rectangle

Method repaint calls method paintComponent

```

1 // Fig. 4.3: CustomPanelTest.java
2 // Using a customized Panel object.
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6
7 public class CustomPanelTest extends JFrame {
8     private JPanel buttonPanel;
9     private CustomPanel myPanel;
10    private JButton circleButton, squareButton;
11
12    // set up GUI
13    public CustomPanelTest()
14    {
15        super( "CustomPanel Test" );
16
17        // create custom drawing area
18        myPanel = new CustomPanel();
19        myPanel.setBackground( Color.GREEN );
20
21        // set up squareButton
22        squareButton = new JButton( "Square" );
23        squareButton.addActionListener(
24

```

Instantiate CustomPanel object and set background to green

```
25     new ActionListener() { // anonymous inner class
26
27         // draw a square
28         public void actionPerformed( ActionEvent event )
29         {
30             myPanel.draw( CustomPanel.SQUARE );
31         }
32
33     } // end anonymous inner class
34
35 ); // end call to addActionListener
36
37 circleButton = new JButton( "Circle" );
38 circleButton.addActionListener(
39
40     new ActionListener() { // anonymous inner class
41
42         // draw a circle
43         public void actionPerformed( ActionEvent event )
44         {
45             myPanel.draw( CustomPanel.CIRCLE );
46         }
47
48     } // end anonymous inner class
49
50 ); // end call to addActionListener
51
```

When user presses squareButton,  
draw square on CustomPanel

When user presses circleButton,  
draw circle on CustomPanel

```
52 // set up panel containing buttons
53 buttonPanel = new JPanel();
54 buttonPanel.setLayout( new GridLayout( 1, 2 ) );
55 buttonPanel.add( circleButton );
56 buttonPanel.add( squareButton );
57
58 // attach button panel & custom drawing area to content pane
59 Container container = getContentPane();
60 container.add( myPanel, BorderLayout.CENTER );
61 container.add( buttonPanel, BorderLayout.SOUTH );
62
63 setSize( 300, 150 );
64 setVisible( true );
65
66 } // end constructor CustomPanelTest
67
68 public static void main( String args[] )
69 {
70     CustomPanelTest application = new CustomPanelTest();
71     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
72 }
73
74 } // end class CustomPanelTest
```

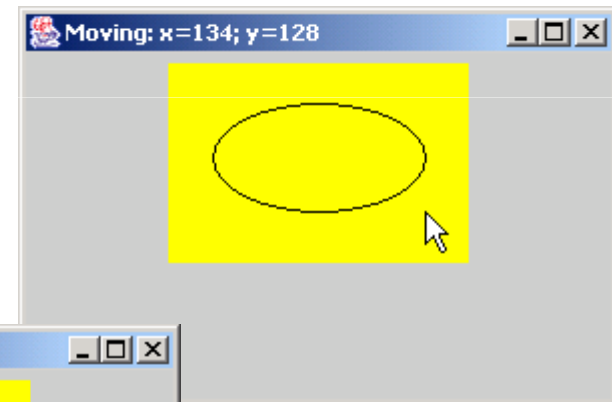
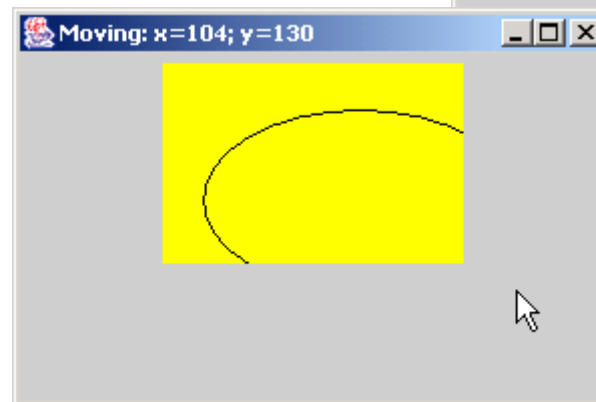
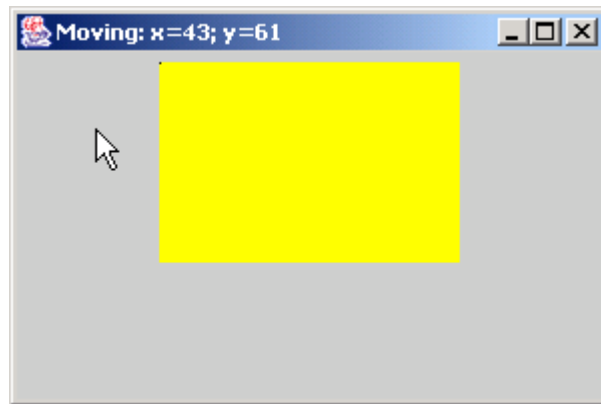
Use GridLayout to organize buttons

## Exercício

- Pergunta: Porquê inicialmente não surge nenhuma forma (círculo ou quadrado) dentro do painel?
- Fazer com que o painel seja apresentado inicialmente com o círculo no seu interior
- Fazer inicialização de cor de fundo do painel na classe do próprio painel.
- Lembrar do conceito de encapsulamento, comportamento específico de uma classe deve estar definido na própria classe e não externamente

## 4.4 Uma subclasse JPanel que trata seus próprios eventos

- JPanel1
  - Does not support conventional events
    - e.g., events offered by buttons, text areas, etc.
  - Capable of recognizing lower-level events
    - e.g., mouse events, key events, etc.
  - Self-contained panel
    - Listens for its own mouse events



```

1 // Fig. 4.4: SelfContainedPanel.java
2 // A self-contained JPanel class that handles its own mouse events.
3
4
5 import java.awt.*;
6 import java.awt.event.*;
7 import javax.swing.*;
8
9 public class SelfContainedPanel extends JPanel {
10     private int x1, y1, x2, y2;
11
12     // set up mouse event handling for SelfContainedPanel
13     public SelfContainedPanel()
14     {
15         // set up mouse listener
16         addMouseListener(
17
18             new MouseAdapter() { // anonymous inner class
19
20                 // handle mouse press event
21                 public void mousePressed( MouseEvent event )
22                 {
23                     x1 = event.getX();
24                     y1 = event.getY();
25                 }
26

```

Self-contained JPanel  
listens for MouseEvents

Save coordinates where user  
pressed mouse button

```
27     // handle mouse release event
28     public void mouseReleased( MouseEvent event )
29     {
30         x2 = event.getX();
31         y2 = event.getY();
32         repaint();
33     }
34
35 } // end anonymous inner class
36
37 ); // end call to addMouseListener
38
39 // set up mouse motion listener
40 addMouseMotionListener(
41
42     new MouseMotionAdapter() { // anonymous inner class
43
44         // handle mouse drag event
45         public void mouseDragged( MouseEvent event )
46         {
47             x2 = event.getX();
48             y2 = event.getY();
49             repaint();
50         }
51
```

Save coordinates where user released mouse button, then repaint

Self-contained JPanel listens for when mouse moves

Save coordinates where user dragged mouse, then repaint

```
52     } // end anonymous inner class
53
54     ); // end call to addMouseListener
55
56 } // end constructor SelfContainedPanel
57
58 // return preferred width and height of SelfContainedPanel
59 public Dimension getPreferredSize()
60 {
61     return new Dimension( 150, 100 );
62 }
63
64 // paint an oval at the specified coordinates
65 public void paintComponent( Graphics g )
66 {
67     super.paintComponent( g );
68
69     g.drawOval( Math.min( x1, x2 ), Math.min( y1, y2 ),
70               Math.abs( x1 - x2 ), Math.abs( y1 - y2 ) );
71 }
72
73 } // end class SelfContainedPanel
```

Draw oval

```
1 // Fig. 4.5: SelfContainedPanelTest.java
2 // Creating a self-contained subclass of JPanel that processes
3 // its own mouse events.
4 import java.awt.*;
5 import java.awt.event.*;
6 import javax.swing.*;
7
8
9
10 public class SelfContainedPanelTest extends JFrame {
11     private SelfContainedPanel myPanel;
12
13     // set up GUI and mouse motion event handlers for application window
14     public SelfContainedPanelTest()
15     {
16         // set up a SelfContainedPanel
17         myPanel = new SelfContainedPanel();
18         myPanel.setBackground( Color.YELLOW );
19
20         Container container = getContentPane();
21         container.setLayout( new FlowLayout() );
22         container.add( myPanel );
23
```

Instantiate SelfContainedPanel  
object and set background to yellow

```
24 // set up mouse motion event handling
25 addMouseListener(
26
27     new MouseMotionListener() { // anonymous inner class
28
29         // handle mouse drag event
30         public void mouseDragged( MouseEvent event )
31         {
32             setTitle( "Dragging: x=" + event.getX() +
33                     "; y=" + event.getY() );
34         }
35
36         // handle mouse move event
37         public void mouseMoved( MouseEvent event )
38         {
39             setTitle( "Moving: x=" + event.getX() +
40                     "; y=" + event.getY() );
41         }
42
43     } // end anonymous inner class
44
45 ); // end call to addMouseListener
46
47 setSize( 300, 200 );
48 setVisible( true );
49
50 } // end constructor selfContainedPanelTest
```

Register anonymous-inner-class object  
to handle mouse motion events

Display String in title bar  
indicating x-y coordinate where  
mouse-motion event occurred

```
51
52     public static void main( String args[] )
53     {
54         SelfContainedPanelTest application = new SelfContainedPanelTest();
55         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
56     }
57
58 } // end class SelfContainedPanelTest
```

## Perguntas

- Porque a barra de status não apresenta a informação de posição do mouse quando, este encontra-se sobre a parte amarela
- Como seria possível criar várias elipses ao invés de apenas uma ?

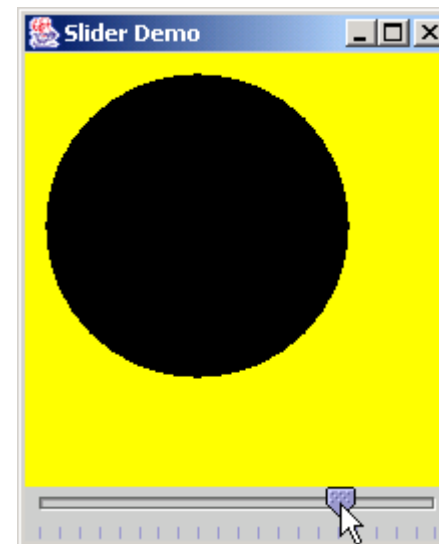
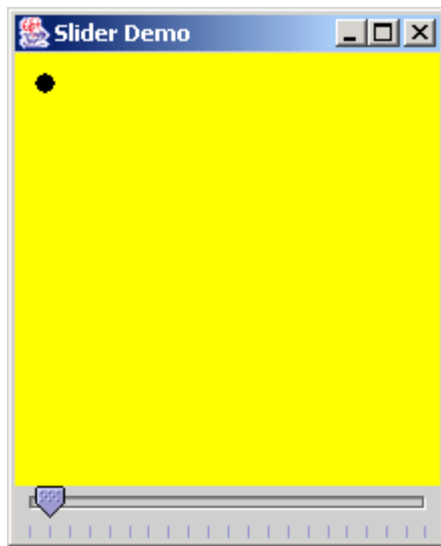
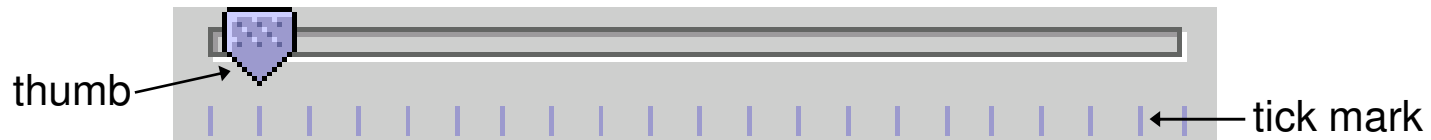
## Exercício

- Criar ao invés de uma elipse um círculo, caso esteja sendo pressionado a tecla Shift durante o “dragging” do mouse

## 4.5 JSlider

- JSlider
  - Enable users to select from range of integer values
  - Several features
    - Tick marks (major and minor)
    - Snap-to ticks
    - Orientation (horizontal and vertical)

# Fig. 4.6 JSlider component with horizontal orientation



```
1 // Fig. 4.7: OvalPanel.java
2 // A customized JPanel class.
3 import java.awt.*;
4 import javax.swing.*;
5
6 public class OvalPanel extends JPanel {
7     private int diameter = 10;
8
9     // draw an oval of the specified diameter
10    public void paintComponent( Graphics g )
11    {
12        super.paintComponent( g );
13
14        g.fillOval( 10, 10, diameter, diameter );
15    }
16
17    // validate and set diameter, then repaint
18    public void setDiameter( int newDiameter )
19    {
20        // if diameter invalid, default to 10
21        diameter = ( newDiameter >= 0 ? newDiameter : 10 );
22        repaint();
23    }
24
```

Draw filled oval of diameter

Set diameter, then repaint


```
25 // used by layout manager to determine preferred size
26 public Dimension getPreferredSize()
27 {
28     return new Dimension( 200, 200 );
29 }
30
31 // used by layout manager to determine minimum size
32 public Dimension getMinimumSize()
33 {
34     return getPreferredSize();
35 }
36
37 } // end class OvalPanel
```

```
1 // Fig. 4.8: SliderDemo.java
2 // Using JSliders to size an oval.
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6 import javax.swing.event.*;
7
8 public class SliderDemo extends JFrame {
9     private JSlider diameterSlider;
10    private OvalPanel myPanel;
11
12    // set up GUI
13    public SliderDemo()
14    {
15        super( "Slider Demo" );
16
17        // set up OvalPanel
18        myPanel = new OvalPanel();
19        myPanel.setBackground( Color.YELLOW );
20
21        // set up JSlider to control diameter value
22        diameterSlider =
23            new JSlider( SwingConstants.HORIZONTAL, 0, 200, 10 );
24        diameterSlider.setMajorTickSpacing( 10 );
25        diameterSlider.setPaintTicks( true );
26
```

Instantiate OvalPanel object  
and set background to yellow



Instantiate horizontal JSlider object  
with min. value of 0, max. value of 200  
and initial thumb location at 10







```
27 // register JSlider event listener
28 diameterslider.addChangeListener(
29
30     new ChangeListener() { // anonymous inner class
31
32         // handle change in slider value
33         public void stateChanged( ChangeEvent e )
34         {
35             myPanel.setDiameter( diameterslider.getValue() );
36         }
37
38     } // end anonymous inner class
39
40 ); // end call to addChangeListener
41
42 // attach components to content pane
43 Container container = getContentPane();
44 container.add( diameterslider, BorderLayout.SOUTH );
45 container.add( myPanel, BorderLayout.CENTER );
46
47 setSize( 220, 270 );
48 setVisible( true );
49
50 } // end constructor sliderDemo
51
```

Register anonymous  
ChangeListener object  
to handle JSlider events

When user accesses JSlider,  
set OvalPanel's diameter  
according to JSlider value

```
52     public static void main( String args[] )
53     {
54         SliderDemo application = new SliderDemo();
55         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
56     }
57
58 } // end class SliderDemo
```

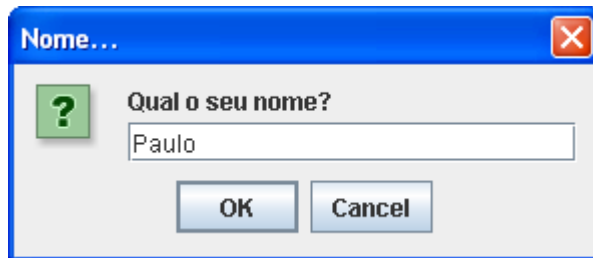
## 4.6 Caixas de Diálogo Simples - JOptionPane

- Utilização da classe
  - `javax.swing.JOptionPane`
- Caixa de Diálogo para Informação ao Usuário
  - `JOptionPane.showMessageDialog`
- Parâmetros:
  - (Component parentComponent, Object message, String title, int messageType );
  - Dois últimos argumentos podem ser omitidos
- Tipo de Mensagem (messageType)
  - `JOptionPane.PLAIN_MESSAGE` - nenhum ícone
  - `JOptionPane.ERROR_MESSAGE` - ícone de erro 
  - `JOptionPane.INFORMATION_MESSAGE` - ícone de informação 
  - `JOptionPane.WARNING_MESSAGE` - ícone de aviso 
  - `JOptionPane.QUESTION_MESSAGE` - ícone de interrogação 

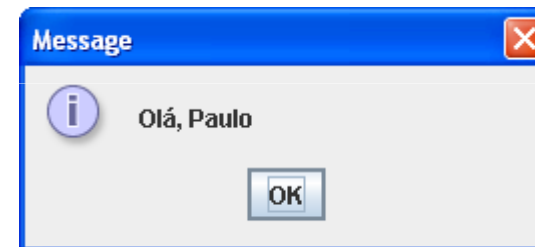
# Caixas de Diálogo Simples - JOptionPane

- Caixa de Diálogo para Captar informação do Usuário
  - JOptionPane.showInputDialog
  - Parâmetros: (Component parentComponent, Object message, String title, int messageType );
  - Dois últimos argumentos podem ser omitidos
- Caixa de Diálogo para obter confirmação do Usuário
  - JOptionPane.showConfirmDialog
  - Parâmetros: (Component parentComponent, Object message, String title, int optionType, int messageType );
  - optionType: JOptionPane.YES\_NO\_OPTION ou YES\_NO\_CANCEL\_OPTION
  - Três últimos argumentos podem ser omitidos
  - Retorna um inteiro indicando o valor digitado pelo usuário:
    - JOptionPane.YES\_OPTION, JOptionPane.NO\_OPTION, JOptionPane.CANCEL\_OPTION

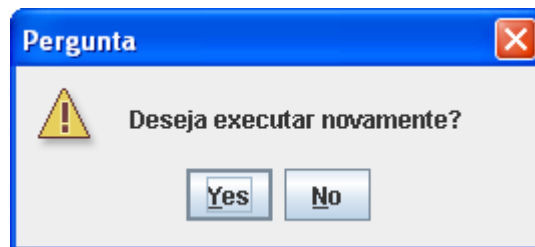
# Caixas de Diálogo Úteis - JOptionPane



`JOptionPane.showInputDialog`



`JOptionPane.showMessageDialog`



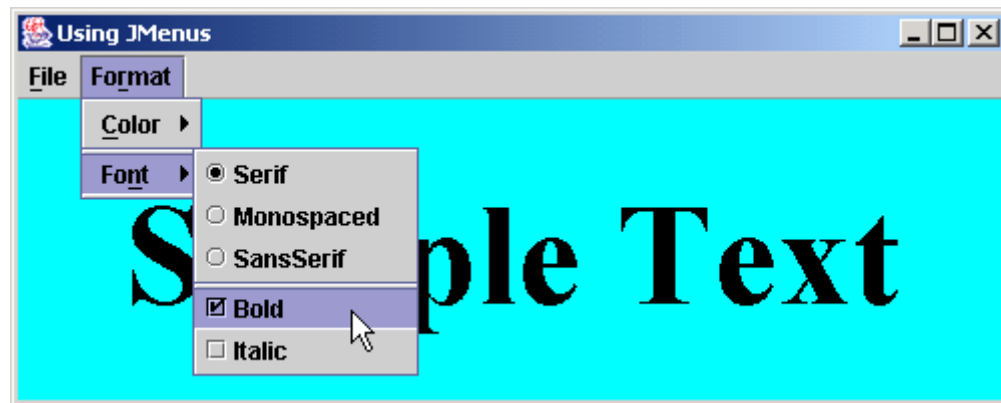
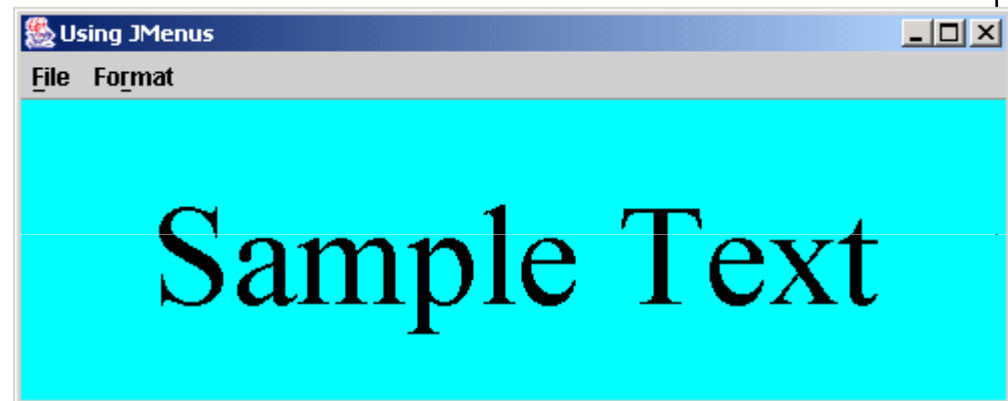
`JOptionPane.showConfirmDialog`

# Exemplo

```
1 import javax.swing.JOptionPane;
2
3 public class Dialogos {
4
5     public static void main(String args[]) {
6         int continua;
7         do {
8             String resposta;
9             resposta = JOptionPane.showInputDialog(null, "Qual o seu nome?",
10                "Nome...",JOptionPane.QUESTION_MESSAGE);
11
12             JOptionPane.showMessageDialog(null,"Olá, "+resposta);
13             continua=JOptionPane.showConfirmDialog(null,
14                "Deseja executar novamente?","Pergunta",
15                JOptionPane.YES_NO_OPTION,JOptionPane.QUESTION_MESSAGE);
16
17
18         } while(continua==JOptionPane.YES_OPTION);
19     }
20
21 }
```

## 4.7 Usando Menus com Frames

- Menus
  - Allows for performing actions with cluttering GUI
  - Contained by menu bar
    - JMenuBar
  - Comprised of menu items
    - JMenuItem



```

1 // Fig. 4.9: MenuTest.java
2 // Demonstrating menus
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6
7 public class MenuTest extends JFrame {
8     private final Color colorValues[] =
9         { Color.BLACK, Color.BLUE, Color.RED, Color.GREEN };
10    private JRadioButtonMenuItem colorItems[], fonts[];
11    private JCheckBoxMenuItem styleItems[];
12    private JLabel displayLabel;
13    private ButtonGroup fontGroup, colorGroup;
14    private int style;
15
16    // set up GUI
17    public MenuTest()
18    {
19        super( "Using JMenus" );
20
21        // set up File menu and its menu items
22        JMenu fileMenu = new JMenu( "File" );
23        fileMenu.setMnemonic( 'F' );
24

```

Instantiate **File** JMenu

```
25 // set up About... menu item
26 JMenuItem aboutItem = new JMenuItem( "About..." );
27 aboutItem.setMnemonic( 'A' );
28 fileMenu.add( aboutItem );
29 aboutItem.addActionListener(
30
31     new ActionListener() { // anonymous inner class
32
33         // display message dialog when user selects About...
34         public void actionPerformed((ActionEvent event)
35         {
36             JOptionPane.showMessageDialog( MenuTest.this,
37                 "This is an example\nof using menus",
38                 "About", JOptionPane.PLAIN_MESSAGE );
39         }
40
41     } // end anonymous inner class
42
43 ); // end call to addActionListener
44
45 // set up Exit menu item
46 JMenuItem exitItem = new JMenuItem( "Exit" );
47 exitItem.setMnemonic( 'x' );
48 fileMenu.add( exitItem );
49 exitItem.addActionListener(
50
```

Instantiate **About...** JMenuItem  
to be placed in fileMenu

When user selects **About...**  
JMenuItem, display message  
dialog with appropriate text

Instantiate **Exit** JMenuItem  
to be placed in fileMenu

```

51     new ActionListener() { // anonymous inner class
52
53         // terminate application when user clicks exitItem
54         public void actionPerformed( ActionEvent event )
55         {
56             system.exit( 0 );
57         }
58
59     } // end anonymous inner class
60
61 ); // end call to addActionListener
62
63 // create menu bar and attach it to MenuTest window
64 JMenuBar bar = new JMenuBar();
65 setJMenuBar( bar );
66 bar.add( fileMenu );
67
68 // create Format menu, its submenus and menu items
69 JMenu formatMenu = new JMenu( "Format" );
70 formatMenu.setMnemonic( 'r' );
71
72 // create Color submenu
73 String colors[] = { "Black", "Blue", "Red", "Green" };
74

```

When user selects **Exit**  
JMenuItem, exit system

Instantiate JMenuBar  
to contain JMenus

Instantiate **Format** JMenu

```

75  JMenu colorMenu = new JMenu( "color" );
76  colorMenu.setMnemonic( 'c' );
77
78  colorItems = new JRadioButtonMenuItem[ colors.length ];
79  colorGroup = new ButtonGroup();
80  ItemHandler itemHandler = new ItemHandler();
81
82  // create color radio button menu items
83  for ( int count = 0; count < colors.length; count++ ) {
84      colorItems[ count ] =
85          new JRadioButtonMenuItem( colors[ count ] );
86      colorMenu.add( colorItems[ count ] );
87      colorGroup.add( colorItems[ count ] );
88      colorItems[ count ].addActionListener( itemHandler );
89  }
90
91  // select first color menu item
92  colorItems[ 0 ].setSelected( true );
93
94  // add format menu to menu bar
95  formatMenu.add( colorMenu );
96  formatMenu.addSeparator();
97
98  // create Font submenu
99  String fontNames[] = { "Serif", "Monospaced", "SansSerif" };
100

```

Instantiate **Color JMenu**  
(submenu of **Format JMenu**)

Instantiate  
**JRadioButtonMenuItems** for  
**Color JMenu** and ensure that only  
one menu item is selected at a time

Separator places line  
between **JMenuItems**

```
101 JMenu fontMenu = new JMenu( "Font" );
102 fontMenu.setMnemonic( 'n' );
103
104 fonts = new JRadioButtonMenuItem[ fontNames.length ];
105 fontGroup = new ButtonGroup();
106
107 // create Font radio button menu items
108 for ( int count = 0; count < fonts.length; count++ ) {
109     fonts[ count ] = new JRadioButtonMenuItem( fontNames[ count ] );
110     fontMenu.add( fonts[ count ] );
111     fontGroup.add( fonts[ count ] );
112     fonts[ count ].addActionListener( itemHandler
113 }
114
115 // select first Font menu item
116 fonts[ 0 ].setSelected( true );
117
118 fontMenu.addSeparator();
119
120 // set up style menu items
121 String styleNames[] = { "Bold", "Italic" };
122
123 styleItems = new JCheckBoxMenuItem[ styleNames.length ];
124 styleHandler styleHandler = new styleHandler();
125
```

Instantiate **Font** JMenu  
(submenu of **Format** JMenu)

Instantiate  
JRadioButtonMenuItems for  
**Font** JMenu and ensure that only  
one menu item is selected at a time

```

126 // create style checkbox menu items
127 for ( int count = 0; count < styleNames.length; count++ ) {
128     styleItems[ count ] =
129         new JCheckBoxMenuItem( styleNames[ count ] );
130     fontMenu.add( styleItems[ count ] );
131     styleItems[ count ].addItemListener( styleHandler );
132 }
133
134 // put Font menu in Format menu
135 formatMenu.add( fontMenu );
136
137 // add Format menu to menu bar
138 bar.add( formatMenu );
139
140 // set up label to display text
141 displayLabel = new JLabel( "Sample Text", SwingConstants.CENTER );
142 displayLabel.setForeground( colorValues[ 0 ] );
143 displayLabel.setFont( new Font( "Serif", Font.PLAIN, 72 ) );
144
145 getContentPane().setBackground( Color.CYAN );
146 getContentPane().add( displayLabel, BorderLayout.CENTER );
147
148 setSize( 500, 200 );
149 setVisible( true );
150
151 } // end constructor
152

```

```

153 public static void main( String args[] )
154 {
155     MenuTest application = new MenuTest();
156     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
157 }
158
159 // inner class to handle action events from menu items
160 private class ItemHandler implements ActionListener {
161
162     // process color and font selections
163     public void actionPerformed((ActionEvent event) )
164     {
165         // process color selection
166         for ( int count = 0; count < colorItems.length; count++ )
167
168             if ( colorItems[ count ].isSelected() ) {
169                 displayLabel.setForeground( colorValues[ count ] );
170                 break;
171             }
172
173         // process font selection
174         for ( int count = 0; count < fonts.length; count++ )
175
176             if ( event.getSource() == fonts[ count ] ) {
177                 displayLabel.setFont(
178                     new Font( fonts[ count ].getText(), style, 72 ) );
179                 break;
180             }
181     }
182 }

```

Invoked when user selects JMenuItem

Determine which font or color menu generated event

Set font or color of JLabel, respectively

```

181     repaint();
182
183     } // end method actionPerformed
184
185 } // end class ItemHandler
186
187 // inner class to handle item events from check b
188 private class StyleHandler implements ItemListene
189
190     // process font style selections
191     public void itemStateChanged( ItemEvent e )
192     {
193         style = 0;
194
195         // check for bold selection
196         if ( styleItems[ 0 ].isSelected() )
197             style += Font.BOLD;
198
199         // check for italic selection
200         if ( styleItems[ 1 ].isSelected() )
201             style += Font.ITALIC;
202
203         displayLabel.setFont(
204             new Font( displayLabel.getFont().getName(), style, 72 ) );
205

```

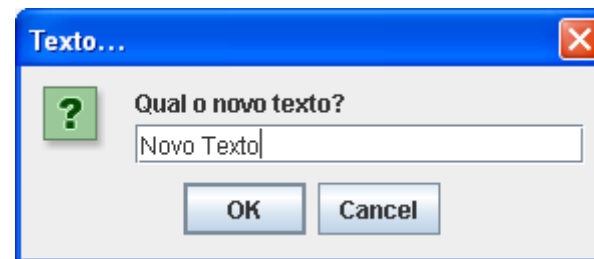
Invoked when user selects  
JCheckBoxMenuItem

Determine new font style

```
206
207     repaint();
208     }
209
210 } // end class StyleHandler
211
212 } // end class MenuTest
```

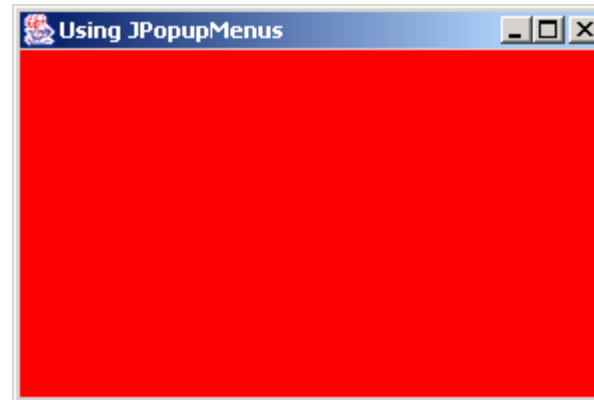
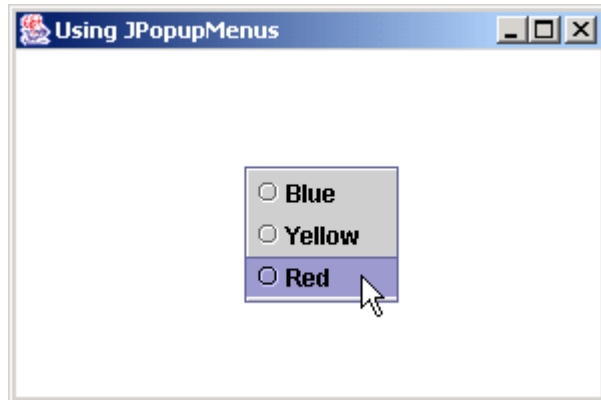
## Exercício

- Introduza um novo item no Menu File. O item deve ser “Text”, a tecla mnemônica deve ser ‘T’. Ao ser acionado, o item abre uma caixa de diálogo que pede ao usuário um novo texto, o qual será utilizado para modificar o texto apresentado ao centro da janela. Mantendo a cor, estilo e fonte do texto.



## 4.8 JPopupMenu

- Context-sensitive popup menus
  - JPopupMenu
  - Menu generated depending on which component is accessed



```

1 // Fig. 4.10: PopupTest.java
2 // Demonstrating JPopupMenu
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6
7 public class PopupTest extends JFrame {
8     private JRadioButtonMenuItem items[];
9     private final Color colorValues[] =
10         { Color.BLUE, Color.YELLOW, Color.RED };
11     private JPopupMenu popupMenu;
12
13     // set up GUI
14     public PopupTest()
15     {
16         super( "Using JPopupMenu" );
17
18         ItemHandler handler = new ItemHandler();
19         String colors[] = { "Blue", "Yellow", "Red" };
20
21         // set up popup menu and its items
22         ButtonGroup colorGroup = new ButtonGroup();
23         popupMenu = new JPopupMenu();
24         items = new JRadioButtonMenuItem[ 3 ];
25

```

Instantiate JPopupMenu object

```

26 // construct each menu item and add to popup menu; also
27 // enable event handling for each menu item
28 for ( int count = 0; count < items.length; count++ ) {
29     items[ count ] = new JRadioButtonMenuItem( colors[ count ] );
30     popupMenu.add( items[ count ] );
31     colorGroup.add( items[ count ] );
32     items[ count ].addActionListener( handler );
33 }
34
35 getContentPane().setBackground( Color.WHITE );
36
37 // declare a MouseListener for the window that displays
38 // a JPopupMenu when the popup trigger event occurs
39 addMouseListener(
40
41     new MouseAdapter() { // anonymous inner class
42
43         // handle mouse press event
44         public void mousePressed( MouseEvent event )
45         {
46             checkForTriggerEvent( event );
47         }
48
49         // handle mouse release event
50         public void mouseReleased( MouseEvent event )
51         {
52             checkForTriggerEvent( event );
53     }

```

Create JRadioButtonMenuItem objects to add to JPopupMenu

Determine whether popup-trigger event occurred when user presses or releases mouse button

```

54
55     // determine whether event should trigger popup menu
56     private void checkForTriggerEvent( MouseEvent event )
57     {
58         if ( event.isPopupTrigger() )
59             popupMenu.show(
60                 event.getComponent(), event.getX(), event.getY() );
61     }
62
63     } // end anonymous inner clas
64
65     ); // end call to addMouseListener
66
67     setSize( 300, 200 );
68     setVisible( true );
69
70 } // end constructor PopupTest
71
72 public static void main( String args[] )
73 {
74     PopupTest application = new PopupTest();
75     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
76 }
77

```

Show JPopupMenu if  
popup-trigger occurred

```
78 // private inner class to handle menu item events
79 private class ItemHandler implements ActionListener {
80
81     // process menu item selections
82     public void actionPerformed( ActionEvent event )
83     {
84         // determine which menu item was selected
85         for ( int i = 0; i < items.length; i++ )
86             if ( event.getSource() == items[ i ] ) {
87                 getContentPane().setBackground( colorValues[ i ] );
88                 return;
89             }
90     }
91
92 } // end private inner class ItemHandler
93
94 } // end class PopupTest
```

Invoked when user selects  
JRadioButtonMenuItem

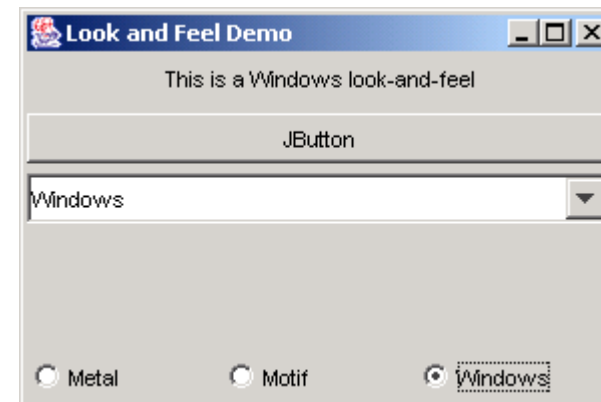
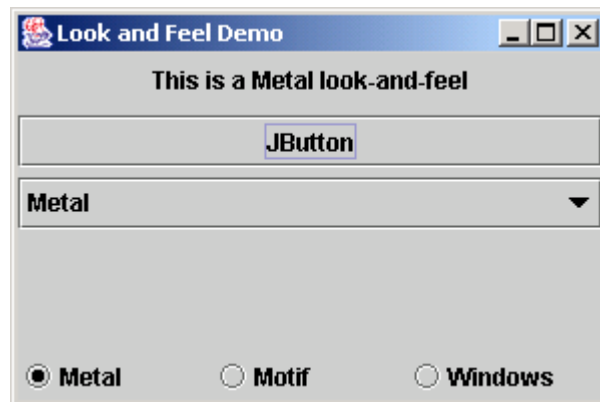
Determine which  
JRadioButtonMenuItem was selected,  
then set window background color

## Exercício

- Altere o código do exemplo anterior para incluir as opções Green, Black e White, no pop-up Menu e que ao serem selecionadas alterem a cor de fundo da janela para a cor correspondente

## 4.9 Pluggable Look-and-Feel

- É possível mudar a aparência de sua aplicação Swing em tempo de execução
  - Windows
  - Motif
  - Java
- **UIManager.LookAndFeelInfo** looks
- **// recupera os LookAndFeel instalados**
- **looks=UIManager.getInstalledLookAndFeels();**
- **//muda o LookAndFeel**
- **UIManager.setLookAndFeel(look[i]);**



```
1 // Fig. 4.11: LookAndFeelDemo.java
2 // Changing the look and feel.
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6
7 public class LookAndFeelDemo extends JFrame {
8     private final String strings[] = { "Metal", "Motif", "Windows" };
9     private UIManager.LookAndFeelInfo looks[];
10    private JRadioButton radio[];
11    private ButtonGroup group;
12    private JButton button;
13    private JLabel label;
14    private JComboBox comboBox;
15
16    // set up GUI
17    public LookAndFeelDemo()
18    {
19        super( "Look and Feel Demo" );
20
21        Container container = getContentPane();
22
23        // set up panel for NORTH of BorderLayout
24        JPanel northPanel = new JPanel();
25        northPanel.setLayout( new GridLayout( 3, 1, 0, 5 ) );
26
```

Hold installed look-and-feel information

```
27 // set up label for NORTH panel
28 label = new JLabel( "This is a Metal look-and-feel",
29     SwingConstants.CENTER );
30 northPanel.add( label );
31
32 // set up button for NORTH panel
33 button = new JButton( "JButton" );
34 northPanel.add( button );
35
36 // set up combo box for NORTH panel
37 comboBox = new JComboBox( strings );
38 northPanel.add( comboBox );
39
40 // create array for radio buttons
41 radio = new JRadioButton[ strings.length ];
42
43 // set up panel for SOUTH of BorderLayout
44 JPanel southPanel = new JPanel();
45 southPanel.setLayout( new GridLayout( 1, radio.length ) );
46
47 // set up radio buttons for SOUTH panel
48 group = new ButtonGroup();
49 ItemHandler handler = new ItemHandler();
50
```

```

51     for ( int count = 0; count < radio.length; count++ ) {
52         radio[ count ] = new JRadioButton( strings[ count ] );
53         radio[ count ].addItemListener( handler );
54         group.add( radio[ count ] );
55         southPanel.add( radio[ count ] );
56     }
57
58     // attach NORTH and SOUTH panels to content pane
59     container.add( northPanel, BorderLayout.NORTH );
60     container.add( southPanel, BorderLayout.SOUTH );
61
62     // get installed look-and-feel information
63     looks = UIManager.getInstalledLookAndFeels();
64
65     setSize( 300, 200 );
66     setVisible( true );
67
68     radio[ 0 ].setSelected( true );
69
70 } // end constructor LookAndFeelDemo
71
72 // use UIManager to change look-and-feel of GUI
73 private void changeTheLookAndFeel( int value )
74 {

```

```

75     // change look and feel
76     try {
77         UIManager.setLookAndFeel( looks[ value ].getClassName() );
78         SwingUtilities.updateComponentTreeUI( this );
79     }
80
81     // process problems changing look and feel
82     catch ( Exception exception ) {
83         exception.printStackTrace();
84     }
85 }
86
87 public static void main( String args[] )
88 {
89     LookAndFeelDemo application = new LookAndFeelDemo();
90     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
91 }
92
93 // private inner class to handle radio button events
94 private class ItemHandler implements ItemListener {
95
96     // process user's look-and-feel selection
97     public void itemStateChanged( ItemEvent event )
98     {
99         for ( int count = 0; count < radio.length; count++ )
100

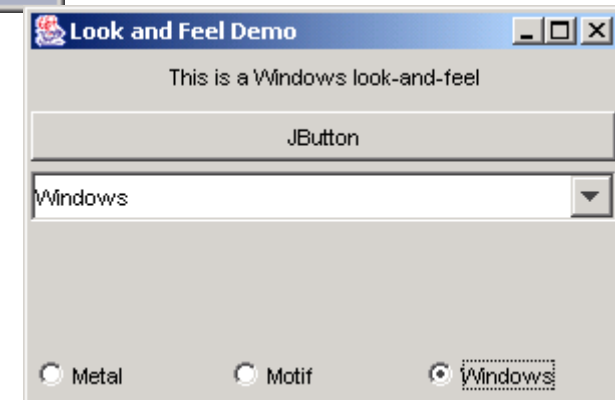
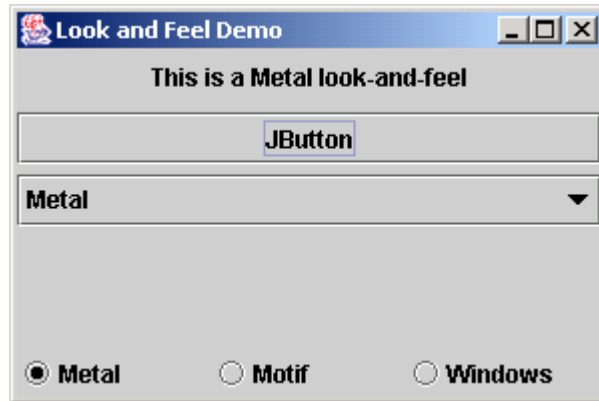
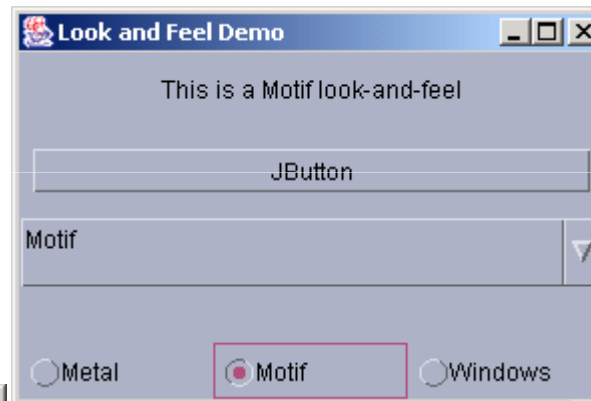
```

Change look-and-feel

```

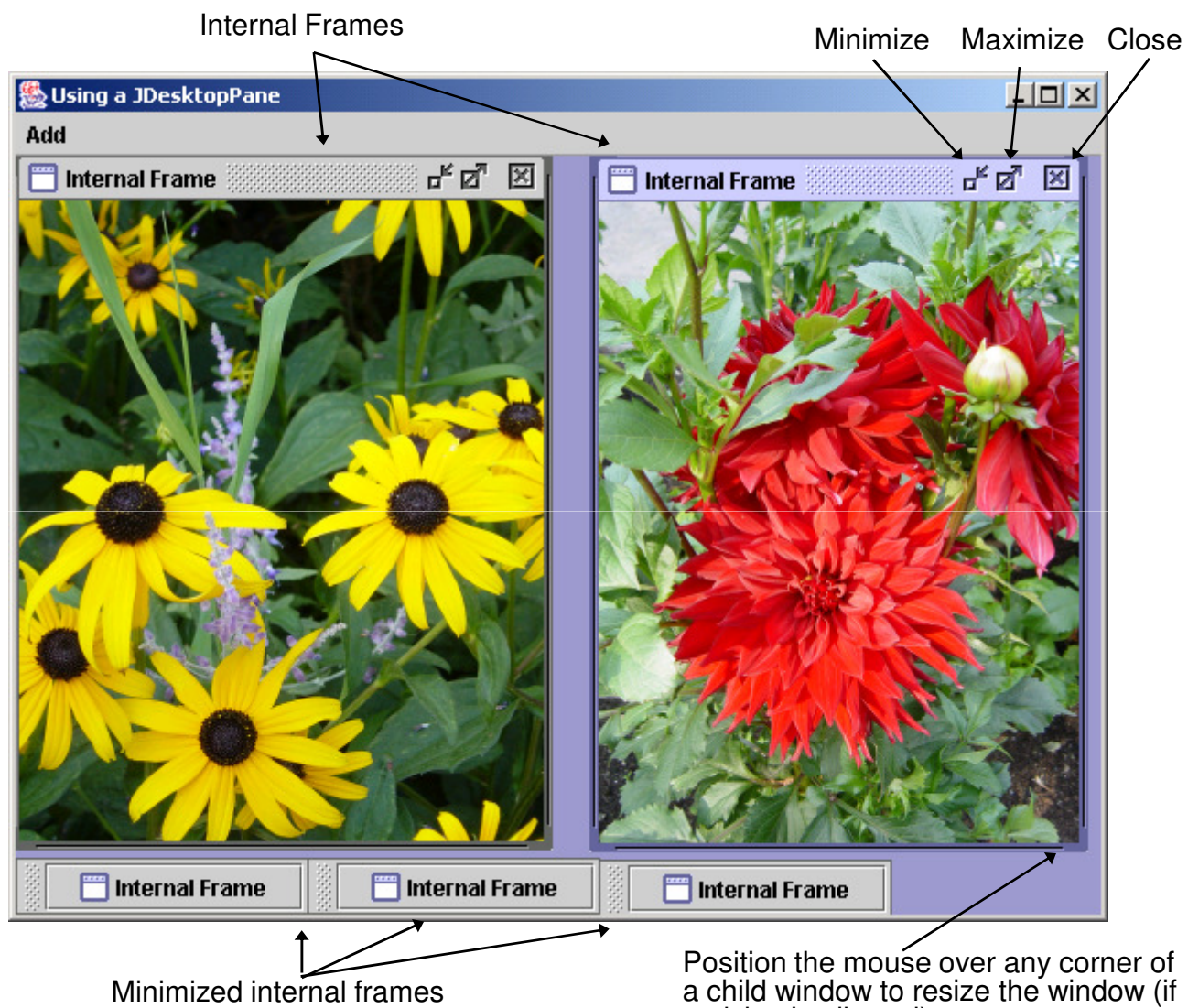
101         if ( radio[ count ].isSelected() ) {
102             label.setText( "This is a " +
103                 strings[ count ] + " look-and-feel" );
104             comboBox.setSelectedIndex( count );
105             changeTheLookAndFeel( count );
106         }
107     }
108
109 } // end private inner class ItemHandler
110
111 } // end class LookAndFeelDemo

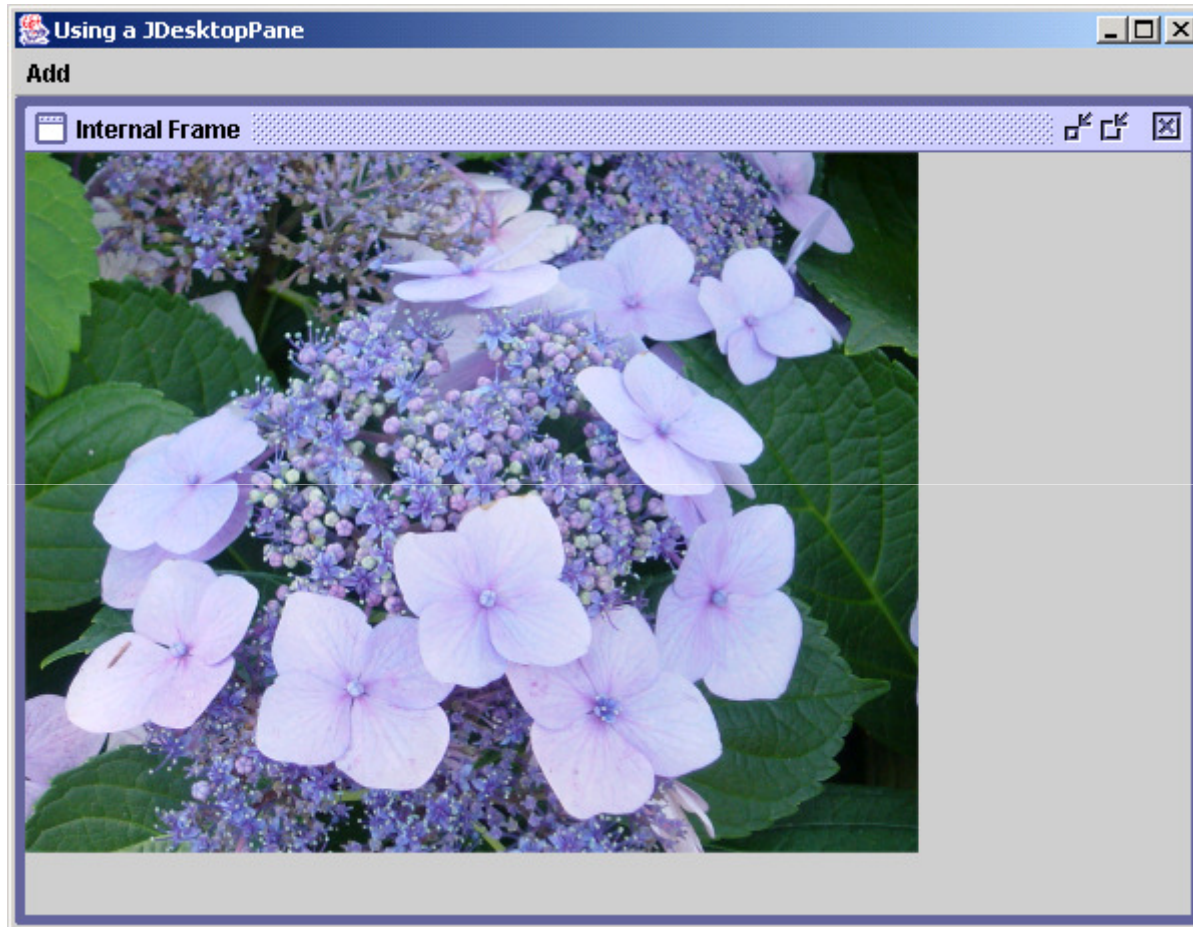
```



## 4.10 JDesktopPane and JInternalFrame

- Multiple document interface
  - Main (parent) window
  - Child windows
  - Switch freely among documents





```

1 // Fig. 4.12: DesktopTest.java
2 // Demonstrating JDesktopPane.
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6
7 public class DesktopTest extends JFrame {
8     private JDesktopPane theDesktop;
9
10    // set up GUI
11    public DesktopTest()
12    {
13        super( "Using a JDesktopPane" );
14
15        // create menu bar, menu and menu item
16        JMenuBar bar = new JMenuBar();
17        JMenu addMenu = new JMenu( "Add" );
18        JMenuItem newFrame = new JMenuItem( "Internal Frame" );
19
20        addMenu.add( newFrame );
21        bar.add( addMenu );
22
23        setJMenuBar( bar );
24
25        // set up desktop
26        theDesktop = new JDesktopPane();
27        getContentPane().add( theDesktop );

```

Manages JInternalFrame child windows displayed in JDesktopPane

28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54

```
// set up listener for newFrame menu item
newFrame.addActionListener(
    new ActionListener() { // anonymous inner class
        // display new internal window
        public void actionPerformed( ActionEvent event ) {
            // create internal frame
            JInternalFrame frame = new JInternalFrame(
                "Internal Frame", true, true, true, true );
            // attach panel to internal frame content pane
            Container container = frame.getContentPane();
            MyJPanel panel = new MyJPanel();
            container.add( panel, BorderLayout.CENTER );
            // set size internal frame to size of its contents
            frame.pack();
            // attach internal frame to desktop and show it
            theDesktop.add( frame );
            frame.setVisible( true );
        }
    } // end anonymous inner class
```

Handle event when user  
selects JMenuItem

Invoked when user  
selects JMenuItem

Create JInternalFrame

JPanels can be added  
to JInternalFrames

Use preferred  
size for window

```

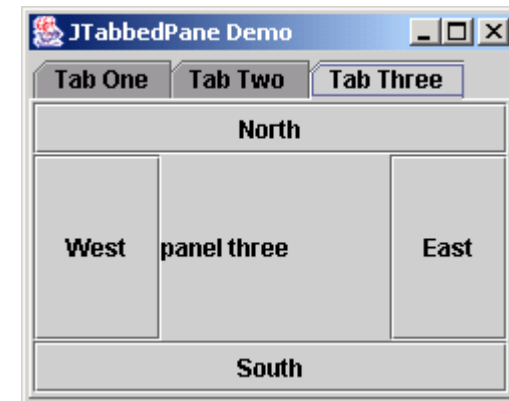
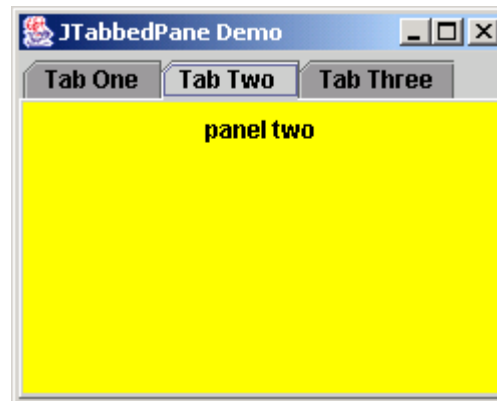
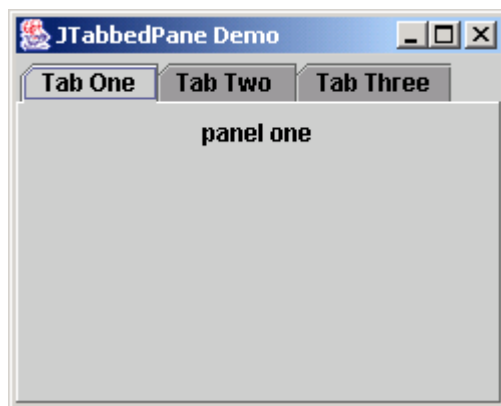
55
56     ); // end call to addActionListener
57
58     setSize( 600, 460 );
59     setVisible( true );
60
61 } // end constructor
62
63 public static void main( String args[] )
64 {
65     DesktopTest application = new DesktopTest();
66     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
67 }
68
69 } // end class DesktopTest
70
71 // class to display an ImageIcon on a panel
72 class MyJPanel extends JPanel {
73     private ImageIcon imageIcon;
74     private String[] images = { "yellowflowers.png", "purpleflowers.png",
75         "redflowers.png", "redflowers2.png", "lavenderflowers.png" };
76
77     // load image
78     public MyJPanel()
79     {

```

```
80     int randomNumber = ( int ) ( Math.random() * 5 );
81     ImageIcon = new ImageIcon( images[ randomNumber ] );
82 }
83
84 // display ImageIcon on panel
85 public void paintComponent( Graphics g )
86 {
87     // call superclass paintComponent method
88     super.paintComponent( g );
89
90     // display icon
91     ImageIcon.paintIcon( this, g, 0, 0 );
92 }
93
94 // return image dimensions
95 public Dimension getPreferredSize()
96 {
97     return new Dimension( ImageIcon.getWidth(),
98         ImageIcon.getHeight() );
99 }
100
101 } // end class MyJPanel
```

## 4.11 JTabbedPane

- Arranges GUI components into layers
  - One layer visible at a time
  - Access each layer via a tab
  - JTabbedPane



```

1 // Fig. 4.13: JTabbedPaneDemo.java
2 // Demonstrating JTabbedPane.
3 import java.awt.*;
4 import javax.swing.*;
5
6 public class JTabbedPaneDemo extends JFrame {
7
8     // set up GUI
9     public JTabbedPaneDemo()
10    {
11        super( "JTabbedPane Demo " );
12
13        // create JTabbedPane
14        JTabbedPane tabbedPane = new JTabbedPane();
15
16        // set up pane1 and add it to JTabbedPane
17        JLabel label1 = new JLabel( "pane1 one", SwingConstants.CENTER );
18        JPanel pane1 = new JPanel();
19        pane1.add( label1 );
20        tabbedPane.addTab( "Tab One", null, pane1, "First Panel" );
21
22        // set up pane2 and add it to JTabbedPane
23        JLabel label2 = new JLabel( "pane1 two", SwingConstants.CENTER );
24        JPanel pane2 = new JPanel();
25        pane2.setBackground( Color.YELLOW );
26        pane2.add( label2 );
27        tabbedPane.addTab( "Tab Two", null, pane2, "Second Panel" );

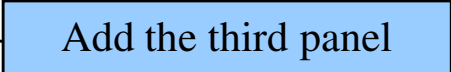
```

```

28
29 // set up panel3 and add it to JTabbedPane
30 JLabel label3 = new JLabel( "panel three" );
31 JPanel panel3 = new JPanel();
32 panel3.setLayout( new BorderLayout() );
33 panel3.add( new JButton( "North" ), BorderLayout.NORTH );
34 panel3.add( new JButton( "West" ), BorderLayout.WEST );
35 panel3.add( new JButton( "East" ), BorderLayout.EAST );
36 panel3.add( new JButton( "South" ), BorderLayout.SOUTH );
37 panel3.add( label3, BorderLayout.CENTER );
38 tabbedPane.addTab( "Tab Three", null, panel3, "Third Panel" );
39
40 // add JTabbedPane to container
41 getContentPane().add( tabbedPane );
42
43 setSize( 250, 200 );
44 setVisible( true );
45
46 } // end constructor
47
48 public static void main( String args[] )
49 {
50     JTabbedPaneDemo tabbedPaneDemo = new JTabbedPaneDemo();
51     tabbedPaneDemo.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
52 }
53
54 } // end class CardDeck

```

Add the third panel

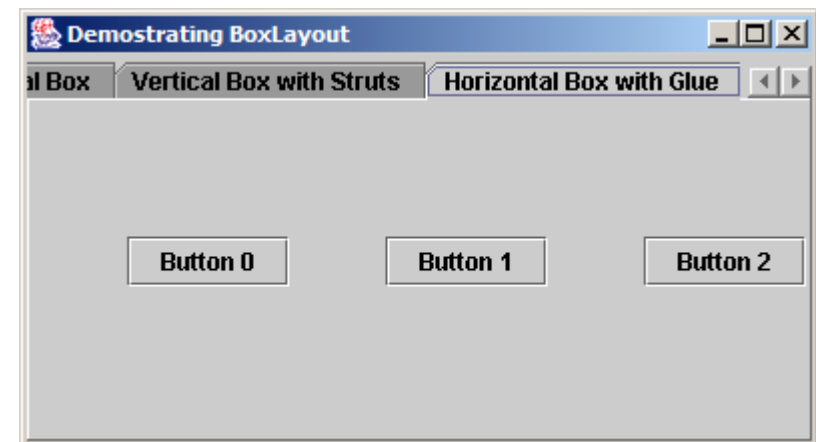
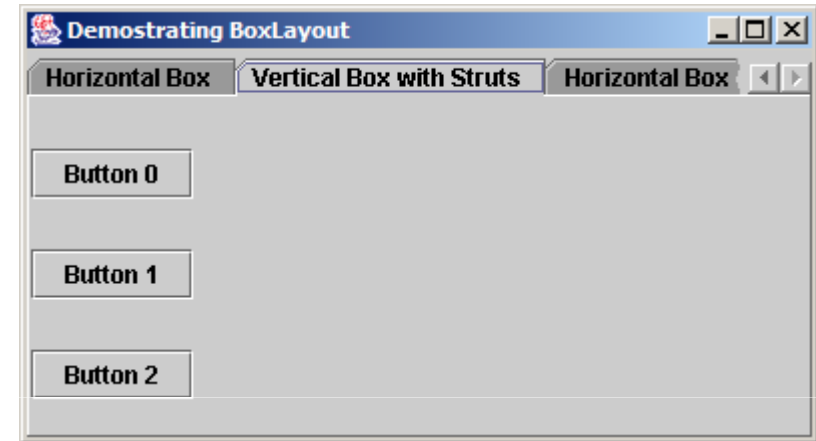
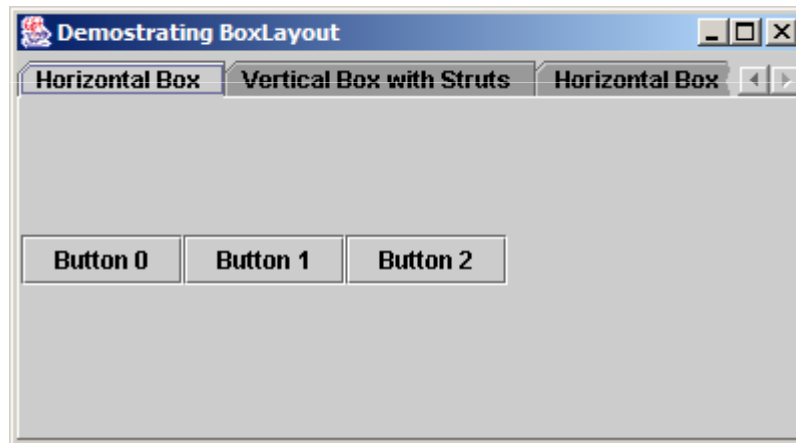


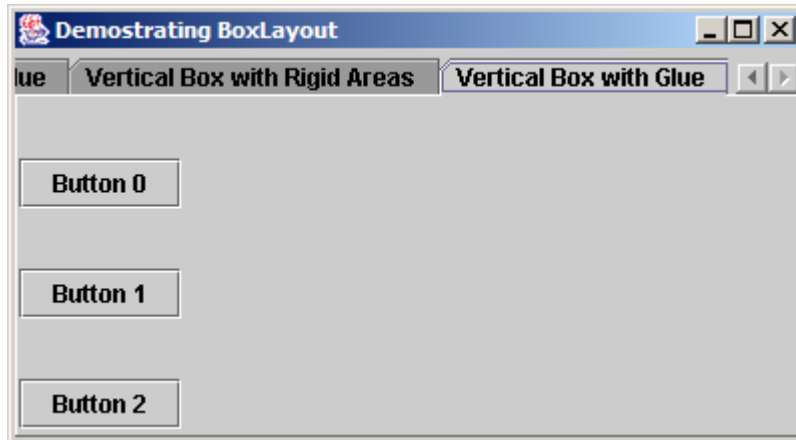
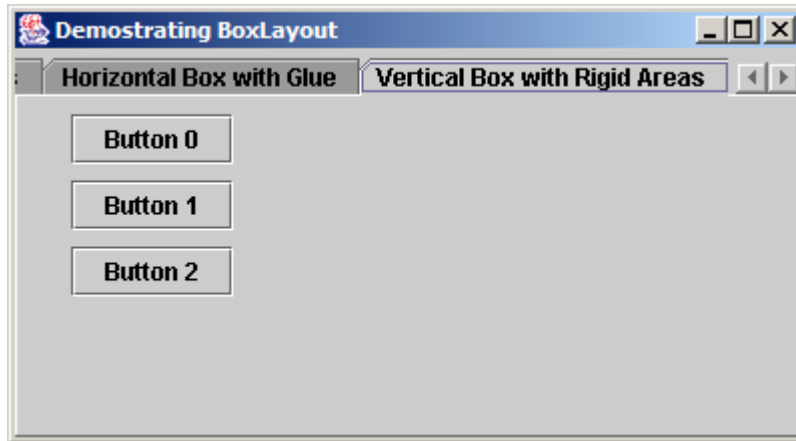
## 4.12 Layout Managers: BorderLayout e GridBagLayout

- Dois Novos Layout Managers
  - BorderLayout:
    - Permite que os componentes GUI sejam arranjados da esquerda para direita ou de cima para baixo em um container. A classe Box declara um container com BorderLayout como Layout padrão e provê métodos estáticos para criar Box com BorderLayout vertical ou horizontal
  - GridBagLayout:
    - Similar a GridLayout, porém cada componente pode variar de tamanho e os componentes podem ser adicionados em qualquer ordem.

# BoxLayout Layout Manager

- **BoxLayout**
  - Arranges GUI components
    - Horizontally along x-axis
    - Vertically along y-axis





```
1 // Fig. 4.15: BorderLayoutDemo.java
2 // Demonstrating BorderLayout.
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6
7 public class BorderLayoutDemo extends JFrame {
8
9     // set up GUI
10    public BorderLayoutDemo()
11    {
12        super( "Demonstrating BorderLayout" );
13
14        // create Box containers with BorderLayout
15        Box horizontal1 = Box.createHorizontalBox();
16        Box vertical1 = Box.createVerticalBox();
17        Box horizontal2 = Box.createHorizontalBox();
18        Box vertical2 = Box.createVerticalBox();
19
20        final int SIZE = 3; // number of buttons on each Box
21
22        // add buttons to Box horizontal1
23        for ( int count = 0; count < SIZE; count++ )
24            horizontal1.add( new JButton( "Button " + count ) );
25
```

Create BOXes

Add three JButtons to  
horizontal BOX

```

26 // create strut and add buttons to Box vertical1
27 for ( int count = 0; count < SIZE; count++ ) {
28     vertical1.add( Box.createVerticalStrut( 25 ) );
29     vertical1.add( new JButton( "Button " + count ) );
30 }
31
32 // create horizontal glue and add buttons to Box horizontal2
33 for ( int count = 0; count < SIZE; count++ ) {
34     horizontal2.add( Box.createHorizontalGlue( 0 ) );
35     horizontal2.add( new JButton( "Button " + count ) );
36 }
37
38 // create rigid area and add buttons to Box vertical2
39 for ( int count = 0; count < SIZE; count++ ) {
40     vertical2.add( Box.createRigidArea( new Dim( 25, 0 ) ) );
41     vertical2.add( new JButton( "Button " + count ) );
42 }
43
44 // create vertical glue and add buttons to panel
45 JPanel panel = new JPanel();
46 panel.setLayout( new BorderLayout( panel, BorderLayout.Y_AXIS ) );
47
48 for ( int count = 0; count < SIZE; count++ ) {
49     panel.add( Box.createGlue( 0 ) );
50     panel.add( new JButton( "Button " + count ) );
51 }
52

```

Add three JButtons to vertical BOX

Strut guarantees space between components

Add three JButtons to horizontal BOX

Glue guarantees expandable space between components

Add three JButtons to vertical BOX

Rigid area guarantees fixed component size

```

53 // create a JTabbedPane
54 JTabbedPane tabs = new JTabbedPane(
55     JTabbedPane.TOP, JTabbedPane.SCROLL_TAB_LAYOUT );
56
57 // place each container on tabbed pane
58 tabs.addTab( "Horizontal Box", horizontal1 );
59 tabs.addTab( "Vertical Box with Struts", vertical1 );
60 tabs.addTab( "Horizontal Box with Glue", horizontal2 );
61 tabs.addTab( "Vertical Box with Rigid Areas", vertical2 );
62 tabs.addTab( "Vertical Box with Glue", panel );
63
64 getContentPane().add( tabs ); // place tabbed pane on content pane
65
66 setSize( 400, 220 );
67 setVisible( true );
68
69 } // end constructor
70
71 public static void main( String args[] )
72 {
73     BoxLayoutDemo application = new BoxLayoutDemo();
74     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
75 }
76
77 } // end class BoxLayoutDemo

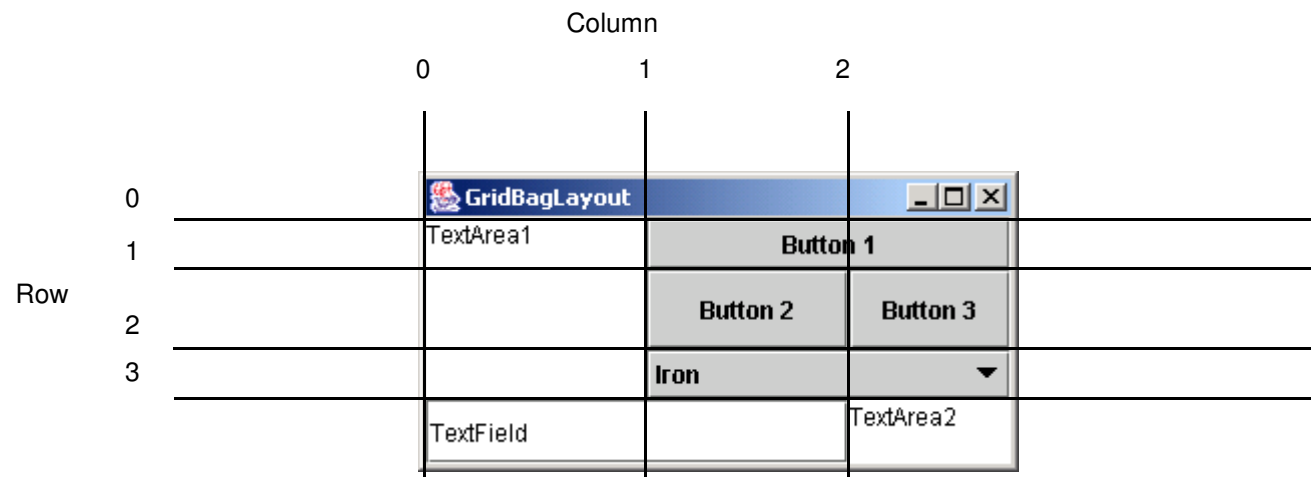
```

Create a JTabbedPane  
to hold the BOXes

# GridBagLayout Layout Manager

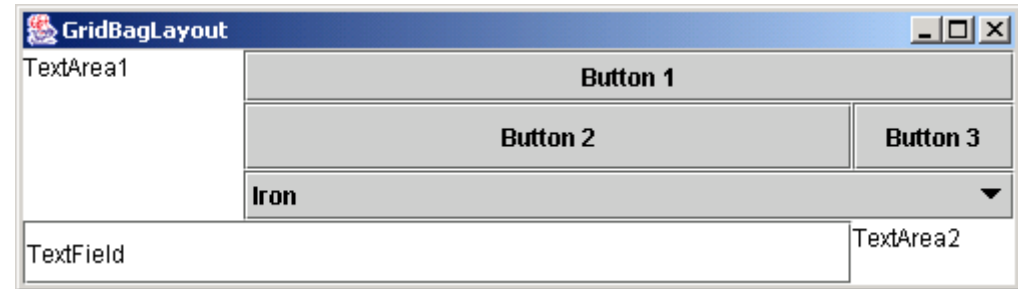
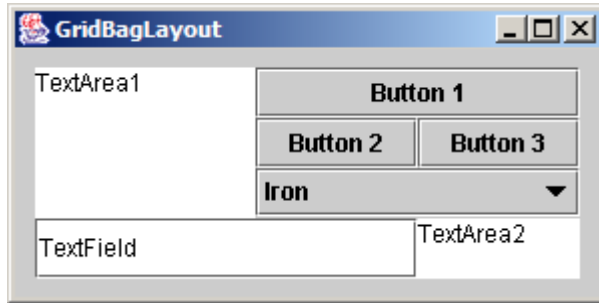
- GridBagLayout
  - Flexible GridBagLayout
    - Components can vary in size
    - Components can occupy multiple rows and columns
    - Components can be added in any order
  - Uses GridBagConstraints
    - Specifies how component is placed in GridBagLayout

# Fig. 4.16 Designing a GUI that will use GridBagLayout

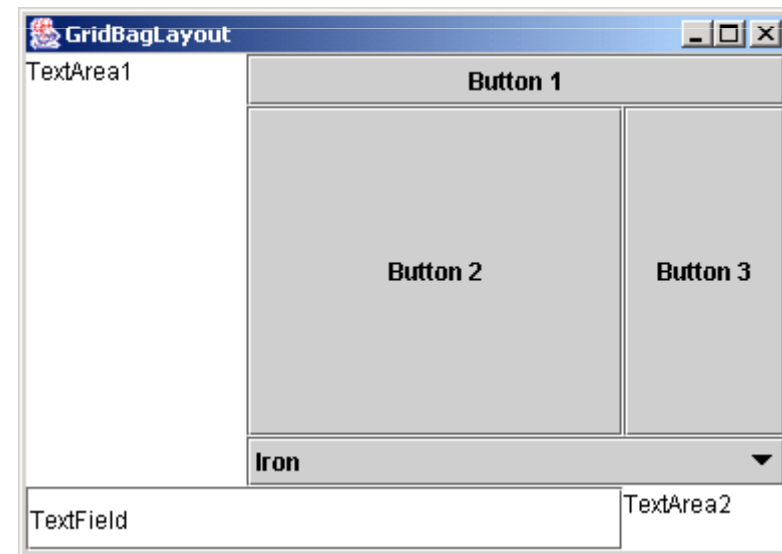


## Fig. 4.17 GridBagConstraints fields

GridBagConstraints field	Description
fill	Resize the component in specified direction (NONE, HORIZONTAL, VERTICAL, BOTH) when the display area is larger than the component.
gridx	The column in which the component will be placed.
gridy	The row in which the component will be placed.
gridwidth	The number of columns the component occupies.
gridheight	The number of rows the component occupies.
weightx	The portion of extra space to allocate horizontally. The grid slot can become wider when extra space is available.
weighty	The portion of extra space to allocate vertically. The grid slot can become taller when extra space is available.



GridBagLayout with the weights set to zero



```

1 // Fig. 4.19: GridBagDemo.java
2 // Demonstrating GridBagLayout.
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6
7 public class GridBagDemo extends JFrame {
8     private Container container;
9     private GridBagLayout layout;
10    private GridBagConstraints constraints;
11
12    // set up GUI
13    public GridBagDemo()
14    {
15        super( "GridBagLayout" );
16
17        container = getContentPane();
18        layout = new GridBagLayout();
19        container.setLayout( layout );
20
21        // instantiate gridbag constraints
22        constraints = new GridBagConstraints();
23
24        // create GUI components
25        JTextArea textArea1 = new JTextArea( "TextArea1", 5, 10 );
26        JTextArea textArea2 = new JTextArea( "TextArea2", 2, 2 );
27

```

Set GridBagLayout  
as layout manager

Used to determine  
component location  
and size in grid

```

28 String names[] = { "Iron", "Steel", "Brass" };
29 JComboBox comboBox = new JComboBox( names );
30
31 JTextField textField = new JTextField( "TextField" );
32 JButton button1 = new JButton( "Button 1" );
33 JButton button2 = new JButton( "Button 2" );
34 JButton button3 = new JButton( "Button 3" );
35
36 // weightx and weighty for textArea1 are both 0: the default
37 // anchor for all components is CENTER: the default
38 constraints.fill = GridBagConstraints.BOTH;
39 addComponent( textArea1, 0, 0, 1, 3 );
40
41 // weightx and weighty for button1 are both 0: the default
42 constraints.fill = GridBagConstraints.HORIZONTAL;
43 addComponent( button1, 0, 1, 2, 1 );
44
45 // weightx and weighty for comboBox are both 0: the default
46 // fill is HORIZONTAL
47 addComponent( comboBox, 2, 1, 2, 1 );
48
49 // button2
50 constraints.weightx = 1000; // can grow wider
51 constraints.weighty = 1; // can grow taller
52 constraints.fill = GridBagConstraints.BOTH;
53 addComponent( button2, 1, 1, 1, 1 );
54

```

If user resizes Container, first JTextArea is filled entire allocated area in grid

First JTextArea spans one row and three columns

If user resizes Container, first JButton fills horizontally in grid

First JButton spans two rows and one column

If user resizes Container, second JButton fills extra space

```
55     // fill is BOTH for button3
56     constraints.weightx = 0;
57     constraints.weighty = 0;
58     addComponent( button3, 1, 2, 1, 1 );
59
60     // weightx and weighty for textField are both 0, fill is BOTH
61     addComponent( textField, 3, 0, 2, 1 );
62
63     // weightx and weighty for textArea2 are both 0, fill is BOTH
64     addComponent( textArea2, 3, 2, 1, 1 );
65
66     setSize( 300, 150 );
67     setVisible( true );
68
69 } // end constructor GridBagDemo
70
71 // method to set constraints on
72 private void addComponent( Component component,
73     int row, int column, int width, int height )
74 {
75     // set gridx and gridy
76     constraints.gridx = column;
77     constraints.gridy = row;
78
```

```
79     // set gridwidth and gridheight
80     constraints.gridwidth = width;
81     constraints.gridheight = height;
82
83     // set constraints and add component
84     layout.setConstraints( component, constraints );
85     container.add( component );
86 }
87
88 public static void main( String args[] )
89 {
90     GridBagDemo application = new GridBagDemo();
91     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
92 }
93
94 } // end class GridBagDemo
```

# GridBagConstraints Constants RELATIVE and REMAINDER

- Constants RELATIVE and REMAINDER
  - Usados em substituição a `gridx` e `gridy`
  - RELATIVE
    - Especifica proximidade necessária ao último componente adicionado  
`constraints.gridwidth = GridBagConstraints.RELATIVE;`
    - `layout.setConstraints( component, constraints );`
    - `Container.add(component);`
  - REMAINDER
    - Especifica que o componente deve ser o último da linha ou coluna
    - `constraints.gridwidth = GridBagConstraints.REMAINDER;`



```
1 // Fig. 4.20: GridBagDemo2.java
2 // Demonstrating GridBagLayout constants.
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6
7 public class GridBagDemo2 extends JFrame {
8     private GridBagLayout layout;
9     private GridBagConstraints constraints;
10    private Container container;
11
12    // set up GUI
13    public GridBagDemo2()
14    {
15        super( "GridBagLayout" );
16
17        container = getContentPane();
18        layout = new GridBagLayout();
19        container.setLayout( layout );
20
21        // instantiate gridbag constraints
22        constraints = new GridBagConstraints();
23
24        // create GUI components
25        String metals[] = { "Copper", "Aluminum", "Silver" };
26        JComboBox comboBox = new JComboBox( metals );
27
```

Set GridBagLayout  
as layout manager

Used to determine  
component location  
and size in grid

```

28     JTextField textField = new JTextField( "TextField" );
29
30     String fonts[] = { "Serif", "Monospaced" };
31     JList list = new JList( fonts );
32
33     String names[] = { "zero", "one", "two", "three", "four" };
34     JButton buttons[] = new JButton[ names.length ];
35
36     for ( int count = 0; count < buttons.length; count++ )
37         buttons[ count ] = new JButton( names[ count ] );
38
39     // define GUI component constraints for textField
40     constraints.gridx = 1;
41     constraints.gridy = 1;
42     constraints.fill = GridBagConstraints.BOTH;
43     constraints.gridwidth = GridBagConstraints.REMAINDER;
44     addComponent( textField );
45
46     // buttons[0] -- weightx and weighty are 1: fill is BOTH
47     constraints.gridwidth = 1;
48     addComponent( buttons[ 0 ] );
49
50     // buttons[1] -- weightx and weighty are 1: fill is BOTH
51     constraints.gridwidth = GridBagConstraints.RELATIVE;
52     addComponent( buttons[ 1 ] );
53

```

Specify textField as last  
(only) component in first row

Place button[0] as first  
component in second row

Place button[1] right  
next to button[0]

```

54 // buttons[2] -- weightx and weighty are 1: fill is BOTH
55 constraints.gridwidth = GridBagConstraints.REMAINDER;
56 addComponent( buttons[ 2 ] );
57
58 // comboBox -- weightx is 1: fill is BOTH
59 constraints.weighty = 0;
60 constraints.gridwidth = GridBagConstraints.REMAINDER;
61 addComponent( comboBox );
62
63 // buttons[3] -- weightx is 1: fill is BOTH
64 constraints.weighty = 1;
65 constraints.gridwidth = GridBagConstraints.REMAINDER;
66 addComponent( buttons[ 3 ] );
67
68 // buttons[4] -- weightx and weighty are 1: fill is BOTH
69 constraints.gridwidth = GridBagConstraints.RELATIVE;
70 addComponent( buttons[ 4 ] );
71
72 // list -- weightx and weighty are 1: fill is BOTH
73 constraints.gridwidth = GridBagConstraints.REMAINDER;
74 addComponent( list );
75
76 setSize( 300, 200 );
77 setVisible( true );
78
79 } // end constructor
80

```

Place button[2] right next to button[1]

Specify comboBox as last (only) component in third row

Specify buttons[3] as last (only) component in fourth row

Place button[4] as first component in fifth row

Specify list as last component in fifth row

```
81 // add a Component to the container
82 private void addComponent( Component component )
83 {
84     layout.setConstraints( component, constraints );
85     container.add( component ); // add component
86 }
87
88 public static void main( String args[] )
89 {
90     GridBagDemo2 application = new GridBagDemo2();
91     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
92 }
93
94 } // end class GridBagDemo2
```