22

Componentes GUI: Parte 2

OBJETIVOS

- Neste capítulo, você aprenderá:
- Como criar e manipular controles deslizantes, menus, menus pop-up e janelas.
- Como alterar a aparência e o comportamento de uma GUI utilizando a aparência e o comportamento plugáveis do Swing.
- Como criar uma interface de múltiplos documentos com `JdesktopPane` e `JInternalFrame`.
- Como utilizar gerenciadores adicionais de layout.

Sumário

22.1 Introdução

- **Aparência e funcionamento plugáveis (*pluggable look-and-feel* — PLAF).**
  - **O Swing pode personalizar a aparência e o comportamento da GUI.**
  - **Motif:**
    - **Aparência e comportamento populares do UNIX.**
- **Interface de múltiplos documentos (*multiple document interface* — MDI).**
  - **Uma janela principal (a janela-pai) contendo outras janelas (janelas-filhas).**
  - **Gerencia vários documentos abertos em paralelo.**

22.2 Jslider

- Jslider:
  - Permite ao usuário selecionar a partir de um intervalo de valores inteiros.
  - Herda de Jcomponent.
  - Contém:
    - Marcas de medida:
      - Pode exibir marcas de medida principais, marcas de medida secundárias e rótulos para marcas de medida.
      - Não são exibidas por padrão.
    - Marcador:
      - Permite que o usuário selecione um valor.
  - Aderência às marcas:
    - Faz o marcador ser atraído para a marca de medida mais próxima.

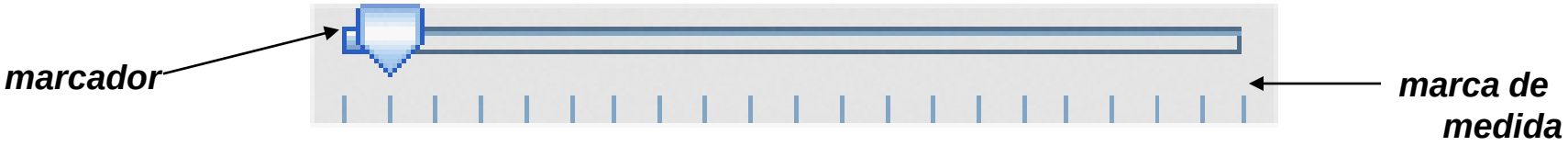


Figura 22.1 | Componente `JSlider` com orientação horizontal.

22.2 `JSlider` (*Continuação*)

- **Se o foco estiver em um `JSlider` (é o componente GUI atualmente selecionado na interface com o usuário).**
  - **As teclas de seta para a esquerda/direita fazem com que o marcador do `JSlider` diminua/aumente por 1.**
  - **As teclas de seta para baixo/para cima fazem com que o marcador do `JSlider` diminua/aumente por 1.**
  - **A tecla *PgDn* (page down) e a tecla *PgUp* (page up) fazem com que o marcador do `JSlider` diminua ou aumente por incrementos de bloco de um décimo da faixa de valores, respectivamente.**
  - **As teclas *End*/*Home* movem o marcador do `JSlider` para o valor mínimo/máximo do `Jslider`.**

22.2 `Jslider` (*Continuação*)

- **Pode ter orientação horizontal ou vertical:**
  - **O valor mínimo está à esquerda ou na parte inferior do `Jslider`.**
  - **Valor máximo está à direita ou na parte superior do `Jslider`.**
  - **O método `JSlider` `setInverted` inverte as posições dos valores mínimos e máximos.**
- **Gera `ChangeEvents` em resposta a interações de usuário:**
  - **Um objeto de uma classe que implementa a interface `ChangeListener` e declara o método `stateChanged` para responder a `ChangeEvents`.**

Observação sobre aparência e comportamento 22.1

Se um novo componente GUI tiver uma largura e altura mínimas (isto é, se as pequenas dimensões resultarem em um componente ineficiente na tela), sobrescreva o método `getMinimumSize` para retornar a largura e a altura mínimas como uma instância da classe `Dimension`.

Observação de engenharia de software 22.1

Para muitos componentes GUI, o método `getMinimumSize` é implementado para retornar o resultado de uma chamada ao método `getPreferredSize` desse componente.

Resumo

OvalPanel.java

(1 de 2)

```
1  // Fig. 22.2: OvalPanel.java
2  // Uma classe personalizada de JPanel.
3  import java.awt.Graphics;
4  import java.awt.Dimension;
5  import javax.swing.JPanel;
6
7  public class OvalPanel extends JPanel
8  {
9     private int diameter = 10; // diâmetro padrão de 10
10
11    // desenha uma oval do diâmetro especificado
12    public void paintComponent( Graphics g )
13    {
14       super.paintComponent( g );
15
16       g.fillOval( 10, 10, diameter, diameter ); // desenha um círculo
17    } // fim do método paintComponent
18
19    // valida e configura o diâmetro, então pinta novamente
20    public void setDiameter( int newDiameter )
21    {
22       // se o diâmetro for inválido, assume o padrão de 10
23       diameter = ( newDiameter >= 0 ? newDiameter : 10 );
24       repaint(); // pinta o painel novamente
25    } // fim do método setDiameter
26
```

Utilizado como a largura e altura do quadro delimitador em que o círculo é exibido

Desenha um círculo preenchido.

Altera o **`diameter`** do círculo e o redesenha (**`repaint`**)

Resumo

```
27    // utilizado pelo gerenciador de layout para determinar o tamanho preferido
28    public Dimension getPreferredSize()
29    {
30       return new Dimension( 200, 200 );
31    } // fim do método getPreferredSize
32
33    // utilizado pelo gerenciador de layout para determinar o tamanho mínimo
34    public Dimension getMinimumSize()
35    {
36       return getPreferredSize();
37    } // fim do método getMinimumSize
38 } // fim da classe OvalPanel
```

Retorna a largura e altura preferidas de um **OvalPanel**

Retorna a largura e altura mínimas de um **OvalPanel**

OvalPanel.java

(2 de 2)

Resumo

SliderFrame.java

(1 de 2)

```
1  // Fig. 22.3: SliderFrame.java
2  // Utilizando JSliders para dimensionar uma oval.
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import javax.swing.JFrame;
6  import javax.swing.JSlider;
7  import javax.swing.SwingConstants;
8  import javax.swing.event.ChangeListener;
9  import javax.swing.event.ChangeEvent;
10
11 public class SliderFrame extends JFrame
12 {
13    private JSlider diameterJSlider; // controle deslizante p/ selecionar o diâmetro
14    private OvalPanel myPanel; // painel para desenhar o círculo
15
16    // construtor sem argumentos
17    public SliderFrame()
18    {
19       super( "Slider Demo" );
20
21       myPanel = new OvalPanel(); // cria painel para desenhar o círculo
22       myPanel.setBackground( Color.YELLOW ); // configura o fundo como amarelo
23
24       // configura o JSlider para controlar o valor do diâmetro
25       diameterJSlider =
26          new JSlider( SwingConstants.HORIZONTAL, 0, 200, 10 );
27       diameterJSlider.setMajorTickSpacing( 10 ); // cria uma marca de medi
28       diameterJSlider.setPaintTicks( true ); // pinta marcas no controle deslizante
29
```

Cria objeto **OvalPanel myPanel**

Cria um objeto **JSlider diameterSlider** como um **JSlider** horizontal com um intervalo entre **0-200** e um valor inicial de **10**

Indica que cada marca de medida principal representa **10** valores e que as marcas de medida devem ser exibidas

Resumo

SliderFrame.java

```
30       // registra o ouvinte de evento do JSlider
31       diameterJSlider.addChangeListener(
32
33          new ChangeListener() // classe interna anônima
34          {
35             // trata da alteração de valor do controle deslizante
36             public void stateChanged( ChangeEvent e )
37             {
38                myPanel.setDiameter( diameterJSlider.getValue() );
39             } // fim do método stateChanged
40          } // fim da classe interna anônima
41       ); // fim da chamada para addChangeListener
42
43       add( diameterJSlider, BorderLayout.SOUTH ); // adiciona
44       add( myPanel, BorderLayout.CENTER ); // adicion
45    } // fim do construtor SliderFrame
46 } // fim da classe SliderFrame
```

Registra um **ChangeListener** para tratar eventos do **diameterSlider**

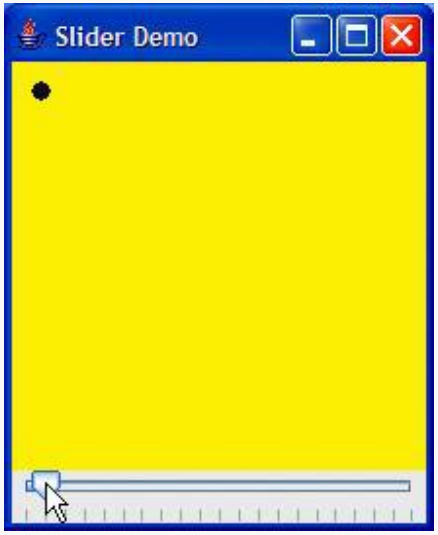
O método **stateChanged** é chamado em resposta a uma interação de usuário

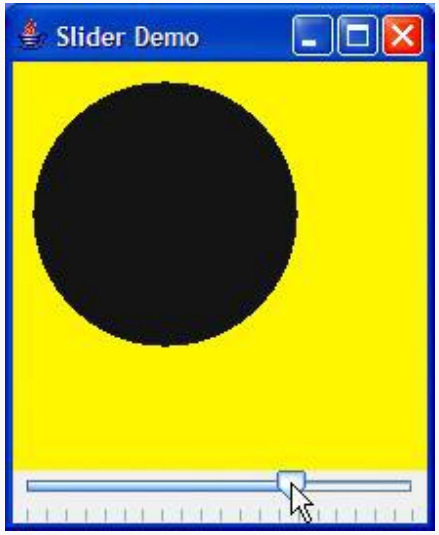
Chama o método **myPanel setDiameter** e passa o valor atual da posição do marcador retornado pelo método **getValue JSlider**

Resumo

SliderDemo.java

```
1  // Fig. 22.4: SliderDemo.java
2  // Testando SliderFrame.
3  import javax.swing.JFrame;
4
5  public class SliderDemo
6  {
7     public static void main( String args[] )
8     {
9        SliderFrame sliderFrame = new SliderFrame();
10       sliderFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       sliderFrame.setSize( 220, 270 ); // configura o tamanho do frame
12       sliderFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe SliderDemo
```





22.3 Windows: Notas adicionais

- `JFrame`:
  - **É uma *janela* com uma barra de título e uma borda.**
  - **Uma subclasse do `java.awt.Frame`.**
    - **Que é uma subclasse do `java.awt.Window`.**
  - **Um dos poucos componentes GUI Swing que não é um componente GUI leve.**
  - **As janelas de aplicação Java são parecidas com qualquer outra janela exibida nessa plataforma.**

Boa prática de programação 22.1

Janelas são um recurso caro ao sistema. Retorne-as ao sistema quando elas não forem mais necessárias.

22.3 Windows: Notas adicionais (*Continuação*)

- O método `JFrame` `setDefaultCloseOperation` determina o que acontece quando o usuário fecha a janela.
  - `DISPOSE_ON_CLOSE`
    - Fecha a `Window` para retornar recursos ao sistema.
  - `DO_NOTHING_ON_CLOSE`
    - Indica que o programa determinará o que fazer quando o usuário indica que a janela deve ser fechada.
  - `HIDE_ON_CLOSE`
    - O padrão.
- Método `JFrame` `setVisible`
  - Exibe a janela na tela.
- Método `JFrame` `setLocation`
  - Especifica a posição da janela quando ela aparece na tela.

Erro comum de programação 22.1

Esquecer de chamar o método `setVisible` em uma janela é um erro de lógica em tempo de execução — a janela não é exibida.

Erro comum de programação 22.2

Esquecer de chamar o método `setSize` em uma janela é um erro de lógica em tempo de execução — somente a barra de título aparece.

22.3 Windows: Notas adicionais (*Continuação*)

- **Manipulação da janela pelo usuário gera eventos de janela:**
  - **O método `addWindowListener` registra ouvintes de evento para eventos de janela.**
  - **A interface `WindowListener` especifica sete métodos de tratamento de evento de janela:**
    - `windowActivated` **— chamado quando o usuário torna uma janela a janela principal.**
    - `windowClosed` **— chamado depois que a janela é fechada.**
    - `windowClosing` **— chamado quando o usuário inicia o fechamento da janela.**

22.3 Windows: Notas adicionais (*Continuação*)

- `windowDeactivated` — chamado quando o usuário torna uma outra janela a janela principal.
- `windowDeiconified` — chamado quando o usuário restaura uma janela depois de ser minimizada.
- `windowIconified` — chamado quando o usuário minimiza uma janela.
- `windowOpened` — chamado quando um programa exibe uma janela na tela pela primeira vez.

22.4 Utilizando menus com frames

- **Menus:**
  - **Permitem que o usuário realize ações sem poluir desnecessariamente uma GUI com componentes extras.**
  - **Só podem ser anexados a objetos das classes que fornecem membros `setMenuBar`, como `JFrame` e `Japplet`.**
  - **Classe `MenuBar`:**
    - **Contém os métodos necessários para gerenciar uma barra de menus.**
  - **Classe `Jmenu`:**
    - **Contém os métodos necessários para gerenciar menus.**
  - **Classe `JMenuItem`:**
    - **Contém os métodos necessários para gerenciar itens de menu.**
      - **Pode ser utilizado para iniciar uma ação ou pode ser um submenu.**

Observação sobre aparência e comportamento 22.2

Os menus simplificam as GUIs porque os componentes podem ser ocultos dentro deles. Esses componentes só serão visíveis quando o usuário procurá-los, selecionando-os no menu.

22.4 Utilizando menus com frames (*Continuação*)

- Classe `JCheckBoxMenuItem`:
  - Contém os métodos necessários para gerenciar itens de menu que podem ser ativados ou desativados.
- Classe `JRadioButtonMenuItem`.
  - Contém os métodos necessários para gerenciar itens de menu que podem ser ativados ou desativados como `JCheckBoxMenuItem`s.
  - Quando múltiplos `JRadioButtonMenuItem`s são mantidos como parte de um `ButtonGroup`, apenas um item no grupo pode ser selecionado de cada vez.
- Mnemônicos:
  - Caracteres especiais que fornecem acesso rápido a um menu ou item do menu a partir do teclado.

Resumo

MenuFrame.java

(1 de 8)

```
1  // Fig. 22.5: MenuFrame.java
2  // Demonstrando menus.
3  import java.awt.Color;
4  import java.awt.Font;
5  import java.awt.BorderLayout;
6  import java.awt.event.ActionListener;
7  import java.awt.event.ActionEvent;
8  import java.awt.event.ItemListener;
9  import java.awt.event.ItemEvent;
10 import javax.swing.JFrame;
11 import javax.swing.JRadioButtonMenuItem;
12 import javax.swing.JCheckBoxMenuItem;
13 import javax.swing.JOptionPane;
14 import javax.swing.JLabel;
15 import javax.swing.SwingConstants;
16 import javax.swing.ButtonGroup;
17 import javax.swing.JMenu;
18 import javax.swing.JMenuItem;
19 import javax.swing.JMenuBar;
20
```

Resumo

MenuFrame.java

(2 de 8)

```
21 public class MenuFrame extends JFrame
22 {
23    private final Color colorValues[] =
24       { Color.BLACK, Color.BLUE, Color.RED, Color.GREEN };
25    private JRadioButtonMenuItem colorItems[]; // itens do menu Color
26    private JRadioButtonMenuItem fonts[]; // itens do menu Font
27    private JCheckBoxMenuItem styleItems[]; // itens do menu Font Style
28    private JLabel displayJLabel; // exibe o texto de exemplo
29    private ButtonGroup fontButtonGroup; // gerencia itens do menu Font
30    private ButtonGroup colorButtonGroup; // gerencia itens do menu Color
31    private int style; // utilizado para criar estilo para fonte
32
33    // construtor sem argumentos configura a GUI
34    public MenuFrame()
35    {
36       super( "Using JMenus" );
37
38       JMenu fileMenu = new JMenu( "File" ); // cria o menu File
39       fileMenu.setMnemonic( 'F' ); // configura o mnemônico como F
40
41       // create About... menu item
42       JMenuItem aboutItem = new JMenuItem( "About..." );
43       aboutItem.setMnemonic( 'A' ); // configura o mnemônico com
44       fileMenu.add( aboutItem ); // adiciona o item about ao men
45       aboutItem.addActionListener(
46
```

Cria um **JMenu**

Chama o método **JMenu setMnemonic**

Adiciona o **JMenuItem** ‘**About…**’ a **fileMenu**

```
47          new ActionListener() // classe interna anônima
48          {
49             // exibe um diálogo de mensagem quando o usuário sel
50             public void actionPerformed( ActionEvent event )
51             {
52                JOptionPane.showMessageDialog( MenuFrame.this,
53                   "This is an example\nof using menus",
54                   "About", JOptionPane.PLAIN_MESSAGE );
55             } // fim do método actionPerformed
56          } // fim da classe interna anônima
57       ); // fim da chamada para addActionListener
58
59       JMenuItem exitItem = new JMenuItem( "Exit" ); // cria o item exit
60       exitItem.setMnemonic( 'x' ); // configura o mnemônico como x
61       fileMenu.add( exitItem ); // adiciona o item exit ao menu File
62       exitItem.addActionListener(
63
64          new ActionListener() // classe interna anônima
65          {
66             // termina o aplicativo quando o usuário clicar em exitItem
67             public void actionPerformed( ActionEvent event )
68             {
69                System.exit( 0 ); // encerra o aplicativo
70             } // fim do método actionPerformed
71          } // fim da classe interna anônima
72       ); // fim da chamada para addActionListener
73
```

Cria um **ActionListener** para processar o evento de ação de **aboutItem**

Exibe uma caixa de diálogo de mensagem.

Cria e adiciona o item do menu **exitItem**

Registra um **ActionListener** que termina a aplicação

```
74      JMenuBar bar = new JMenuBar(); // cria a barra de menu
75      setJMenuBar( bar ); // adiciona a barra de menu ao aplicat
76      bar.add( fileMenu ); // adiciona o menu File à barra de me
77
78      JMenu formatMenu = new JMenu( "Format" ); // cria o menu Format
79      formatMenu.setMnemonic( 'r' ); // configura mnemônico como r
80
81      // array listando cores de string
82      String colors[] = { "Black", "Blue", "Red", "Green" };
83
84      JMenu colorMenu = new JMenu( "Color" ); // cria o menu Color
85      colorMenu.setMnemonic( 'C' ); // configura mnemônico c
86
87      // cria itens do menu Color com botões de opção
88      colorItems = new JRadioButtonMenuItem[ colors.length ];
89      colorButtonGroup = new ButtonGroup(); // gerencia cores
90      ItemHandler itemHandler = new ItemHandler(); // handler para
91
92      // cria itens do menu Color com botões de opção
93      for ( int count = 0; count < colors.length; count++ )
94      {
95         colorItems[ count ] =
96            new JRadioButtonMenuItem( colors[ count ] ); // cria o
97         colorMenu.add( colorItems[ count ] ); // adiciona o item a
98         colorButtonGroup.add( colorItems[ count ] ); // adiciona ao grupo
99         colorItems[ count ].addActionListener( itemHandler );
100     } // fim de for
101
```

Adiciona **fileMenu** a um **JMenuBar** e anexa o **JMenuBar** à janela da aplicação

MenuFrame.java

Cria um menu **formatMenu**

Cria um submenu **colorMenu**

Cria um array **colorItems** para **JRadioButtonMenuItem**

Cria um **ButtonGroup** a fim de assegurar que somente um dos itens de menu seja selecionado em determinado momento

Adiciona **JRadioButtonMenuItem**s a **colorMenu** e registra **ActionListener**s

```
102    colorItems[ 0 ].setSelected( true ); // seleciona primeiro item Color
103
104    formatMenu.add( colorMenu ); // adiciona o menu Color ao menu F
105    formatMenu.addSeparator(); // adiciona o separador no menu
106
107    // array listing font names
108    String fontNames[] = { "Serif", "Monospaced", "SansSerif" };
109    JMenu fontMenu = new JMenu( "Font" ); // cria o menu Font
110    fontMenu.setMnemonic( 'n' ); // configura o mnemônico como n
111
112    // cria itens do menu radiobutton para nomes de fonte
113    fonts = new JRadioButtonMenuItem[ fontNames.length ];
114    fontButtonGroup = new ButtonGroup(); // gerencia nomes de fo
115
116    // cria itens do menu Font com botões de opção
117    for ( int count = 0; count < fonts.length; count++ )
118    {
119       fonts[ count ] = new JRadioButtonMenuItem( fontNames[ count
120       fontMenu.add( fonts[ count ] ); // adiciona fonte ao menu Fo
121       fontButtonGroup.add( fonts[ count ] ); // adiciona ao grupo
122       fonts[ count ].addActionListener( itemHandler ); // adiciona handler
123    } // fim do for
124
125    fonts[ 0 ].setSelected( true ); // seleciona primeiro item de menu Font
126    fontMenu.addSeparator(); // adiciona barra separador ao menu Font
127
```

no

MenuFrame.java

Invoca o método **AbstractButton setSelected**

Adiciona **colorMenu** ao **formatMenu** e adiciona uma linha separadora horizontal

Cria um array **JRadioButtonMenuItem fonts**

Cria um **ButtonGroup** a fim de assegurar que somente um dos itens de menu seja selecionado em determinado momento

Adiciona **JRadioButtonMenuItem**s a **colorMenu** e registra **ActionListener**s

Configura a seleção-padrão e adiciona um separador horizontal

Resumo

MenuFrame.java

```
128     String styleNames[] = { "Bold", "Italic" }; // nomes de estilos
129     styleItems = new JCheckBoxMenuItem[ styleNames.length ];
130     StyleHandler styleHandler = new StyleHandler(); // handler de estilos
131
132     // criar itens do menu Style com caixas de seleção
133     for ( int count = 0; count < styleNames.length; count++ )
134     {
135        styleItems[ count ] =
136           new JCheckBoxMenuItem( styleNames[ count ] ); // para estilo
137        fontMenu.add( styleItems[ count ] ); // adicina ao menu Font
138        styleItems[ count ].addItemListener( styleHandler ); // handler
139     } // fim do for
140
141     formatMenu.add( fontMenu ); // adiciona menu Font ao m
142     bar.add( formatMenu ); // adiciona menu Format à barra de menu
143
144     // set up label to display text
145     displayJLabel = new JLabel( "Sample Text", SwingConstants.CENTER );
146     displayJLabel.setForeground( colorValues[ 0 ] );
147     displayJLabel.setFont( new Font( "Serif", Font.PLAIN, 72 ) );
148
149     getContentPane().setBackground( Color.CYAN ); // configura o fundo
150     add( displayJLabel, BorderLayout.CENTER ); // adiciona displayJLabel
151  } // fim de construtor MenuFrame
152
```

Cria **JCheckBoxMenuItem**s

Adiciona **fontMenu** a **formatMenu** e **formatMenu** a **JMenuBar**

Resumo

MenuFrame.java

```
153 // classe interna para tratar eventos de ação dos itens de menu
154 private class ItemHandler implements ActionListener
155 {
156    // processa seleções de cor e fonte
157    public void actionPerformed( ActionEvent event )
158    {
159       // processa seleções de cor
160       for ( int count = 0; count < colorItems.length; count++ )
161       {
162          if ( colorItems[ count ].isSelected() )
163          {
164             displayJLabel.setForeground( colorValues[ count ] );
165             break;
166          } // fim do if
167       } // fim do for
168
169       // processa seleção de fonte
170       for ( int count = 0; count < fonts.length; count++ )
171       {
172          if ( event.getSource() == fonts[ count ] )
173          {
174             displayJLabel.setFont(
175                new Font( fonts[ count ].getText(), style,
176          } // fim do if
177       } // fim do for
178
```

Determina o **`JRadioButtonMenuItem`** selecionado

O método **`getSource`** retorna uma referência ao **`JRadioButtonMenuItem`** que gerou o evento

Resumo

MenuFrame.java

```
179         repaint(); // desenha novamente o aplicativo
180      } // fim do método actionPerformed
181   } // fim da classe ItemHandler
182
183   // classe interna para tratar eventos dos itens de menu com caixa de seleção
184   private class StyleHandler implements ItemListener
185   {
186      // processa seleções de estilo da fonte
187      public void itemStateChanged( ItemEvent e )
188      {
189         style = 0; // inicializa estilo
190
191         // verifica seleção de negrito
192         if ( styleItems[ 0 ].isSelected() )
193            style += Font.BOLD; // adiciona negrito ao estilo
194
195         // verifica seleção de itálico
196         if ( styleItems[ 1 ].isSelected() )
197            style += Font.ITALIC; // adiciona itálico ao estilo
198
199         displayJLabel.setFont(
200            new Font( displayJLabel.getFont().getName(), style, 72 ) );
201         repaint(); // desenha novamente o aplicativo
202      } // fim do método itemStateChanged
203   } // fim da classe StyleHandler
204} // fim da classe MenuFrame
```

Chamado se o usuário selecionar um **`JCheckBoxMenuItem`** no **`fontMenu`**

Determina se um ou ambos os **`JCheckBoxMenuItem`**s estão selecionados

Observação sobre aparência e comportamento 22.3

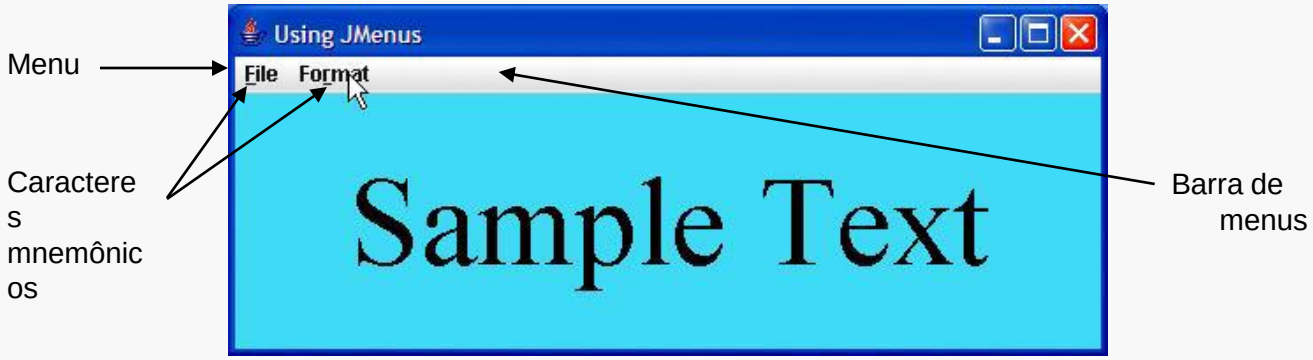
Os mnemônicos fornecem acesso rápido a comandos de menu e comandos de botão pelo teclado.

Resumo

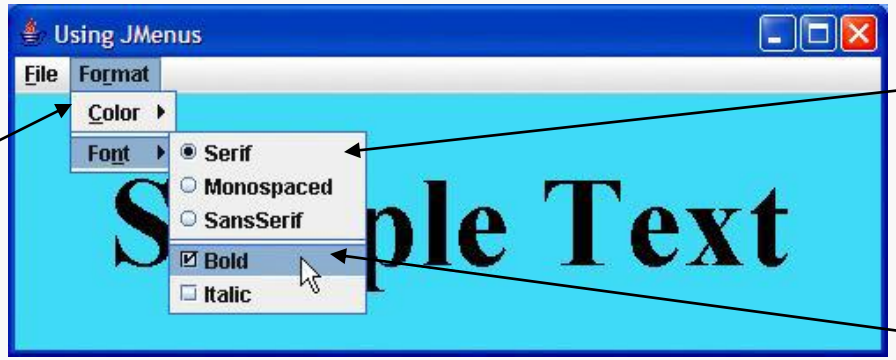
MenuTest.java

(1 de 2)

```
1  // Fig. 22.6: MenuTest.java
2  // Testando MenuFrame.
3  import javax.swing.JFrame;
4
5  public class MenuTest
6  {
7     public static void main( String args[] )
8     {
9        MenuFrame menuFrame = new MenuFrame(); // cria MenuFrame
10       menuFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       menuFrame.setSize( 500, 200 ); // configura o tamanho do frame
12       menuFrame.setVisible( true ); // exibe o frame
13    } // fim do main
14 } // fim da classe MenuTest
```



Resumo



`MenuTest.java`

(2 de 2)

Observação sobre aparência e comportamento 22.4

Diferentes mnemônicos devem ser utilizados para cada botão ou item de menu. Normalmente, a primeira letra do rótulo no item de menu ou botão é utilizada como o mnemônico. Se diversos botões ou itens de menu iniciam com a mesma letra, escolha a próxima letra mais significativa no nome (por exemplo, `X` comumente é escolhida para um botão ou item de menu chamado `Exit`).

22.4 Utilizando menus com frames (*Continuação*)

- **Método `showMessageDialog`:**
  - **Especificar a janela-pai ajuda a determinar onde a caixa de diálogo será exibida:**
    - **Se especificada como `null`, a caixa de diálogo aparece no centro da tela.**
    - **Caso contrário, ela aparece centralizada na janela-pai especificada.**
  - **Caixa de diálogo modal:**
    - **Não permite que nenhuma outra janela na aplicação seja acessada até que a caixa de diálogo seja fechada.**
    - **Em geral, caixas de diálogo são modais.**

Erro comum de programação 22.3

Esquecer de configurar a barra de menus com o método `JFrame` `setJMenuBar` resulta na barra de menus não sendo exibido no `JFrame`.

Observação sobre aparência e comportamento 22.5

Os menus aparecem da esquerda para a direita na ordem em que eles são adicionados a um `JMenuBar`.

Observação sobre aparência e comportamento 22.6

Um submenu é criado adicionando-se um menu como um item de menu a um outro menu. Quando o mouse é posicionado sobre um submenu (ou o mnemônico do submenu é pressionado), o submenu expande para mostrar seus itens de menu.

Observação sobre aparência e comportamento 22.7

Separadores podem ser adicionados a um menu para agrupar itens de menu logicamente.

Observação sobre aparência e comportamento 22.8

Qualquer componente GUI 'leve' (isto é, um componente que é uma subclasse de `JComponent`) pode ser adicionado a um `JMenu` ou a um `JMenuBar`.

22.5 JPopupMenu

- **Menu pop-up sensível ao contexto:**
  - **Fornece opções específicas do componente para o qual o evento de gatilho pop-up foi gerado.**
    - **Na maioria dos sistemas, o evento de acionamento de pop-up ocorre quando o usuário pressiona e libera o botão direito do mouse.**
  - **Criado com a classe `JPopupMenu`.**

Observação sobre aparência e comportamento 22.9

O evento de acionamento do pop-up é específico da plataforma. Na maioria das plataformas que utilizam um mouse com múltiplos botões, o evento de acionamento do pop-up ocorre quando o usuário clica com o botão direito do mouse em um componente que suporta um menu pop-up.

Resumo

PopupFrame.java

(1 de 4)

```
1  // Fig. 22.7: PopupFrame.java
2  // Demonstrando JPopupMenus.
3  import java.awt.Color;
4  import java.awt.event.MouseAdapter;
5  import java.awt.event.MouseEvent;
6  import java.awt.event.ActionListener;
7  import java.awt.event.ActionEvent;
8  import javax.swing.JFrame;
9  import javax.swing.JRadioButtonMenuItem;
10 import javax.swing.JPopupMenu;
11 import javax.swing.ButtonGroup;
12
13 public class PopupFrame extends JFrame
14 {
15    private JRadioButtonMenuItem items[]; // armazena itens para cores
16    private final Color colorValues[] =
17       { Color.BLUE, Color.YELLOW, Color.RED }; // cores a serem utilizadas
18    private JPopupMenu popupMenu; // permite ao usuário selecionar cores
19
20    // construtor sem argumentos configura a GUI
21    public PopupFrame()
22    {
23       super( "Using JPopupMenus" );
24
25       ItemHandler handler = new ItemHandler(); // handler para itens de menu
26       String colors[] = { "Blue", "Yellow", "Red" }; // array de cores
27
```

Uma instância da classe **`ItemHandler`** processará os eventos de item nos itens de menu

Resumo

PopupFrame.java

(2 de 4)

```
28       ButtonGroup colorGroup = new ButtonGroup(); // gerencia itens de cores
29       popupMenu = new JPopupMenu(); // cria menu pop-up
30       items = new JRadioButtonMenuItem[ 3 ]; // itens para selecionar cor
31
32       // constrói item de menu, adiciona menu popup, ativa tratamento de event
33       for ( int count = 0; count < items.length; count++ )
34       {
35          items[ count ] = new JRadioButtonMenuItem( colors[ count ] );
36          popupMenu.add( items[ count ] ); // adiciona item ao menum pop-up
37          colorGroup.add( items[ count ] ); // adiciona item ao grupo de botão
38          items[ count ].addActionListener( handler ); // adiciona handler
39       } // fim do for
40
41       setBackground( Color.WHITE ); // configura fundo como branco
42
43       // declara um MouseListener para a janela exibir menu pop
44       addMouseListener(
45
46          new MouseAdapter() // classe interna anônima
47          {
48             // trata evento de pressionamento de mouse
49             public void mousePressed( MouseEvent event )
50             {
51                checkForTriggerEvent( event ); // verifica o acionamento
52             } // fim do método mousePressed
53
```

Cria um objeto **JPopupMenu**

Cria e adiciona **JRadioButtonMenuItem** e registra **ActionListener**s

Registra um **MouseListener** para tratar os eventos de mouse da janela da aplicação

Resumo

PopupFrame.java

(3 de 4)

```
54          // trata eventos de liberação de botão do mouse
55          public void mouseReleased( MouseEvent event )
56          {
57             checkForTriggerEvent( event ); // verifica acionamento
58          } // fim do método mouseReleased
59
60          // determina se o evento deve acionar menu popup
61          private void checkForTriggerEvent( MouseEvent event )
62          {
63             if ( event.isPopupTrigger() )
64                popupMenu.show(
65                   event.getComponent(), event
66          } // fim do método checkForTriggerEvent
67       } // fim da classe interna anônima
68    ); // fim da chamada para addMouseList
69 } // fim do construtor PopupFrame
70
```

Se o evento de gatilho pop-up tiver ocorrido, o método **`JPopupMenu show`** exibe o **`JPopupMenu`**

O componente de origem e os argumentos de coordenadas determinam onde o **`JPopupMenu`** aparecerá

Resumo

PopupFrame.java

(4 de 4)

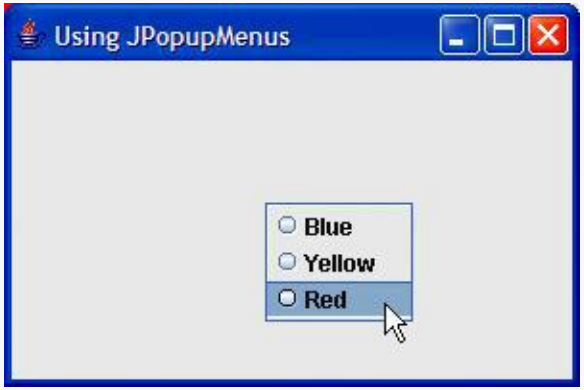
```
71    // classe interna privada para tratar eventos de item de menu
72    private class ItemHandler implements ActionListener
73    {
74       // processa seleções de item de menu
75       public void actionPerformed( ActionEvent event )
76       {
77          // determina qual item de menu foi selecionado
78          for ( int i = 0; i < items.length; i++ )
79          {
80             if ( event.getSource() == items[ i ] )
81             {
82                getContentPane().setBackground( colorValues[ i ] );
83                return;
84             } // fim do if
85          } // fim do for
86       } // fim do método actionPerformed
87    } // fim da classe interna privada ItemHand
88 } // fim da classe PopupFrame
```

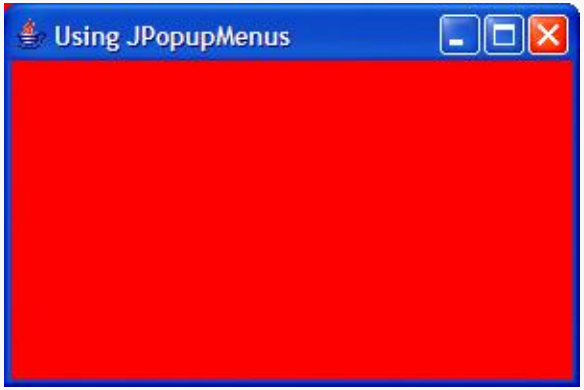
Determina qual **`JRadioButtonMenuItem`** o usuário selecionou e configura a cor de segundo plano

Resumo

PopupTest.java

```
1  // Fig. 22.8: PopupTest.java
2  // Testando PopupFrame.
3  import javax.swing.JFrame;
4
5  public class PopupTest
6  {
7     public static void main( String args[] )
8     {
9        PopupFrame popupFrame = new PopupFrame(); // cria PopupFrame
10       popupFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       popupFrame.setSize( 300, 200 ); // configura o tamanho do frame
12       popupFrame.setVisible( true ); // exibe o frame
13    } // fim do main
14 } // fim da classe PopupTest
```





Observação sobre aparência e comportamento 22.10

Exibir um `JPopupMenu` para o evento de acionamento de pop-up de múltiplos componentes GUI requer o registro de handlers de eventos de mouse para cada um desses componentes GUI.

22.6 Aparência e comportamento plugáveis

- **Aparências das aplicações Java:**
  - **Um programa que utiliza componentes GUI do Abstract Window Toolkit do Java assume a aparência e o comportamento da plataforma:**
    - **Permite aos usuários da aplicação em cada plataforma utilizarem componentes GUI que eles já conhecem.**
    - **Também introduz questões interessantes de portabilidade.**
  - **Os componentes GUI leves do Swing fornecem funcionalidades uniformes:**
    - **Definem uma aparência e um comportamento uniformes entre diferentes plataformas (conhecido como aparência e comportamento metal).**
    - **Também podem personalizar a aparência e o comportamento como um estilo do Windows Microsoft, como um estilo Motif (UNIX) ou uma aparência e comportamento do Macintosh.**

Dica de portabilidade 22.1

Os componentes GUI têm uma aparência diferente em diferentes plataformas e podem exigir quantidades diferentes de espaço para serem exibidos. Isso poderia alterar seus layouts e alinhamentos da GUI.

Dica de portabilidade 22.2

Os componentes GUI em diferentes plataformas têm diferentes funcionalidades-padrão (por exemplo, algumas plataformas permitem que um botão com o foco seja ‘pressionado’ com a barra de espaço e outras, não).

Resumo

LookAndFeelFrame.java

(1 de 4)

```
1  // Fig. 22.9: LookAndFeelFrame.java
2  // Alterando a aparência e comportamento.
3  import java.awt.GridLayout;
4  import java.awt.BorderLayout;
5  import java.awt.event.ItemListener;
6  import java.awt.event.ItemEvent;
7  import javax.swing.JFrame;
8  import javax.swing.UIManager;
9  import javax.swing.JRadioButton;
10 import javax.swing.ButtonGroup;
11 import javax.swing.JButton;
12 import javax.swing.JLabel;
13 import javax.swing.JComboBox;
14 import javax.swing.JPanel;
15 import javax.swing.SwingConstants;
16 import javax.swing.SwingUtilities;
17
18 public class LookAndFeelFrame extends JFrame
19 {
20    // nomes de string das aparências e comportamentos
21    private final String strings[] = { "Metal", "Motif", "Windows" };
22    private UIManager.LookAndFeelInfo looks[]; // aparência e comportamentos
23    private JRadioButton radio[]; // botões de opção para selecionar a aparência e comportamento
24    private ButtonGroup group; // grupo para botões de opção
25    private JButton button; // exibe a aparência do botão
26    private JLabel label; // exibe a aparência do rótulo
27    private JComboBox comboBox; // exibe a aparência da caixa de combinação
28
```

Resumo

LookAndFeelFrame.java

(2 de 4)

```
29    // configura a GUI
30    public LookAndFeelFrame()
31    {
32       super( "Look and Feel Demo" );
33
34       JPanel northPanel = new JPanel(); // cria o painel North
35       northPanel.setLayout( new GridLayout( 3, 1, 0, 5 ) );
36
37       label = new JLabel( "This is a Metal look-and-feel",
38          SwingConstants.CENTER ); // cria o rótulo
39       northPanel.add( label ); // adiciona o rótulo ao painel
40
41       button = new JButton( "JButton" ); // cria o botão
42       northPanel.add( button ); // adiciona o botão ao painel
43
44       comboBox = new JComboBox( strings ); // cria a caixa de combinação
45       northPanel.add( comboBox ); // adiciona a caixa de combinação ao painel
46
47       // cria um array para botões de opção
48       radio = new JRadioButton[ strings.length ];
49
50       JPanel southPanel = new JPanel(); // cria o painel South
51       southPanel.setLayout( new GridLayout( 1, radio.length ) );
52
53       group = new ButtonGroup(); // grupo de botões para a aparência e comportamento
54       ItemHandler handler = new ItemHandler(); // handler da aparência e comportamento
55
```

Resumo

LookAndFeelFrame.java

```
56       for ( int count = 0; count < radio.length; count++ )
57       {
58          radio[ count ] = new JRadioButton( strings[ count ] );
59          radio[ count ].addItemListener( handler ); // adiciona handler
60          group.add( radio[ count ] ); // adiciona botões de opção ao grupo
61          southPanel.add( radio[ count ] ); // adiciona botões de opção ao painel
62       } // fim do for
63
64       add( northPanel, BorderLayo
65       add( southPanel, BorderLayo
66
67       // obtém as informações sobre a aparência e comportamento instaladas
68       looks = UIManager.getInstalledLookAndFeels();
69       radio[ 0 ].setSelected( true ); // configura a seleção padrão
70    } // fim do construtor LookAndFeelFrame
71
72    // utiliza UIManager para alterar a aparência e comportamento da GUI
73    private void changeTheLookAndFeel( int value )
74    {
75       try // muda a aparência e comportamento
76       {
77          // configura a aparência e comportamento para esse aplicativo
78          UIManager.setLookAndFeel( looks[ value ].getClassName() );
79
80          // atualiza os componentes nesse aplicativo
81          SwingUtilities.updateComponentTreeUI( this );
82       } // fim do try
```

Obtém o array dos objetos **`UIManager.LookAndFeelInfo`** que descrevem cada aparência e comportamento disponível no seu sistema

Invoca o método **`static setLookAndFeel`** para alterar a aparência e comportamento

Invoca o método **`static updateComponentTreeUI`** para alterar a aparência e comportamento de cada componente GUI anexado à aplicação

Resumo

LookAndFeelFrame.java

(4 de 4)

```
83       catch ( Exception exception )
84       {
85          exception.printStackTrace();
86       } // fim do catch
87    } // fim do método changeTheLookAndFeel
88
89    // classe interna private para tratar eventos de botão de opção
90    private class ItemHandler implements ItemListener
91    {
92       // processa a seleção de aparência e comportamento feita pelo usuário
93       public void itemStateChanged( ItemEvent event )
94       {
95          for ( int count = 0; count < radio.length; count++ )
96          {
97             if ( radio[ count ].isSelected() )
98             {
99                label.setText( String.format( "This is a %s look-and-feel",
100                  strings[ count ] ) );
101            comboBox.setSelectedIndex( count ); // configura o índice da caixa de combinação
102               changeTheLookAndFeel( count ); // muda a aparência e comportamento
103            } // fim do if
104         } // fim do for
105      } // fim do método itemStateChanged
106   } // fim da classe interna private ItemHand
107} // fim da classe LookAndFeelFrame
```

Chama o método utilitário **changeTheLookAndFeel**

Dica de desempenho 22.1

Cada aparência e comportamento são representados por uma classe Java. O método `UIManager getInstalledLookAndFeels` não carrega cada classe. Em vez disso, fornece os nomes das classes de aparência e comportamento disponíveis de modo que uma escolha possa ser feita (presumivelmente, uma vez na inicialização do programa). Isso reduz o overhead de ter de carregar todas as classes de aparência e comportamento mesmo se o programa não utilizar algumas delas.

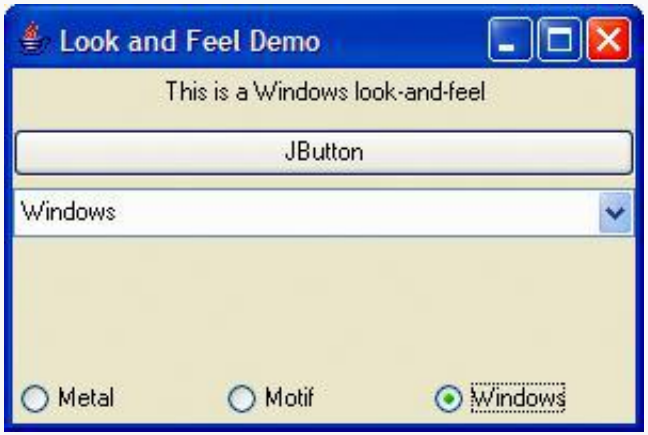
Resumo

LookAndFeelDemo.java

(1 de 2)

```
1  // Fig. 22.10: LookAndFeelDemo.java
2  // Mudando a aparência e o comportamento.
3  import javax.swing.JFrame;
4
5  public class LookAndFeelDemo
6  {
7     public static void main( String args[] )
8     {
9        LookAndFeelFrame lookAndFeelFrame = new LookAndFeelFrame();
10       lookAndFeelFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       lookAndFeelFrame.setSize( 300, 200 ); // configura o tamanho do frame
12       lookAndFeelFrame.setVisible( true ); // exibe o frame
13    } // fim do main
14 } // fim da classe LookAndFeelDemo
```

Resumo



`LookAndFeelDemo.java`

(2 de 2)

22.7 JDesktopPane e JInternalFrame

- **Interface de múltiplos documentos:**
  - **Uma janela principal (chamada janela-pai) contém outras janelas (chamadas janelas-filhas).**
  - **Gerencia vários documentos abertos que estão sendo processados em paralelo.**
  - **Implementada pelo `JDesktopPane` e `JInternalFrame` do Swing.**

Resumo

DeskTopFrame.java

(1 de 4)

```
1  // Fig. 22.11: DesktopFrame.java
2  // Demonstrando JDesktopPane.
3  import java.awt.BorderLayout;
4  import java.awt.Dimension;
5  import java.awt.Graphics;
6  import java.awt.event.ActionListener;
7  import java.awt.event.ActionEvent;
8  import java.util.Random;
9  import javax.swing.JFrame;
10 import javax.swing.JDesktopPane;
11 import javax.swing.JMenuBar;
12 import javax.swing.JMenu;
13 import javax.swing.JMenuItem;
14 import javax.swing.JInternalFrame;
15 import javax.swing.JPanel;
16 import javax.swing.ImageIcon;
17
18 public class DesktopFrame extends JFrame
19 {
20    private JDesktopPane theDesktop;
21
22    // configura a GUI
23    public DesktopFrame()
24    {
25       super( "Using a JDesktopPane" );
26
27       JMenuBar bar = new JMenuBar(); // cria a barra de menu
28       JMenu addMenu = new JMenu( "Add" ); // cria o menu Add
29       JMenuItem newFrame = new JMenuItem( "Internal Frame" );
30
```

Cria um **JMenuBar**, um **JMenu** e um **JMenuItem**

```
31      addMenu.add( newFrame ); // adiciona um novo item de frame ao menu Add
32      bar.add( addMenu ); // adiciona o menu Add à barra de menus
33      setJMenuBar( bar ); // configura a barra de menus
34
35      theDesktop = new JDesktopPane(); // cria o painel
36      add( theDesktop ); // adiciona painel de área de trabalho ao frame
37
38      // configura o ouvinte para o item de menu newF
39      newFrame.addActionListener(
40
41         new ActionListener() // classe interna anôni
42         {
43            // exibe a nova janela interna
44            public void actionPerformed( ActionEvent event )
45            {
46               // cria o frame interno
47               JInternalFrame frame = new JInternalFrame(
48                  "Internal Frame", true, true, true, true );
49
50               MyJPanel panel = new MyJPanel(); // cria um novo painel
51               frame.add( panel, BorderLayout.CENTER ); // adiciona o painel
52               frame.pack(); // configura frame interno de acordo c/ o tamanho do conteúdo
53
```

Adiciona o **JMenuItem** ao **JMenu**, e o **JMenu** ao **JMenuBar**, e configura o **JMenuBar** para a janela da aplicação

DeskTopFrame

O **JDesktopPane** será utilizado para gerenciar as janelas-filhas do **JInternalFrame**

Cria um objeto **JInternalFrame**

Os argumentos de construtor especificam a string da barra de título e se o usuário pode ou não redimensionar, fechar, maximizar e minimizar o frame interno

Configura o tamanho da janela-filha

Resumo

```
54                theDesktop.add( frame ); // anexa frame interno
55                frame.setVisible( true ); // mostra o frame interno
56             } // fim do método actionPerformed
57          } // fim da classe interna anônima
58       ); // fim da chamada para addActionListener
59    } // fim do construtor DesktopFrame
60 } // fim da classe DesktopFrame
61
62 // classe para exibir um ImageIcon em um painel
63 class MyJPanel extends JPanel
64 {
65    private static Random generator = new Random();
66    private ImageIcon picture; // imagem a ser exibida
67    private String[] images = { "yellowflowers.png", "purpleflowers.png",
68       "redflowers.png", "redflowers2.png", "lavenderflowers.png" };
69
70    // carrega a imagem
71    public MyJPanel()
72    {
73       int randomNumber = generator.nextInt( 5 );
74       picture = new ImageIcon( images[ randomNumber ] ); // configura o ícone
75    } // fim do construtor MyJPanel
76
```

Adiciona o **`JInternalFrame`** a **`theDesktop`** e exibe o **`JInternalFrame`**

`.java`

(3 de 4)

Resumo

```
77    // exibe o imageIcon no painel
78    public void paintComponent( Graphics g )
79    {
80       super.paintComponent( g );
81       picture.paintIcon( this, g, 0, 0 ); // exibe o ícone
82    } // fim do método paintComponent
83
84    // retorna dimensões da imagem
85    public Dimension getPreferredSize()
86    {
87       return new Dimension( picture.getIconWidth(),
88          picture.getIconHeight() );
89    } // fim do método getPreferredSize
90 } // fim da classe MyJPanel
```

Especifica o tamanho preferido do painel para uso pelo método **pack**

DeskTopFrame
va

(4 de 4)

Resumo

DeskTopTest.java

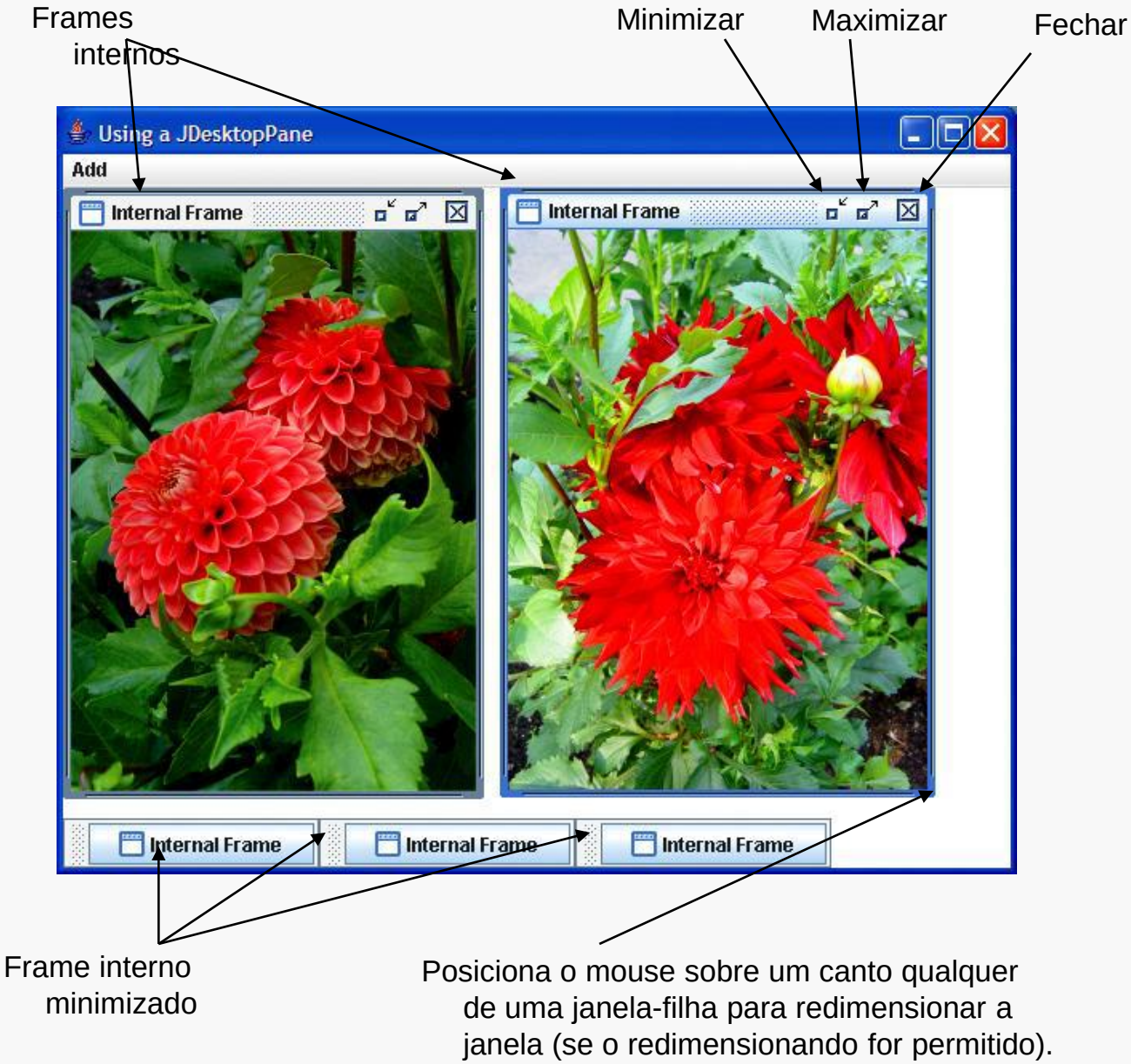
(1 de 3)

```
1  // Fig. 22.12: DesktopTest.java
2  // Demonstrando JDesktopPane.
3  import javax.swing.JFrame;
4
5  public class DesktopTest
6  {
7     public static void main( String args[] )
8     {
9        DesktopFrame desktopFrame = new DesktopFrame();
10       desktopFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       desktopFrame.setSize( 600, 480 ); // configura o tamanho do frame
12       desktopFrame.setVisible( true ); // exibe o frame
13    } // fim do main
14 } // fim da classe DesktopTest
```

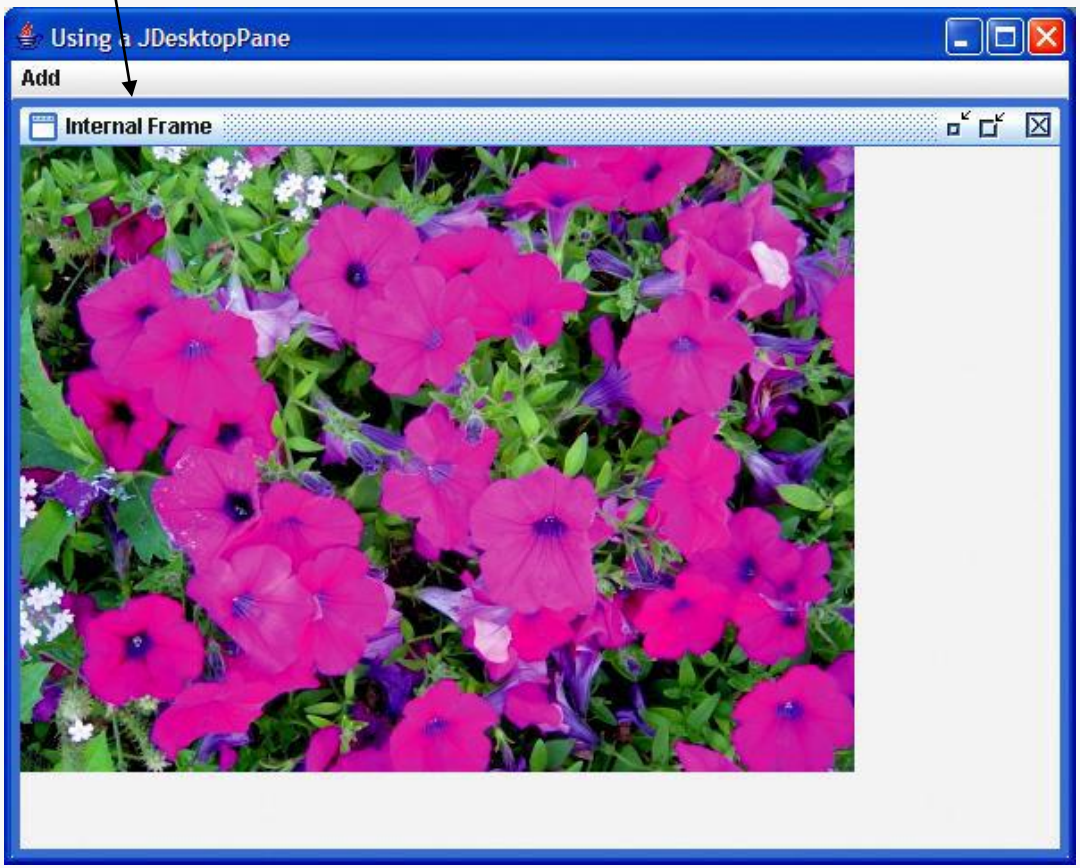
Resumo

DeskTopTest.java

(2 de 3)



Frame interno maximizado



Resumo

`DeskTopTest.java`

(3 de 3)

22.8 JTabbedPane

- JTabbedPane:
  - **Organiza componentes GUI em camadas em que somente uma camada é visível de cada vez.**
    - **Quando o usuário clica em uma guia, a camada apropriada é exibida:**
      - **As guias podem ser posicionadas na parte superior (padrão), à esquerda, à direita ou na parte inferior.**
      - **Qualquer componente pode ser posicionado em uma guia.**
      - **Se as guias não se ajustarem em uma linha, elas serão empacotadas a fim de formar linhas adicionais das guias.**

Resumo

JTabbedPaneFrame.java

(1 de 2)

```
1  // Fig. 22.13: JTabbedPaneFrame.java
2  // Demonstrando JTabbedPane.
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import javax.swing.JFrame;
6  import javax.swing.JTabbedPane;
7  import javax.swing.JLabel;
8  import javax.swing.JPanel;
9  import javax.swing.JButton;
10 import javax.swing.SwingConstants;
11
12 public class JTabbedPaneFrame extends JFrame
13 {
14    // configurando GUI
15    public JTabbedPaneFrame()
16    {
17       super( "JTabbedPane Demo " );
18
19       JTabbedPane tabbedPane = new JTabbedPane(); // cria JTabbedPane
20
21       // configura o pane11 e o adiciona ao JTabbedPane
22       JLabel label1 = new JLabel( "panel one", SwingConstants.CENTER );
23       JPanel panel1 = new JPanel(); // cria o primeiro painel
24       panel1.add( label1 ); // adiciona o rótulo ao painel
25       tabbedPane.addTab( "Tab One", null, panel1, "First Panel" );
26
```

Cria um **JTabbedPane** vazio com as configurações-padrão

Chama o método **JTabbedPane addTab** com argumentos que especificam a string de título da guia, uma referência **Icon** para exibir na guia, o **COMPONENT** para exibir quando o usuário clica na guia e a string da dica de tela da guia

Resumo

JTabbedPaneFrame.java
(2 de 2)

```
27       // configura panel2 e o adiciona ao JTabbedPane
28       JLabel label2 = new JLabel( "panel two", SwingConstants.CENTER );
29       JPanel panel2 = new JPanel(); // cria o segundo panel
30       panel2.setBackground( Color.YELLOW ); // configura o fundo como amarelo
31       panel2.add( label2 ); // adiciona o rótulo ao painel
32       tabbedPane.addTab( "Tab Two", null, panel2, "Second Panel" );
33
34       // configura o panel3 e o adiciona ao JTabbedP
35       JLabel label3 = new JLabel( "panel three" );
36       JPanel panel3 = new JPanel(); // cria o terceiro painel
37       panel3.setLayout( new BorderLayout() ); // utiliza o borderlayout
38       panel3.add( new JButton( "North" ), BorderLayout.NORTH );
39       panel3.add( new JButton( "West" ), BorderLayout.WEST );
40       panel3.add( new JButton( "East" ), BorderLayout.EAST );
41       panel3.add( new JButton( "South" ), BorderLayout.SOUTH );
42       panel3.add( label3, BorderLayout.CENTER );
43       tabbedPane.addTab( "Tab Three", null, panel3, "Third Panel" );
44
45       add( tabbedPane ); // adicion a JTabbedPane ao frame
46    } // fim do construtor JTabbedPaneFrame
47 } // fim da classe JTabbedPaneFrame
```

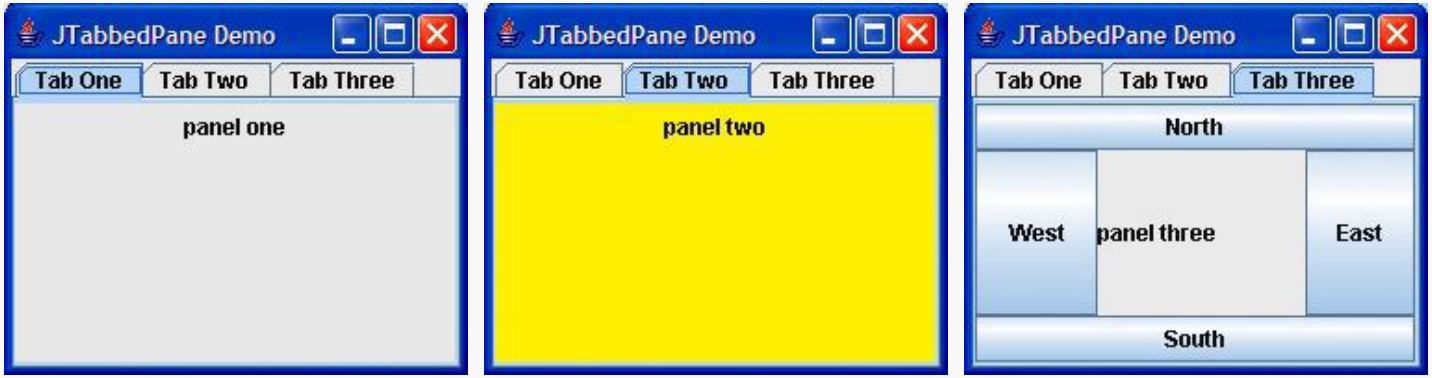
Adiciona **panel2** a **tabbedPane**

Adiciona **panel3** a **tabbedPane**

Resumo

JTabbedPaneDemo.java

```
1  // Fig. 22.14: JTabbedPaneDemo.java
2  // Demonstrando o JTabbedPane.
3  import javax.swing.JFrame;
4
5  public class JTabbedPaneDemo
6  {
7     public static void main( String args[] )
8     {
9        JTabbedPaneFrame tabbedPaneFrame = new JTabbedPaneFrame();
10       tabbedPaneFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       tabbedPaneFrame.setSize( 250, 200 ); // configura o tamanho do frame
12       tabbedPaneFrame.setVisible( true ); // exibe o frame
13    } // fim do main
14 } // fim da classe JTabbedPaneDemo
```



22.9 Gerenciadores de layout: `BoxLayout` e `GridBagLayout`

- **Gerenciador de layout `BoxLayout`:**
  - **Organiza componentes GUI horizontalmente ao longo de um eixo *x* ou verticalmente ao longo do eixo *y* do contêiner.**

Gerenciador de layout	Descrição
`BoxLayout`	Um gerenciador de layout que permite que os componentes GUI sejam organizados da esquerda para a direita ou de cima para baixo em um contêiner. A classe `Box` declara um contêiner com `BoxLayout` como seu gerenciador-padrão de layout e fornece métodos static para criar um `Box` com um `BoxLayout` horizontal ou vertical.
`GridBagLayout`	Um gerenciador de layout semelhante a `GridLayout`, mas diferentemente dele pelo fato de que os componentes podem variar de tamanho e podem ser adicionados em qualquer ordem.

Figura 22.15 | Gerenciadores de layout adicionais.

Resumo

BoxLayoutFrame.java

(1 de 3)

```
1  // Fig. 22.16: BoxLayoutFrame.java
2  // Demonstrando BoxLayout.
3  import java.awt.Dimension;
4  import javax.swing.JFrame;
5  import javax.swing.Box;
6  import javax.swing.JButton;
7  import javax.swing.BoxLayout;
8  import javax.swing.JPanel;
9  import javax.swing.JTabbedPane;
10
11 public class BoxLayoutFrame extends JFrame
12 {
13    // configura a GUI
14    public BoxLayoutFrame()
15    {
16       super( "Demonstrating BoxLayout" );
17
18       // cria contêineres Box com BoxLayout
19       Box horizontal1 = Box.createHorizontalBox();
20       Box vertical1 = Box.createVerticalBox();
21       Box horizontal2 = Box.createHorizontalBox();
22       Box vertical2 = Box.createVerticalBox();
23
24       final int SIZE = 3; // número de botões em cada Box
25
26       // adiciona botões a Box horizontal1
27       for ( int count = 0; count < SIZE; count++ )
28          horizontal1.add( new JButton( "Button " + count ) );
29
```

Cria contêineres **Box** com os métodos **Box static createHorizontalBox** e **createVerticalBox**

Adiciona três **JButton**s a **horizontal1**

Resumo

```
30      // cria a cola horizontal e adiciona botões a Box horizontal2
31      for ( int count = 0; count < SIZE; count++ )
32      {
33         vertical1.add( Box.createVerticalStrut( 25 ) );
34         vertical1.add( new JButton( "Button " + count ) );
35      } // fim do for
36
37      // cria uma área rígida e adiciona botões a Box vertical2
38      for ( int count = 0; count < SIZE; count++ )
39      {
40         horizontal2.add( Box.createHorizontalGlue() );
41         horizontal2.add( new JButton( "Button " + count ) );
42      } // fim do for
43
44      // cria uma área rígida e adiciona botões a Box vertical2
45      for ( int count = 0; count < SIZE; count++ )
46      {
47         vertical2.add( Box.createRigidArea( new Dimension( 12, 8 ) ) );
48         vertical2.add( new JButton( "Button " + count ) );
49      } // fim do for
50
51      // cria cola vertical e adiciona botões ao painel
52      JPanel panel = new JPanel();
53      panel.setLayout( new BoxLayout( panel, BoxLayout.Y_AXIS ) );
54
```

Adiciona três estruturas verticais e três **JButton**s a **vertical1**

Adiciona a cola horizontal e três **JButton**s a **horizontal2**

Adiciona três áreas rígidas e três **JButton**s a **vertical2**

Utiliza o método **Container setLayout** para configurar o layout do painel como um **BoxLayout** vertical

.java

(2 de 3)

Resumo

BoxLayoutFrame.java

(3 de 3)

```
55       for ( int count = 0; count < SIZE; count++ )
56       {
57          panel.add( Box.createGlue() );
58          panel.add( new JButton( "Button " + count ) );
59       } // fim do for
60
61       // cria um JTabbedPane
62       JTabbedPane tabs = new JTabbedPane(
63          JTabbedPane.TOP, JTabbedPane.SCROLL_TAB_LAYOUT );
64
65       // coloca cada contêiner no painel com guias
66       tabs.addTab( "Horizontal Box", horizontal1 );
67       tabs.addTab( "Vertical Box with Struts", vertical1 );
68       tabs.addTab( "Horizontal Box with Glue", horizontal2 );
69       tabs.addTab( "Vertical Box with Rigid Areas", vertical2 );
70       tabs.addTab( "Vertical Box with Glue", panel );
71
72       add( tabs ); // coloca o painel com guias no frame
73    } // fim do construtor BoxLayoutFrame
74 } // fim da classe BoxLayoutFrame
```

Adiciona a cola e três **JButton**s a **panel**

Cria um **JTabbedPane** em que as guias devem rolar se houver muitas guias para caber em uma linha

22.9 Gerenciadores de layout: `BoxLayout` e `GridBagLayout` (*Cont.*)

- **Estruturas verticais:**
  - **Componente GUI invisível que tem uma altura fixa de pixels:**
    - **Utilizado para garantir uma quantidade fixa de espaço entre componentes GUI.**
    - **Método `Box` `static` `createVerticalStrut`.**
      - **Argumento `int` determina a altura da estrutura em pixels.**
    - **`Box` também declara o método `createHorizontalStrut`.**

22.9 Gerenciadores de layout: `BoxLayout` e `GridBagLayout` (*Cont.*)

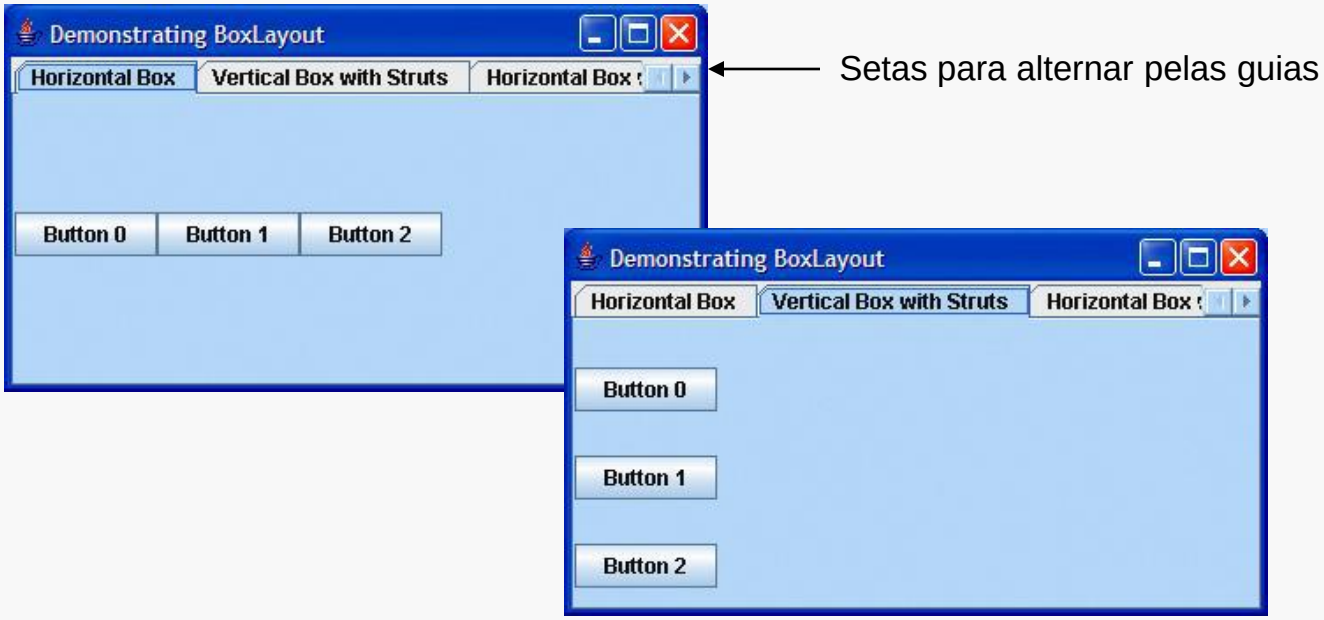
- **Cola horizontal:**
  - **Um componente GUI invisível que ocupa espaço adicional entre componentes GUI de tamanho fixo.**
    - **Quando o contêiner é redimensionado, componentes separados por cola permanecem no mesmo tamanho, mas a cola se expande ou se contrai para ocupar o espaço entre eles.**
  - **Métodos `Box` `static` `createHorizontalGlue` e `createVerticalGlue`.**

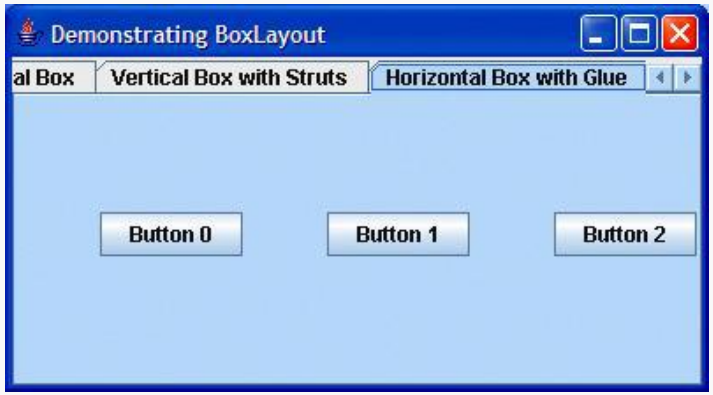
Resumo

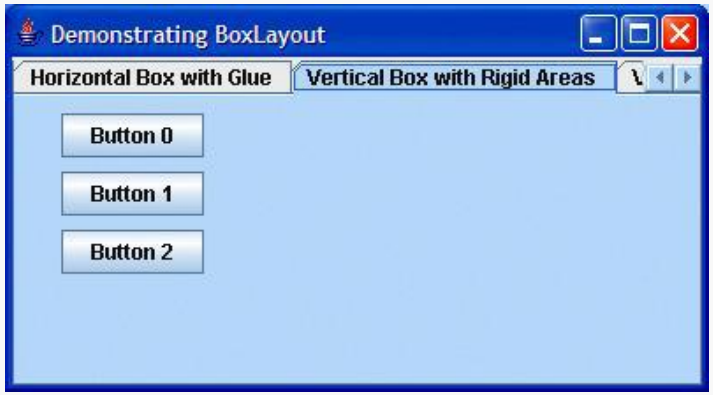
BoxLayoutDemo.java

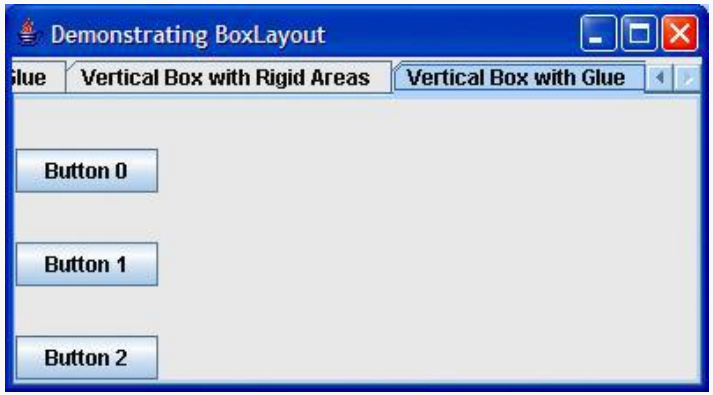
(1 de 2)

```
1  // Fig. 22.17: BoxLayoutDemo.java
2  // Demonstrando BoxLayout.
3  import javax.swing.JFrame;
4
5  public class BoxLayoutDemo
6  {
7     public static void main( String args[] )
8     {
9        BoxLayoutFrame boxLayoutFrame = new BoxLayoutFrame();
10       boxLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       boxLayoutFrame.setSize( 400, 220 ); // configura o tamanho do frame
12       boxLayoutFrame.setVisible( true ); // exibe o frame
13    } // fim do main
14 } // fim da classe BoxLayoutDemo
```









Resumo

`BoxLayoutDemo.java`

(2 de 2)

22.9 Gerenciadores de layout: `BoxLayout` e `GridBagLayout` (*Cont.*)

- **Áreas rígidas:**
  - **Um componente GUI invisível que sempre tem uma largura fixa e uma altura fixa em pixels.**
    - **O argumento do objeto `Dimension` para o método `Box` `static` `createRigidArea` especifica a largura e a altura da área.**

22.9 Gerenciadores de layout: `BoxLayout` e `GridBagLayout` (*Cont.*)

- **O gerenciador de layout `GridBagLayout`:**
  - **Semelhante a `GridLayout` pelo fato de que organiza componentes em uma grade, porém mais flexível.**
    - **Os componentes podem variar de tamanho e podem ser adicionados em qualquer ordem.**
  - **Determinando a aparência da GUI:**
    - **Desenhe a GUI no papel.**
    - **Desenhe uma grade sobre ela, dividindo os componentes em linhas e colunas.**
      - **Os números iniciais de linha e coluna devem ser 0.**
      - **Utilizados pelo gerenciador de layout `GridBagLayout` para posicionar adequadamente os componentes na grade.**

22.9 Gerenciadores de layout: BoxLayout e GridBagLayout (*Cont.*)

- Objeto `GridBagConstraints`:
  - Descreve como um componente é posicionado em um `GridBagLayout`.
  - `Anchor` especifica a posição relativa do componente em uma área que ele não preenche.
    - Constantes: `NORTH`, `NORTHEAST`, `EAST`, `SOUTHEAST`, `SOUTH`, `SOUTHWEST`, `WEST`, `NORTHWEST` e `CENTER` (o padrão).
  - `fill` define como o componente aumenta se a área em que pode ser exibido for maior que o componente.
    - Constantes: `NONE` (o padrão), `VERTICAL`, `HORIZONTAL` e `BOTH`.

22.9 Gerenciadores de layout: BoxLayout e GridBagLayout (*Cont.*)

- `gridx` e `gridy` especificam onde o canto superior esquerdo do componente é posicionado na grade.
- `gridwidth` e `gridheight` especificam o número de colunas e linhas que um componente ocupa.
- `weightx` e `weighty` especificam como distribuir espaço extra horizontal e vertical para componentes da grade em um `GridBagLayout` quando o contêiner é redimensionado.
  - Um valor zero indica que o componente da grade não aumenta nessa dimensão por conta própria.
    - Entretanto, se o componente avançar sobre uma coluna/linha que contém um componente com valor de peso diferente de zero, ele aumentará na mesma proporção dos outros componentes nessa coluna/linha.
  - Utilize valores de peso positivos diferentes de zero para evitar 'aglomeração'.

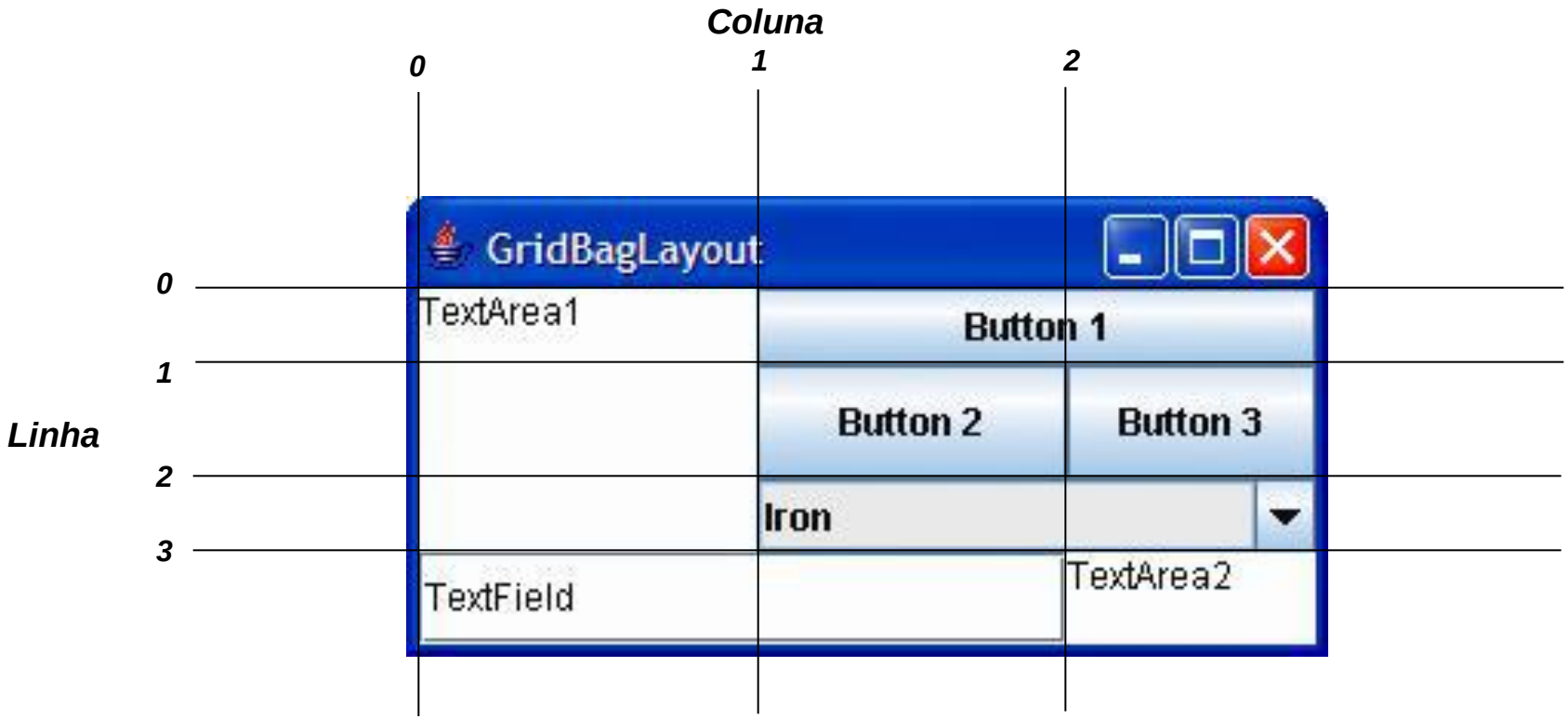


Figura 22.18 | Projetando uma GUI que utilizará `GridBagLayout`.

Campo GridBagCons-traints	Descrição
anchor	Especifica a posição relativa (NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, NORTHWEST, CENTER) do componente em uma área que ele não preenche.
fill	Redimensiona o componente na direção especificada (NONE, HORIZONTAL, VERTICAL, BOTH) quando a área de exibição for maior que o componente.
gridx	A coluna em que o componente será colocado.
gridy	A linha em que o componente será colocado.
gridwidth	O número de colunas que o componente ocupa.
gridheight	O número de linhas que o componente ocupa.
weightx	A quantidade de espaço extra a alocar horizontalmente. O componente na grade pode tornar-se mais largo se houver espaço extra disponível.
weighty	A quantidade de espaço extra a alocar verticalmente. O componente na grade pode tornar-se mais alto se houver espaço extra disponível.

Figura 22.19 | Campos GridBagConstraints.

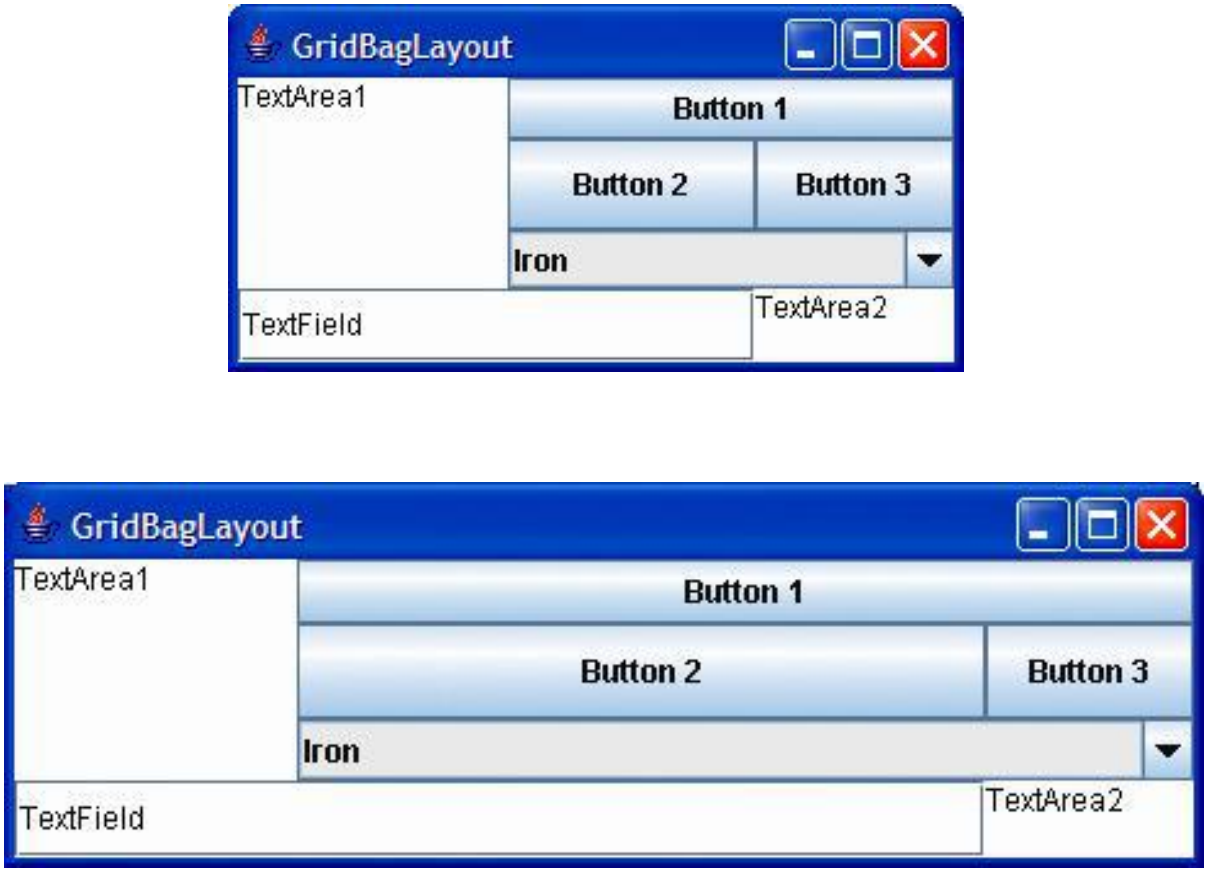


Figura 22.20 | `GridBagLayout` com os pesos configurados como zero.

Resumo

GridBagFrame.java

(1 de 3)

```
1  // Fig. 22.21: GridBagFrame.java
2  // Demonstrando GridBagLayout.
3  import java.awt.GridBagLayout;
4  import java.awt.GridBagConstraints;
5  import java.awt.Component;
6  import javax.swing.JFrame;
7  import javax.swing.JTextArea;
8  import javax.swing.JTextField;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11
12 public class GridBagFrame extends JFrame
13 {
14    private GridBagLayout layout; // layout desse frame
15    private GridBagConstraints constraints; // restrições desse layout
16
17    // configura a GUI
18    public GridBagFrame()
19    {
20       super( "GridBagLayout" );
21       layout = new GridBagLayout();
22       setLayout( layout ); // configura o layout de frame
23       constraints = new GridBagConstraints(); // instancia restrições
24
25       // cria componentes GUI
26       JTextArea textArea1 = new JTextArea( "TextArea1", 5, 10 );
27       JTextArea textArea2 = new JTextArea( "TextArea2", 2, 2 );
28
```

Cria um objeto **GridBagLayout**

Cria um objeto **GridBagConstraints**

Resumo

GridBagFrame.java

(2 de 3)

```
29      String names[] = { "Iron", "Steel", "Brass" };
30      JComboBox comboBox = new JComboBox( names );
31
32      JTextField textField = new JTextField( "TextField" );
33      JButton button1 = new JButton( "Button 1" );
34      JButton button2 = new JButton( "Button 2" );
35      JButton button3 = new JButton( "Button 3" );
36
37      // weightx e weighty para textArea1 são 0: o padrão
38      // anchor para todos os componentes CENTER: o padrão
39      constraints.fill = GridBagConstraints.BOTH;
40      addComponent( textArea1, 0, 0, 1, 3 );
41
42      // weightx e weighty para button1 são 0: o padrã
43      constraints.fill = GridBagConstraints.HORIZONTAL
44      addComponent( button1, 0, 1, 2, 1 );
45
46      // weightx e weighty para comboBox são 0: o padr
47      // fill é HORIZONTAL
48      addComponent( comboBox, 2, 1, 2, 1 );
49
50      // button2
51      constraints.weightx = 1000; // pode crescer na largura
52      constraints.weighty = 1;    // pode crescer na altura
53      constraints.fill = GridBagConstraints.BOTH;
54      addComponent( button2, 1, 1, 1, 1 );
55
```

Faz com que a **JTextArea** sempre preencha toda a sua área alocada

Chama o método utilitário **addComponent** com o objeto **JTextArea**, a linha, a coluna e os números de colunas e linhas a distribuir são passados como argumentos

Quando a janela for redimensionada, **button2** crescerá.

Resumo

```
56       // preenchimento é BOTH para button3
57       constraints.weightx = 0;
58       constraints.weighty = 0;
59       addComponent( button3, 1, 2, 1, 1 );
60
61       // weightx e weighty para textField são 0, preenchimento é BOTH
62       addComponent( textField, 3, 0, 2, 1 );
63
64       // weightx e weighty para textArea2 são 0, preenchimento é BOTH
65       addComponent( textArea2, 3, 2, 1, 1 );
66    } // fim do construtor GridBagFrame
67
68    // método no qual configurar as restrições
69    private void addComponent( Component component,
70       int row, int column, int width, int height )
71    {
72       constraints.gridx = column; // configura gridx
73       constraints.gridy = row; // configura gridy
74       constraints.gridwidth = width; // configura gridwidth
75       constraints.gridheight = height; // configura gridheight
76       layout.setConstraints( component, constraints ); // configura constraints
77       add( component ); // adiciona component
78    } // fim do método addComponent
79 } // fim da classe GridBagFrame
```

button3 ainda aumentará por causa dos valores de peso do **button2**

Configura restrições e adiciona um componente

GridBagFrame .java

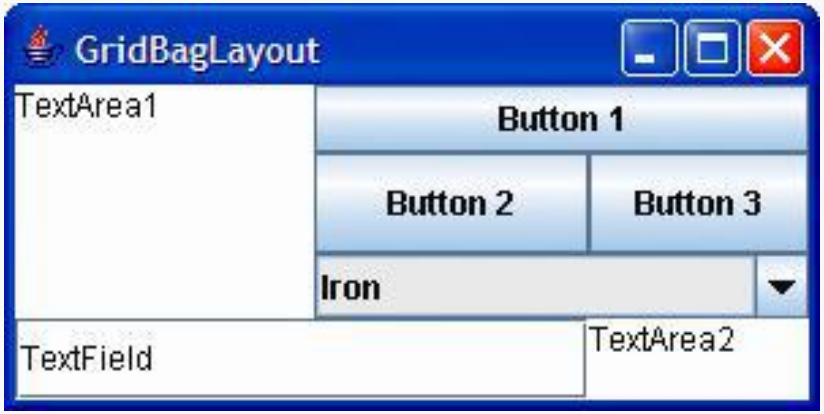
(3 de 3)

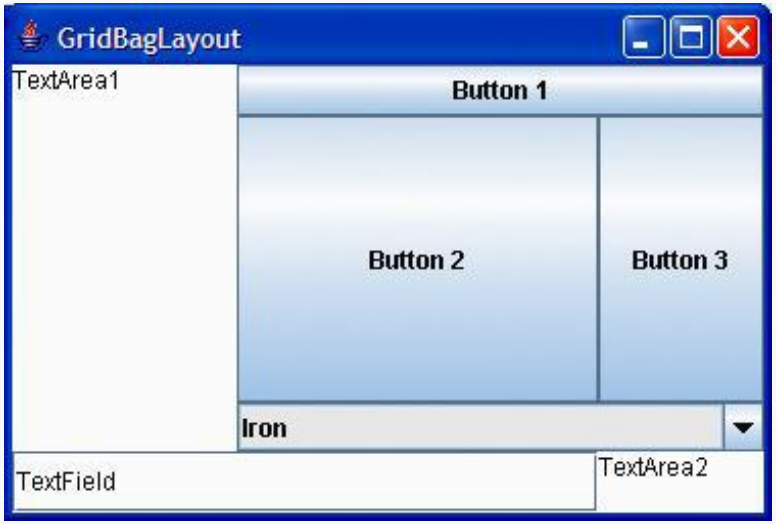
Resumo

GridBagDemo.java

(1 de 2)

```
1  // Fig. 22.22: GridBagDemo.java
2  // Demonstrando GridBagLayout.
3  import javax.swing.JFrame;
4
5  public class GridBagDemo
6  {
7     public static void main( String args[] )
8     {
9        GridBagFrame gridBagFrame = new GridBagFrame();
10       gridBagFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       gridBagFrame.setSize( 300, 150 ); // configura tamanho
12       gridBagFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe GridBagDemo
```





Resumo

`GridBagDemo.java`

(2 de 2)

22.9 Gerenciadores de layout: `BoxLayout` e `GridBagLayout` (*Cont.*)

- **Constantes `GridBagConstraints`:**
  - **`RELATIVE`:**
    - **Especifica que o penúltimo componente em uma linha particular deve ser posicionado à direita do componente anterior na linha.**
  - **`REMAINDER`:**
    - **Especifica que um componente é o último componente em uma linha.**
  - **Os componentes que não são o penúltimo ou o último componente em uma linha devem especificar valores para `gridwidth` e `gridheight`:**

Resumo

GridBagFrame2.java

(1 de 3)

```
1  // Fig. 22.23: GridBagFrame2.java
2  // Demonstrando as constantes GridBagLayout.
3  import java.awt.GridBagLayout;
4  import java.awt.GridBagConstraints;
5  import java.awt.Component;
6  import javax.swing.JFrame;
7  import javax.swing.JComboBox;
8  import javax.swing.JTextField;
9  import javax.swing.JList;
10 import javax.swing.JButton;
11
12 public class GridBagFrame2 extends JFrame
13 {
14    private GridBagLayout layout; // layout desse frame
15    private GridBagConstraints constraints; // restrições desse layout
16
17    // configura a GUI
18    public GridBagFrame2()
19    {
20       super( "GridBagLayout" );
21       layout = new GridBagLayout();
22       setLayout( layout ); // configura layout do frame
23       constraints = new GridBagConstraints(); // instancia restrições
24
25       // cria componentes da GUI
26       String metals[] = { "Copper", "Aluminum", "Silver" };
27       JComboBox comboBox = new JComboBox( metals );
28
29       JTextField textField = new JTextField( "TextField" );
30
```

Cria um objeto **GridBagLayout**

Resumo

GridBagFrame2.java

(2 de 3)

```
31    String fonts[] = { "Serif", "Monospaced" };T
32    JList list = new JList( fonts );
33
34    String names[] = { "zero", "one", "two", "three", "four" };
35    JButton buttons[] = new JButton[ names.length ];
36
37    for ( int count = 0; count < buttons.length; count++ )
38       buttons[ count ] = new JButton( names[ count ] );
39
40    // define define restrições dos componentes GUI para textField
41    constraints.weightx = 1;
42    constraints.weighty = 1;
43    constraints.fill = GridBagConstraints.BOTH;
44    constraints.gridwidth = GridBagConstraints.REMAINDER;
45    addComponent( textField );
46
47    // buttons[0] -- weightx e weighty são 1: fill é BOTH
48    constraints.gridwidth = 1;
49    addComponent( buttons[ 0 ] );
50
51    // buttons[1] -- weightx e weighty são 1: fill é BOTH
52    constraints.gridwidth = GridBagConstraints.RELATIVE;
53    addComponent( buttons[ 1 ] );
54
55    // buttons[2] -- weightx e weighty são 1: fill é BOTH
56    constraints.gridwidth = GridBagConstraints.REMAINDER;
57    addComponent( buttons[ 2 ] );
58
```

Especifica que o **`JTextField`** é o último componente na linha.

Especifica que **`JButton`** deve ser posicionado em relação ao componente anterior

Esse **`JButton`** é o último componente na linha

Resumo

```
59       // comboBox -- weightx é 1: fill é BOTH
60       constraints.weighty = 0;
61       constraints.gridwidth = GridBagConstraints.REMAINDER;
62       addComponent( comboBox );
63
64       // buttons[3] -- weightx é 1: fill é BOTH
65       constraints.weighty = 1;
66       constraints.gridwidth = GridBagConstraints.REMAINDER;
67       addComponent( buttons[ 3 ] );
68
69       // buttons[4] -- weightx e weighty são 1: fill é BOTH
70       constraints.gridwidth = GridBagConstraints.RELATIVE;
71       addComponent( buttons[ 4 ] );
72
73       // list -- weightx e weighty são 1: fill é BOTH
74       constraints.gridwidth = GridBagConstraints.REMAINDER;
75       addComponent( list );
76    } // fim do construtor GridBagFrame2
77
78    // adiciona um componente ao contêiner
79    private void addComponent( Component component )
80    {
81       layout.setConstraints( component, constraints );
82       add( component ); // adiciona componente
83    } // fim do método addComponent
84 } // fim da classe GridBagFrame2
```

O **JComboBox** é o único componente na linha

GridBagFrame2.java

(3 de 3)

Esse **JButton** é o único componente na linha

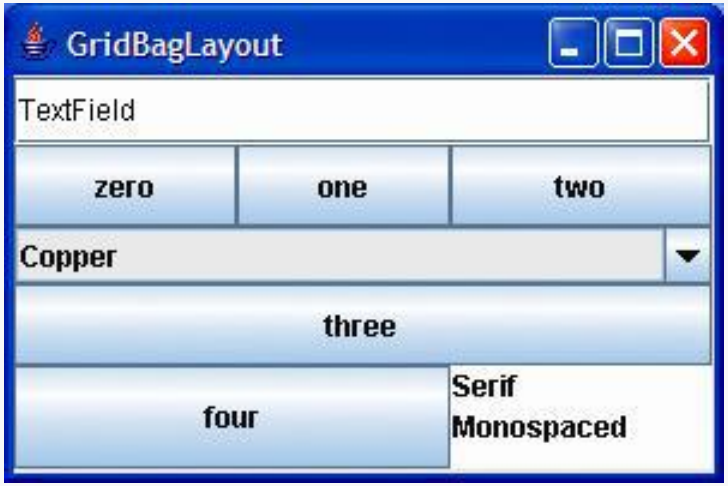
Esse **JButton** é o penúltimo componente na linha

Resumo

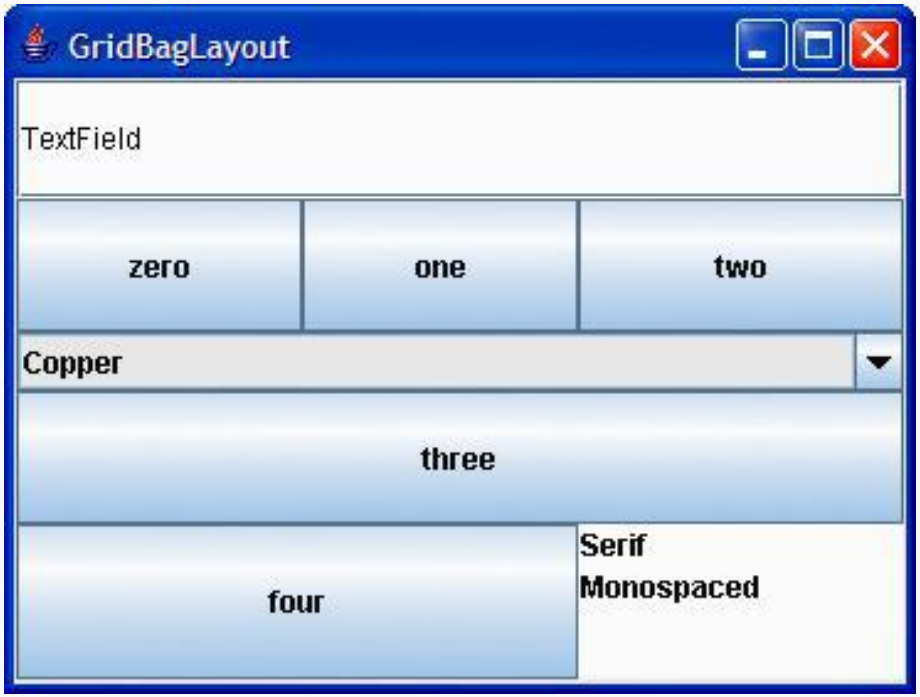
GridBagDemo2.java

(1 de 2)

```
1  // Fig. 22.24: GridBagDemo2.java
2  // Demonstrando constantes do GridBagLayout.
3  import javax.swing.JFrame;
4
5  public class GridBagDemo2
6  {
7     public static void main( String args[] )
8     {
9        GridBagFrame2 gridBagFrame = new GridBagFrame2();
10       gridBagFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       gridBagFrame.setSize( 300, 200 ); // configura o tamanho do frame
12       gridBagFrame.setVisible( true ); // exibe o frame
13    } // fim do main
14 } // fim da classe GridBagDemo2
```



Resumo



`GridBagDemo2.java`

(2 de 2)