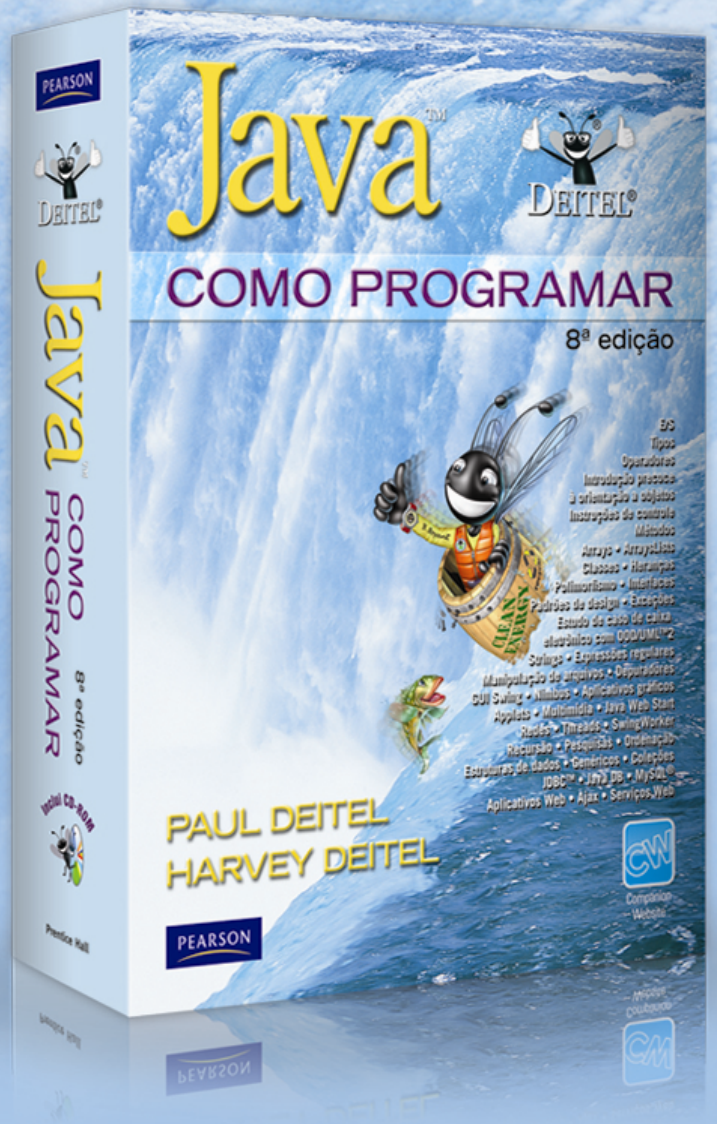
Capítulo 25 Componentes GUI: Parte 2

Java Como Programar, 8/E



COMO PROGRAMAR

8ª edição

OBJETIVOS

Neste capítulo, você aprenderá:

- A criar e manipular controles deslizantes, menus, menus pop-up e janelas.
- A alterar programaticamente a aparência e comportamento de uma GUI, utilizando a aparência e comportamento plugável do Swing.
- A criar uma interface de múltiplos documentos com `JDesktopPane` e `JInternalFrame`.
- A utilizar gerenciadores adicionais de layout.

COMO PROGRAMAR

8ª edição

25.1 Introdução

- Neste capítulo, discutimos:
  - Componentes e gerenciadores de layout adicionais e projetamos a base para construir GUIs mais complexas.
  - Sliders que permitem selecionar a partir de um intervalo de valores inteiros.
  - Detalhes adicionais de janelas.
  - Menus que permitem realizar tarefas eficientemente no programa.
  - **Interface plugável (*pluggable look-and-feel*, PLAF)** do Swing.
  - Interface de múltiplos documentos (*multiple document interface*, MDI) — uma janela principal (frequentemente chamada janela-pai) contendo outras janelas (frequentemente chamadas janelas-filhas) para gerenciar vários documentos abertos em paralelo.

25.2 Jslider

- **Jsliders** permitem a um usuário selecionar a partir de um intervalo de valores inteiros.
- A Figura 25.1 mostra um `Jslider` horizontal com **marcas de medidas** e a **caixa de rolagem** que permitem ao usuário selecionar um valor.
- Podem ser personalizadas para exibir marcas de medida principais, marcas de medida secundárias e rótulos para as marcas de medida.
- Também suportam **marcas de aderência**, que fazem a miniatura, quando posicionada entre duas marcas, aderir à marca mais próxima.

Figura 25.1 | Um componente `JSlider` com uma orientação horizontal.

- A tecla da seta para baixo e a tecla da seta para cima também fazem com que a caixa de rolagem do `JSlider` diminua ou aumente por 1, respectivamente.
- 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 do intervalo de valores, respectivamente.
- A tecla *Home* move a caixa de rolagem para o valor mínimo do `JSlider`, e a tecla *End* move a caixa de rolagem para o valor máximo do `JSlider`.

- `JSliders` têm uma orientação horizontal ou uma orientação vertical.
- Para um `JSlider` horizontal, o valor mínimo está na extremidade esquerda do `JSlider` e o valor máximo está na extremidade direita.
- Para um `JSlider` vertical, o valor mínimo está na parte inferior e o valor máximo na parte superior.
- As posições de valor mínimo e máximo em um `JSlider` podem ser alternadas chamando o método `JSlider` **setInverted** com argumento `boolean true`.


```
1  // Figura 25.2: 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 e então repinta
20    public void setDiameter( int newDiameter )
21    {
22       // se diâmetro inválido, assume o padrão de 10
23       diameter = ( newDiameter >= 0 ? newDiameter : 10 );
24       repaint(); // repinta o painel
25    } // fim do método setDiameter
```

Figura 25.2 | Subclasse `JPanel` para desenhar círculos de um diâmetro especificado. (Parte 1 de 2.)

```
26
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
```

Figura 25.2 | Subclasse `JPanel` para desenhar círculos de um diâmetro especificado. (Parte 2 de 2.)


```
1  // Figura 25.3: 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 para selecionar o diâmetro
14    private OvalPanel myPanel; // painel para desenhar um círculo
15
16    // construtor sem argumento
17    public SliderFrame()
18    {
19       super( "Slider Demo" );
20
21       myPanel = new OvalPanel(); // cria o painel para desenhar um círculo
22       myPanel.setBackground( Color.YELLOW ); // configura o fundo como amarelo
23
```

Figura 25.3 | Valor do `JSlider` utilizado para determinar o diâmetro de um círculo. (Parte 1 de 2.)

```
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 medida a cada 10
28          diameterJSlider.setPaintTicks( true ); // pinta as marcas de medida no controle deslizante
29
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 a addChangeListener
42
43          add( diameterJSlider, BorderLayout.SOUTH ); // adiciona um controle deslizante ao quadro
44          add( myPanel, BorderLayout.CENTER ); // adiciona painel ao frame
45       } // fim do construtor de SliderFrame
46    } // fim da classe SliderFrame
```

Figura 25.3 | Valor do `JSlider` utilizado para determinar o diâmetro de um círculo. (Parte 2 de 2.)


```
1  // Figura 25.4: 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
```

Figura 25.4 | Classe de teste para `SliderFrame`. (Parte 1 de 2.)

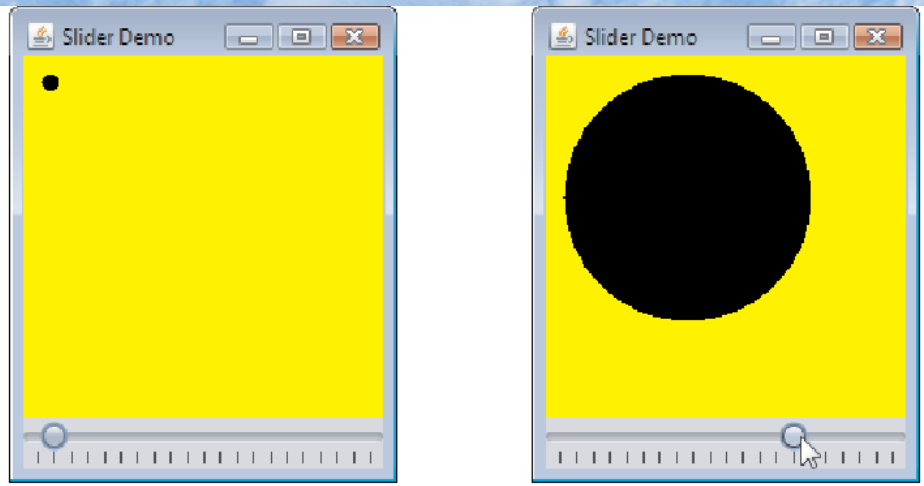


Figura 25.4 | Classe de teste para `SliderFrame`. (Parte 2 de 2.)

- O método `JSlider` setMajorTickSpacing indica que cada marca de medida principal representa 10 valores no intervalo de valores suportados pelo `JSlider`.
- O método `JSlider` **setPaintTicks** com um argumento `true` indica que as marcas de medida devem ser exibidas (elas não são exibidas por padrão).
- `JSlider`s geram **ChangeEvent**s (o pacote `javax.swing.event`) em resposta a interações de usuário.
- Tratado por um **ChangeListener** (pacote `javax.swing.event`) que declara o método **stateChanged**.
- O método `JSlider` **getValue** retorna a posição da caixa de rolagem atual.

25.3 Janelas: notas adicionais

- Um `JFrame` *é uma* **janela** com uma **barra de título** e uma borda.
- `JFrame` é uma subclasse de `Frame`, que é uma subclasse de `Window`.
- Estes são componentes Swing GUI "de maior peso".
- Uma janela é fornecida pelo kit de ferramentas da plataforma local.
- Por padrão, quando o usuário fecha uma janela `JFrame`, ela é oculta, mas é possível controlar isso com o método `JFrame` **setDefaultCloseOperation**.
- A interface **WindowConstants** (pacote `javax.swing`), que a classe `JFrame` implementa, declara três constantes — `DISPOSE_ON_CLOSE`, `DO_NOTHING_ON_CLOSE` e `HIDE_ON_CLOSE` (o padrão) — para usar com esse método.

- A classe `Window` (uma superclasse indireta de `JFrame`) declara o método **dispose** para retornar os recursos de uma janela para o sistema.
- Quando uma janela não é mais necessária em um aplicativo, você deve descartá-la explicitamente.
- Pode ser feito chamando o método `dispose` de `Window` ou chamando o método `setDefaultCloseOperation` com o argumento `WindowConstants.DISPOSE_ON_CLOSE`.
- Uma janela não é exibida até o programa invocar o método `setVisible` da janela com um argumento `true`.
- O tamanho de uma janela deve ser configurado com uma chamada para o método `setSize`.
- A posição de uma janela quando aparece na tela é especificada com o método **setLocation.**

Dica de desempenho 25.1

Uma janela é um recurso de sistema caro. Retorne-a ao sistema chamando seu método `dispose` *quando a janela não mais é necessária.*

Erro comum de programação 25.1

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

Erro comum de programação 25.2

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

- Quando o usuário manipula a janela, ocorrem **eventos de janela.**
- Os ouvintes de evento são registrados para eventos de janela com o método `Window` **addWindowListener.**
- A interface **WindowListener** fornece sete métodos de tratamento de evento de janela.
- **windowActivated** (chamado quando o usuário torna uma janela ativa).
- **windowClosed** (chamado depois que a janela é fechada).
- **windowClosing** (chamado quando o usuário inicia o fechamento da janela).
- **windowDeactivated** (chamado quando o usuário torna outra janela ativa).
- **windowDeiconified** (chamado quando a janela é restaurada a partir do estado minimizado).
- **windowIconified** (chamado quando a janela está minimizada).
- **windowOpened** (chamado logo que a janela é aberta pela primeira vez).

Observação sobre a aparência e o comportamento 25.1

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

25.4 Utilizando menus com frames

- Os **menus** são uma parte integrante das GUIs.
- Permitem que o usuário realize ações sem poluir desnecessariamente uma GUI com componentes extras.
- Em GUIs Swing, os menus podem ser anexados somente aos objetos das classes que fornecem o método **setJMenuBar.**
- Duas dessas classes são `JFrame` e `JApplet`.
- As classes usadas para declarar menus são `JMenuBar`, `JMenu`, `JMenuItem`, `JCheckBoxMenuItem` e a `JRadioButtonMenuItem`.

- A classe **JMenuBar** (uma subclasse de `JComponent`) gerencia uma **barra de menu**, que é um contêiner para menus.
- A classe **JMenu** (uma subclasse do `javax.swing.JMenuItem`) contém os métodos necessários para gerenciar menus.
- Os menus contêm itens de menu e são adicionados a barras de menus ou a outros menus como submenus.
- Classe **JMenuItem** (uma subclasse do `javax.swing.AbstractButton`) — contém os métodos necessários para gerenciar **itens de menu.**
- Um item do menu causa um evento de ação quando clicado.
- Pode ser também um **submenu** que fornece mais itens de menu a partir dos quais o usuário pode selecionar.

- Classe **JCheckBoxMenuItem** (uma subclasse de `javax.swing.JMenuItem`) — itens de menu que podem ser ativados e desativados.
- Classe **JRadioButtonMenuItem** (uma subclasse de `javax.swing.JMenuItem`) — itens de menu que podem ser ativados e desativados como `JCheckBoxMenuItem`s.
- Quando múltiplos `JRadioButtonMenuItem`s são mantidos como parte de um `ButtonGroup`, somente um item no grupo pode ser selecionado em determinado momento.
- **Mnemônicos** podem fornecer acesso rápido a um menu ou ao item de menu do teclado.
- Pode ser usado com todas as subclasses de `javax.swing.AbstractButton`.
- O método `JMenu` **setMnemonic** (herdado da classe `AbstractButton`) indica o mnemônico de um menu.


```
1  // Figura 25.5: 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
21 public class MenuFrame extends JFrame
22 {
23    private final Color[] colorValues =
24       { Color.BLACK, Color.BLUE, Color.RED, Color.GREEN };
```

Figura 25.5 | JMenus e mnemônicos. (Parte 1 de 9.)


```
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 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 estilos de fontes
32
33       // construtor sem argumento para configurar 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          // cria item de menu About...
42          JMenuItem aboutItem = new JMenuItem( "About..." );
43          aboutItem.setMnemonic( 'A' ); // configura o mnemônico com A
44          fileMenu.add( aboutItem ); // adiciona o item about ao menu File
45          aboutItem.addActionListener(
46
47             new ActionListener() // classe interna anônima
48             {
```

Figura 25.5 | JMenus e mnemônicos. (Parte 2 de 9.)

```
49             // exibe um diálogo de mensagem quando o usuário seleciona About...
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 clica 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
```

Figura 25.5 | `JMenus` e mnemônicos. (Parte 3 de 9.)

```
73
74          JMenuBar bar = new JMenuBar(); // cria a barra de menus
75          setJMenuBar( bar ); // adiciona uma barra de menus ao aplicativo
76          bar.add( fileMenu ); // adiciona o menu File à barra de menus
77
78          JMenu formatMenu = new JMenu( "Format" ); // cria o menu Format
79          formatMenu.setMnemonic( 'r' ); // configura o 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 o mnemônico como C
86
87          // cria itens de menu de botão de opção para cores
88          colorItems = new JRadioButtonMenuItem[ colors.length ];
89          colorButtonGroup = new ButtonGroup(); // gerencia cores
90          ItemHandler itemHandler = new ItemHandler(); // handler para cores
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 item
```

Figura 25.5 | `JMenus` e mnemônicos. (Parte 4 de 9.)

```
97             colorMenu.add( colorItems[ count ] ); // adiciona o item ao menu Color
98             colorButtonGroup.add( colorItems[ count ] ); // adiciona ao grupo
99             colorItems[ count ].addActionListener( itemHandler );
100         } // fim do for
101
102         colorItems[ 0 ].setSelected( true ); // seleciona o primeiro item Color
103
104         formatMenu.add( colorMenu ); // adiciona o menu Color ao menu Format
105         formatMenu.addSeparator(); // adiciona um separador no menu
106
107         // array listando nomes de fonte
108         String[] fontNames = { "Serif", "Monospaced", "SansSerif" };
109         JMenu fontMenu = new JMenu( "Font" ); // cria a fonte do menu
110         fontMenu.setMnemonic( 'n' ); // configura o mnemônico como n
111
112         // cria itens do menu do tipo radio button para nomes de fonte
113         fonts = new JRadioButtonMenuItem[ fontNames.length ];
114         fontButtonGroup = new ButtonGroup(); // gerencia os nomes das fontes
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 Font
```

Figura 25.5 | JMenus e mnemônicos. (Parte 5 de 9.)


```
121             fontButtonGroup.add( fonts[ count ] ); // adiciona ao grupo de botões
122             fonts[ count ].addActionListener( itemHandler ); // adiciona handler
123          } // fim do for
124
125          fonts[ 0 ].setSelected( true ); // seleciona o primeiro item do menu Font
126          fontMenu.addSeparator(); // adiciona uma barra separadora ao menu Font
127
128          String[] styleNames = { "Bold", "Italic" }; // nomes dos estilos
129          styleItems = new JCheckBoxMenuItem[ styleNames.length ];
130          StyleHandler styleHandler = new StyleHandler(); // handler de estilo
131
132          // cria 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 ] ); // adiciona ao menu Font
138             styleItems[ count ].addItemListener( styleHandler ); // handler
139          } // fim do for
140
141          formatMenu.add( fontMenu ); // adiciona o menu Font ao menu Format
142          bar.add( formatMenu ); // adiciona o menu Format à barra de menus
143
```

Figura 25.5 | JMenus e mnemônicos. (Parte 6 de 9.)

```
144          // configura o rótulo para exibir texto
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 do construtor de MenuFrame
152
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 a seleção 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             } // for final
```

Figura 25.5 | JMenus e mnemônicos. (Parte 7 de 9.)

```
168
169            // processa a 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, 72 ) );
176               } // fim do if
177            } // fim do for
178
179            repaint(); // redesenha o aplicativo
180         } // fim do método actionPerformed
181      } // fim da classe ItemHandler
182
183      // classe interna para tratar eventos de itens de menu da caixa de verificação
184      private class StyleHandler implements ItemListener
185      {
186         // processa seleções de estilo da fonte
187         public void itemStateChanged( ItemEvent e )
188         {
189            String name = displayJLabel.getFont().getName(); // Font atual
190            Font font; // nova fonte baseada nas seleções de usuário
191
```

Figura 25.5 | JMenus e mnemônicos. (Parte 8 de 9.)


```
192             // determina quais itens são verificados e cria Font
193             if ( styleItems[ 0 ].isSelected() &&
194                  styleItems[ 1 ].isSelected() )
195                font = new Font( name, Font.BOLD + Font.ITALIC, 72 );
196             else if ( styleItems[ 0 ].isSelected() )
197                font = new Font( name, Font.BOLD, 72 );
198             else if ( styleItems[ 1 ].isSelected() )
199                font = new Font( name, Font.ITALIC, 72 );
200             else
201                font = new Font( name, Font.PLAIN, 72 );
202
203             displayJLabel.setFont( font );
204             repaint(); // redesenha o aplicativo
205          } // fim do método itemStateChanged
206       } // fim da classe StyleHandler
207    } // fim da classe MenuFrame
```

Figura 25.5 | JMenus e mnemônicos. (Parte 9 de 9.)


```
1  // Figura 25.6: 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(); // criar 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 de main
14 } // fim da classe MenuTest
```

Figura 25.6 | Classe de teste para MenuFrame. (Parte 1 de 2.)

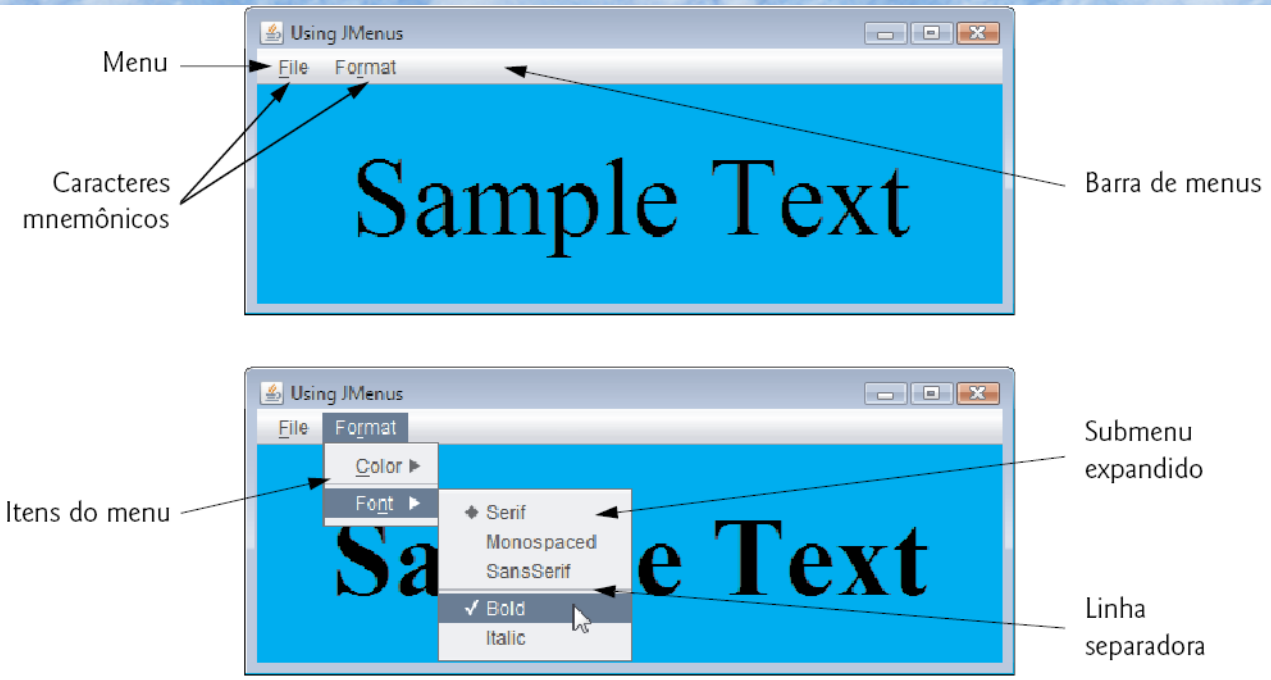


Figura 25.6 | Classe de teste para `MenuFrame`. (Parte 2 de 2.)

Observação sobre a aparência e o comportamento 25.2

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

Observação sobre a aparência e o comportamento 25.3

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 do nome (por exemplo, x *comumente é escolhido para um botão* Exit *ou um item do menu). Mnemônicos não fazem distinção entre maiúsculas e minúsculas.*

- Na maioria das utilizações anteriores de `showMessageDialog`, o primeiro argumento era `null`.
- O primeiro argumento especifica a **janela-pai** que 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.
- Ao utilizar a referência `this` em uma classe interna, especificar `this` sozinha faz referência ao objeto da classe interna.
- Para referir-se à referência `this` dos objetos da classe externa, qualifique `this` com o nome da classe externa e um ponto (`.`).

- Em geral, as caixas de diálogo são modais — não permitem que nenhuma outra janela do aplicativo seja acessada até que a caixa de diálogo seja fechada.
- A classe **JDialog** pode ser utilizada para criar seus próprios diálogos modais ou não modais.
- O método `JMenuBar` **add** anexa um menu a um `JMenuBar`.
- O método `AbstractButton` **setSelected** seleciona o botão especificado.
- O método `JMenu` **addSeparator** adiciona uma linha **separadora** horizontal a um menu.
- O método `AbstractButton` **isSelected** determina se um botão está selecionado.

Erro comum de programação 25.3

Esquecer-se de configurar a barra de menus com o método `JFrame` `set-JMenuBar` *impede que a barra de menus seja exibida no* `JFrame`.

Observação sobre a aparência e o comportamento 25.4

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

Observação sobre a aparência e o comportamento 25.5

Um submenu é criado adicionando um menu como um item de menu a 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 a aparência e o comportamento 25.5

Um submenu é criado adicionando um menu como um item de menu a 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 a aparência e o comportamento 25.7

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

25.5 JPopupMenu

- **Menus pop-up sensíveis ao contexto** são criados com a classe **JPopupMenu** (uma subclasse de `JComponent`).
- Fornecem opções que são específicas ao componente para o qual o **evento de gatilho pop-up** foi gerado. Em geral, ocorre quando o usuário pressiona e libera o botão direito do mouse.
- O método `MouseEvent` **isPopupTrigger** retorna `true` se o evento de gatilho pop-up ocorrer.
- O método `JPopupMenu` **show** exibe um `JPopupMenu`.
- O primeiro argumento especifica o **componente de origem**. Sua posição ajuda a determinar onde o `JPopupMenu` aparecerá na tela.
- Os dois últimos argumentos são as coordenadas *x*-*y* (medidas a partir do canto superior esquerdo do componente de origem) em que o `JPopupMenu` deve aparecer.

Observação sobre a aparência e o comportamento 25.8

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.


```
1  // Figura 25.7: 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; // contém itens para cores
16    private final Color[] colorValues =
17       { Color.BLUE, Color.YELLOW, Color.RED }; // cores a serem utilizadas
18    private JPopupMenu popupMenu; // permite que o usuário selecione a cor
19
20    // construtor sem argumento configura a GUI
21    public PopupFrame()
22    {
23       super( "Using JPopupMenus" );
24
```

Figura 25.7 | JPopupMenu para selecionar cores. (Parte 1 de 4.)


```
25          ItemHandler handler = new ItemHandler(); // handler para itens de menu
26          String[] colors = { "Blue", "Yellow", "Red" }; // array de cores
27
28          ButtonGroup colorGroup = new ButtonGroup(); // gerencia itens de cor
29          popupMenu = new JPopupMenu(); // cria menu pop-up
30          items = new JRadioButtonMenuItem[ colors.length ]; // aplica cores aos itens
31
32          // cria item de menu, adiciona-o ao menu pop-up, permite tratamento de eventos
33          for ( int count = 0; count < items.length; count++ )
34          {
35             items[ count ] = new JRadioButtonMenuItem( colors[ count ] );
36             popupMenu.add( items[ count ] ); // adiciona o item ao menu pop-up
37             colorGroup.add( items[ count ] ); // adiciona o item ao grupo de botões
38             items[ count ].addActionListener( handler ); // adiciona handler
39          } // for final
40
41          setBackground( Color.WHITE ); // configura o fundo como branco
42
43          // declara um MouseListener para a janela a fim de exibir o menu pop-up
44          addMouseListener(
45
46             new MouseAdapter() // classe interna anônima
47             {
```

Figura 25.7 | JPopupMenu para selecionar cores. (Parte 2 de 4.)

```
48              // trata eventos de pressionamento do mouse
49              public void mousePressed( MouseEvent event )
50              {
51                 checkForTriggerEvent( event ); // verifica o acionamento
52              } // fim do método mousePressed
53
54              // trata eventos de liberação de botão do mouse
55              public void mouseReleased( MouseEvent event )
56              {
57                 checkForTriggerEvent( event ); // verifica o acionamento
58              } // fim do método mouseReleased
59
60              // determina se o evento deve acionar o menu de pop-up
61              private void checkForTriggerEvent( MouseEvent event )
62              {
63                 if ( event.isPopupTrigger() )
64                    popupMenu.show(
65                       event.getComponent(), event.getX(), event.getY() );
66              } // fim do método checkForTriggerEvent
67           } // fim da classe interna anônima
68        ); // fim da chamada para addMouseListener
69     } // fim do construtor PopupFrame
70
```

Figura 25.7 | `JPopupMenu` para selecionar cores. (Parte 3 de 4.)


```
71       // classe interna privada para tratar eventos de item de menu
72       private class ItemHandler implements ActionListener
73       {
74          // processa seleções de itens 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 ItemHandler
88    } // fim da classe PopupFrame
```

Figura 25.7 | JPopupMenu para selecionar cores. (Parte 4 de 4.)

```
1  // Figura 25.8: 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 de main
14 } // fim da classe PopupTest
```

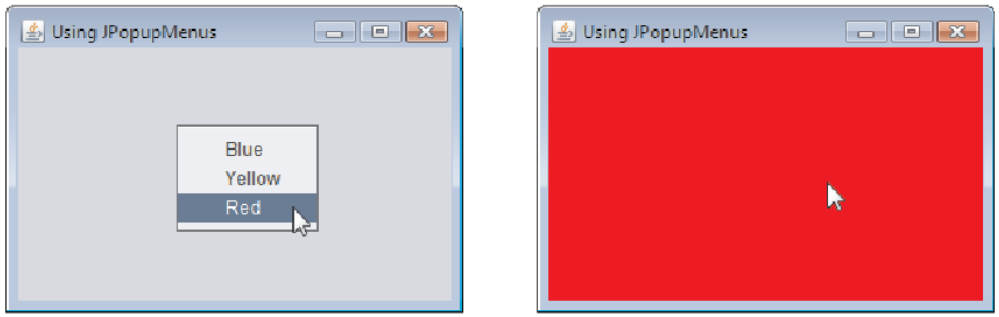


Figura 25.8 | Classe de teste para PopupFrame.

Observação sobre a aparência e o comportamento 25.9

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.*

25.6 Aparência e comportamento plugável

- Componentes AWT GUI do Java (pacote `java.awt`) assumem a aparência e comportamento da plataforma em que o programa executa.
- Introduz questões de portabilidade interessantes.
- Componentes GUI leves do Swing fornecem funcionalidade uniforme pelas plataformas e definem uma aparência e comportamento uniformes por diversas plataformas.
- A partir do Java SE 6 Update 10, essa é a aparência e funcionamento do Nimbus (Seção 14.2).
- As versões anteriores do Java utilizavam a **aparência e comportamento metal**, que ainda são o padrão.
- Podem personalizar a aparência e comportamento.
- As aparências e funcionamentos instalados irão variar entre uma e outra plataforma.

Dica de portabilidade 25.1

Os componentes GUI têm uma aparência distinta em diferentes plataformas e talvez requeiram quantidades diferentes de espaço para serem exibidos. Isso poderia alterar os layouts e alinhamentos da GUI.

Dica de portabilidade 25.2

Os componentes GUI em plataformas distintas têm funcionalidade padrão diferente (por exemplo, algumas plataformas permitem que um botão com o foco seja "pressionado" com a barra de espaço e outras não).


```
1  // Figura 25.9: LookAndFeelFrame.java
2  // Alterando a aparência e o 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    private UIManager.LookAndFeelInfo[] looks; // aparências e comportamentos
21    private String[] lookNames; // nomes das aparências e comportamentos
22    private JRadioButton[] radio; // botões de opção para selecionar a aparência e comportamento
23    private ButtonGroup group; // grupo de botões de opção
24    private JButton button; // exibe a aparência do botão
```

Figura 25.9 | Aparência e comportamento de uma GUI baseada no Swing. (Parte 1 de 5.)

```
25       private JLabel label; // exibe a aparência do rótulo
26       private JComboBox comboBox; // exibe a aparência da caixa de combinação
27
28       // configura a GUI
29       public LookAndFeelFrame()
30       {
31          super( "Look and Feel Demo" );
32
33          // obtém as informações sobre a aparência e o comportamento instalados
34          looks = UIManager.getInstalledLookAndFeels();
35          lookNames = new String[ looks.length ];
36
37          // obtém os nomes das aparências e funcionamentos instalados
38          for ( int i = 0; i < looks.length; i++ )
39             lookNames[ i ] = looks[ i ].getName();
40
41          JPanel northPanel = new JPanel(); // cria o painel North
42          northPanel.setLayout( new GridLayout( 3, 1, 0, 5 ) );
43
44          label = new JLabel( "This is a " + lookNames[0] + " look-and-feel",
45             SwingConstants.CENTER ); // cria o rótulo
46          northPanel.add( label ); // adiciona o rótulo ao painel
47
48          button = new JButton( "JButton" ); // cria o botão
```

Figura 25.9 | Aparência e comportamento de uma GUI baseada no Swing. (Parte 2 de 5.)

```
49          northPanel.add( button ); // adiciona botão ao painel
50
51          comboBox = new JComboBox( lookNames ); // cria a caixa de combinação
52          northPanel.add( comboBox ); // adiciona a caixa de combinação ao painel
53
54          // cria um array para botões de opção
55          radio = new JRadioButton[ looks.length ];
56
57          JPanel southPanel = new JPanel(); // cria o painel South
58
59          // utiliza um GridLayout com 3 botões em cada linha
60          int rows = (int) Math.ceil( radio.length / 3.0 );
61          southPanel.setLayout( new GridLayout( rows, 3 ) );
62
63          group = new ButtonGroup(); // grupo de botões para aparências e comportamentos
64          ItemHandler handler = new ItemHandler(); // handler de aparência e comportamento
65
66          for ( int count = 0; count < radio.length; count++ )
67          {
68             radio[ count ] = new JRadioButton( lookNames[ count ] );
69             radio[ count ].addItemListener( handler ); // adiciona handler
70             group.add( radio[ count ] ); // adiciona o botão de opção ao grupo
71             southPanel.add( radio[ count ] ); // adiciona botão de opções ao painel
72          } // for final
```

Figura 25.9 | Aparência e comportamento de uma GUI baseada no Swing. (Parte 3 de 5.)

```
73
74          add( northPanel, BorderLayout.NORTH ); // adiciona o painel North
75          add( southPanel, BorderLayout.SOUTH ); // adiciona o painel South
76
77          radio[ 0 ].setSelected( true ); // configura a seleção padrão
78       } // fim do construtor LookAndFeelFrame
79
80       // utiliza UIManager para alterar a aparência e o comportamento da GUI
81       private void changeTheLookAndFeel( int value )
82       {
83          try // muda a aparência e o comportamento
84          {
85             // configura a aparência e o comportamento para esse aplicativo
86             UIManager.setLookAndFeel( looks[ value ].getClassName() );
87
88             // atualiza os componentes nesse aplicativo
89             SwingUtilities.updateComponentTreeUI( this );
90          } // fim do try
91          catch ( Exception exception )
92          {
93             exception.printStackTrace();
94          } // fim do catch
95       } // fim do método changeTheLookAndFeel
```

Figura 25.9 | Aparência e comportamento de uma GUI baseada no Swing. (Parte 4 de 5.)

```
96
97       // classe interna private para tratar eventos de botão de opção
98       private class ItemHandler implements ItemListener
99       {
100         // processa a seleção de aparência e comportamento feita pelo usuário
101         public void itemStateChanged( ItemEvent event )
102         {
103            for ( int count = 0; count < radio.length; count++ )
104            {
105               if ( radio[ count ].isSelected() )
106               {
107                  label.setText( String.format(
108                     "This is a %s look-and-feel", lookNames[ count ] ) );
109                  comboBox.setSelectedIndex( count ); // configura o índice da caixa de combinação
110                  changeTheLookAndFeel( count ); // muda a aparência e o comportamento
111               } // fim do if
112            } // fim do for
113         } // fim do método itemStateChanged
114      } // fim da classe interna privada ItemHandler
115   } // fim da classe LookAndFeelFrame
```

Figura 25.9 | Aparência e comportamento de uma GUI baseada no Swing. (Parte 5 de 5.)


```
1  // Figura 25.10: LookAndFeelDemo.java
2  // Alterando 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( 400, 220 ); // configura o tamanho do frame
12       lookAndFeelFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe LookAndFeelDemo
```

Figura 25.10 | Classe de teste para LookAndFeelFrame. (Parte 1 de 3.)

Java™ COMO PROGRAMAR 8ª edição

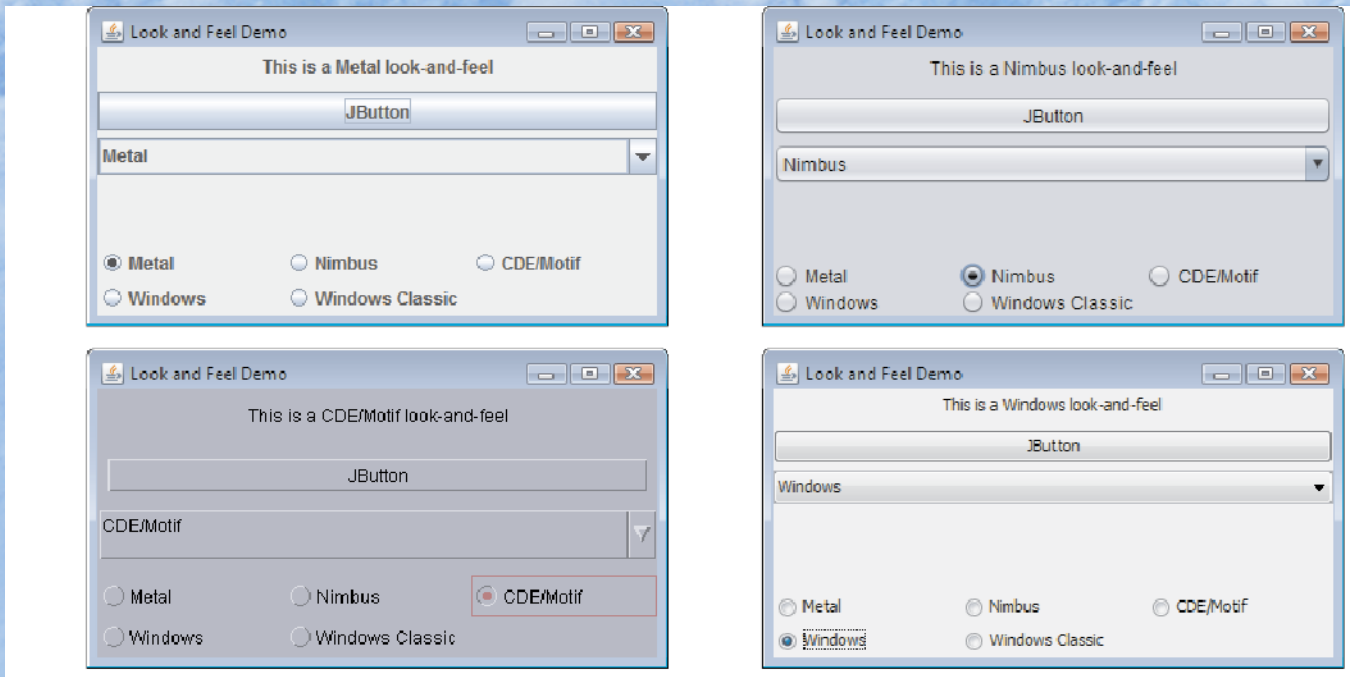


Figura 25.10 | Classe de teste para `LookAndFeelFrame`. (Parte 2 de 3.)

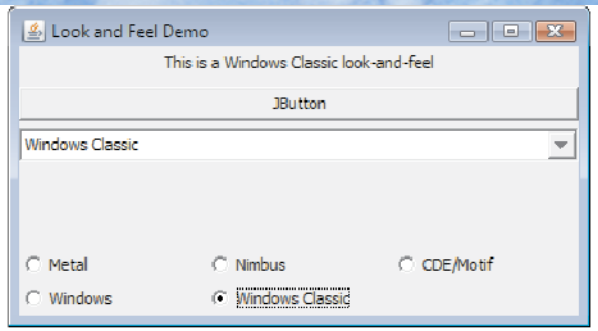


Figura 25.10 | Classe de teste para `LookAndFeelFrame`. (Parte 3 de 3.)

- A classe **UIManager** (pacote `javax.swing`) contém a(uma) classe (aninhada) **LookAndFeelInfo** aninhada(cortar) (uma classe `public static`) que mantém informações sobre a aparência e comportamento.
- O método **getInstalledLookAndFeels** da classe `UIManager` `static` obtém um array de objetos `UIManager.LookAndFeelInfo` que descrevem cada aparência e comportamento disponível no seu sistema.
- O método **setLookAndFeel** estático da classe UIManager muda a aparência e comportamento.
- O método **getClassName** da classe `UIManager.LookAndFeelInfo` determina o nome da classe de aparência e comportamento que corresponda ao objeto `UIManager.LookAndFeelInfo`.
- O método **updateComponentTreeUI** estático do SwingUtilities muda a aparência e comportamento de cada componente associado ao seu argumento Component para a nova aparência e comportamento.

Dica de desempenho 25.2

Cada aparência e comportamento é representada 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.*

25.7 JDesktopPane e JInternalFrame

- **Interface de múltiplos documentos** (**multiple document interface – MDI**) – uma janela principal (chamada de **janela-pai**) contendo outras janelas (chamadas de **janelas-filhas**), para lidar com vários documentos abertos que estão em processamento paralelo.
- As classes **JDesktopPane** e **JInternalFrame** do Swing implementam interfaces de vários documentos.


```
1  // Figura 25.11: 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    {
```

Figura 25.11 | Interface de múltiplos documentos. (Parte 1 de 4.)


```
25          super( "Using a JDesktopPane" );
26
27          JMenuBar bar = new JMenuBar(); // cria a barra de menus
28          JMenu addMenu = new JMenu( "Add" ); // cria o menu Add
29          JMenuItem newFrame = new JMenuItem( "Internal Frame" );
30
31          addMenu.add( newFrame ); // adiciona um novo item de quadro ao menu Add
32          bar.add( addMenu ); // adiciona o menu Add à barra de menus
33          setJMenuBar( bar ); // configura a barra de menus para esse aplicativo
34
35          theDesktop = new JDesktopPane(); // cria o painel de área de trabalho
36          add( theDesktop ); // adiciona painel de área de trabalho ao quadro
37
38          // configura o ouvinte para o item de menu newFrame
39          newFrame.addActionListener(
40
41             new ActionListener() // classe interna anônima
42             {
43                // exibe a nova janela interna
44                public void actionPerformed( ActionEvent event )
45                {
46                   // cria o quadro interno
47                   JInternalFrame frame = new JInternalFrame(
48                      "Internal Frame", true, true, true, true );
```

Figura 25.11 | Interface de múltiplos documentos. (Parte 2 de 4.)

```
49
50                MyJPanel panel = new MyJPanel(); // cria um novo painel
51                frame.add( panel, BorderLayout.CENTER ); // adiciona o painel
52                frame.pack(); // configura o quadro interno segundo o tamanho do conteúdo
53
54                theDesktop.add( frame ); // anexa o quadro interno
55                frame.setVisible( true ); // mostra o quadro 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 final static String[] images = { "yellowflowers.png",
68       "purpleflowers.png", "redflowers.png", "redflowers2.png",
69       "lavenderflowers.png" };
70
```

Figura 25.11 | Interface de múltiplos documentos. (Parte 3 de 4.)

```
71       // carrega a imagem
72       public MyJPanel()
73       {
74          int randomNumber = generator.nextInt( images.length );
75          picture = new ImageIcon( images[ randomNumber ] ); // configura o ícone
76       } // fim do construtor MyJPanel
77
78       // exibe imageIcon no painel
79       public void paintComponent( Graphics g )
80       {
81          super.paintComponent( g );
82          picture.paintIcon( this, g, 0, 0 ); // exibe o ícone
83       } // fim do método paintComponent
84
85       // retorna as dimensões da imagem
86       public Dimension getPreferredSize()
87       {
88          return new Dimension( picture.getIconWidth(),
89             picture.getIconHeight() );
90       } // fim do método getPreferredSize
91    } // fim da classe MyJPanel
```

Figura 25.11 | Interface de múltiplos documentos. (Parte 4 de 4.)


```
1  // Figura 25.12: 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 de main
14 } // fim da classe DesktopTest
```

Figura 25.12 | Classe de teste para DeskTopFrame. (Parte 1 de 3.)

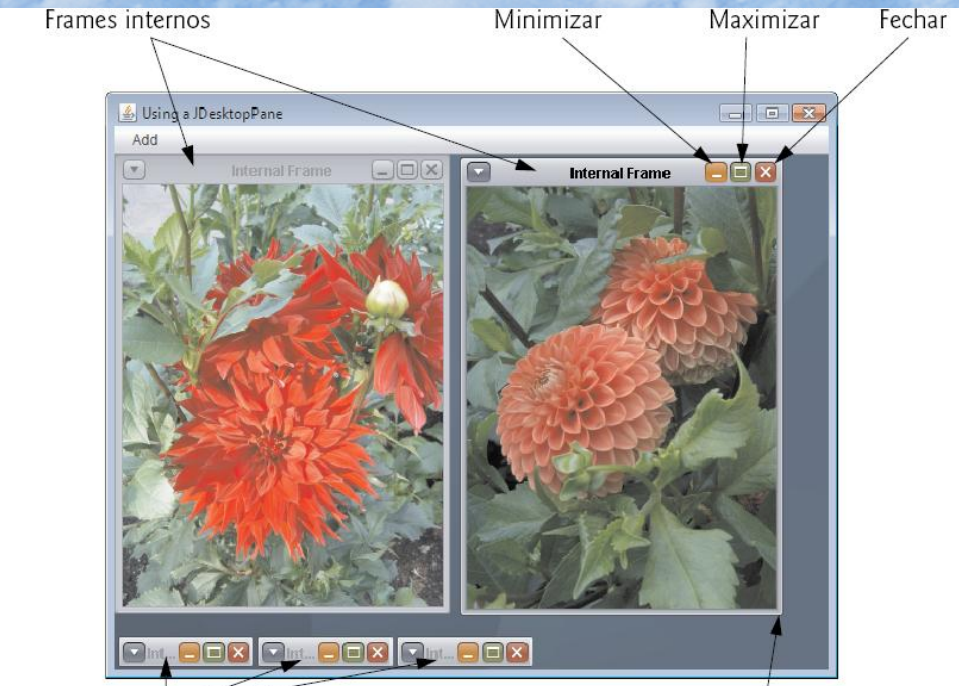


Figura 25.12 | Classe de teste para `DeskTopFrame`. (Parte 2 de 3.)

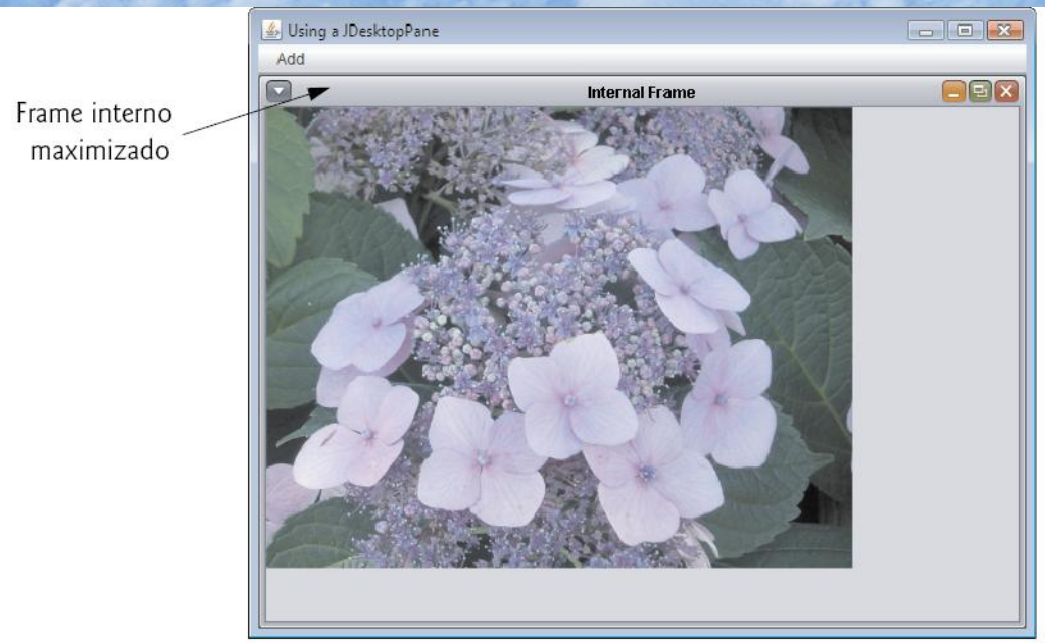


Figura 25.12 | Classe de teste para `DeskTopFrame`. (Parte 3 de 3.)

- O construtor `JInternalFrame` usado aqui aceita cinco argumentos
  - uma `String` para a barra de título da janela interna.
  - um `boolean` que indica se o quadro interno pode ser redimensionado pelo usuário.
  - um `boolean` que indica se o quadro interno pode ser fechado pelo usuário.
  - um `boolean` que indica se o quadro interno pode ser maximizado pelo usuário.
  - um `boolean` que indica se o quadro interno pode ser minimizado pelo usuário.
- Para cada um dos argumentos `boolean`, um valor `true` indica se a operação deve ser permitida (como é caso aqui).

- Um `JInternalFrame` tem um painel de conteúdo ao qual componentes GUI podem ser anexados.
- O método `JInternalFrame` **pack** configura o tamanho da janela-filha.
- Utiliza os tamanhos preferidos dos componentes para determinar o tamanho da janela.
- As classes `JInternalFrame` e `JDesktopPane` fornecem muitos métodos para gerenciar janelas-filhas.

25.8 JTabbedPane

- Um **JTabbedPane** organiza componentes GUI em camadas, das quais somente uma é visível por vez.
- Usuários acessam cada camada clicando na guia.
- As guias aparecem na parte superior por padrão, mas também podem ser posicionadas à esquerda, direita ou parte inferior do `JTabbedPane`.
- Qualquer componente pode ser posicionado em uma guia.
- Se o componente for um contêiner, como um painel, ele poderá utilizar qualquer gerenciador de layout para organizar vários componentes na guia.
- A classe `JTabbedPane` é uma subclasse de `JComponent`.


```
1  // Figura 25.13: JTabbedPaneFrame.java
2  // Demonstrando o 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    // configura a GUI
15    public JTabbedPaneFrame()
16    {
17       super( "JTabbedPane Demo " );
18
19       JTabbedPane tabbedPane = new JTabbedPane(); // cria o 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
```

Figura 25.13 | JTabbedPane utilizado para organizar componentes GUI. (Parte 1 de 2.)


```
24          panel1.add( label1 ); // adiciona o rótulo ao painel
25          tabbedPane.addTab( "Tab One", null, panel1, "First Panel" );
26
27          // configura o panel2 e o adiciona a JTabbedPane
28          JLabel label2 = new JLabel( "panel two", SwingConstants.CENTER );
29          JPanel panel2 = new JPanel(); // cria o segundo painel
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 a JTabbedPane
35          JLabel label3 = new JLabel( "panel three" );
36          JPanel panel3 = new JPanel(); // cria o terceiro painel
37          panel3.setLayout( new BorderLayout() ); // utilize 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 ); // adiciona o JTabbedPane ao quadro
46       } // fim do construtor JTabbedPaneFrame
47    } // fim da classe JTabbedPaneFrame
```

Figura 25.13 | JTabbedPane utilizado para organizar componentes GUI. (Parte 2 de 2.)


```
1  // Figura 25.14: 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 de main
14 } // fim da classe JTabbedPaneDemo
```

Figura 25.14 | Classe de teste para `JTabbedPaneFrame`. (Parte 1 de 2.)

Java™ COMO PROGRAMAR 8ª edição

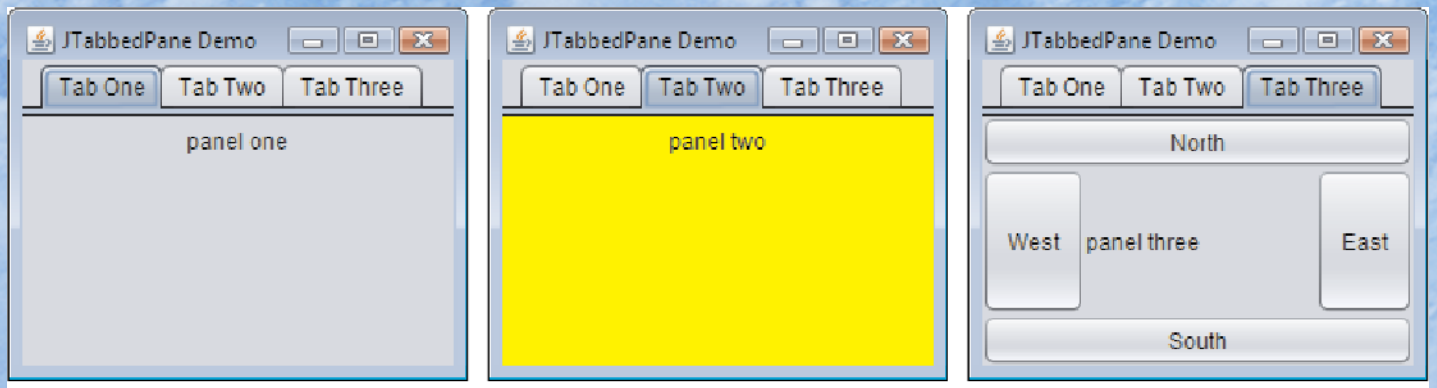


Figura 25.14 | Classe de teste para `JTabbedPaneFrame`. (Parte 2 de 2.)

- O método `JTabbedPane` **addTab** adiciona uma nova guia. Na versão com quatro argumentos:
  - O primeiro argumento é uma `String` que especifica o título da guia.
  - O segundo é uma referência `Icon` que especifica um ícone a ser exibido na guia — pode ser `null`.
  - O terceiro é um `Component` a ser exibido quando o usuário clicar na guia.
  - O último é uma `String` que especifica a dica de ferramenta da guia.

COMO PROGRAMAR

8ª edição

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

- Esta seção apresenta dois gerenciadores adicionais de layout (resumidos na Figura 25.15).

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 os componentes podem variar de tamanho e podem ser adicionados em qualquer ordem.

Figura 25.15 | Gerenciadores de layout adicionais.

- O gerenciador `BoxLayout` (no pacote `javax.swing`) organiza os componentes GUI horizontalmente ao longo do eixo *x* ou verticalmente ao longo do eixo *y* de um contêiner.


```
1  // Figura 25.16: 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
```

Figura 25.16 | Gerenciador de layout `BoxLayout`. (Parte 1 de 4.)

```
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
30          // cria um suporte e adiciona botões a Box vertical1
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 a cola horizontal e adiciona botões a Box horizontal2
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
```

Figura 25.16 | Gerenciador de layout BoxLayout. (Parte 2 de 4.)

```
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
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
```

Figura 25.16 | Gerenciador de layout `BoxLayout`. (Parte 3 de 4.)


```
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 quadro
73       } // fim do construtor BoxLayoutFrame
74    } // fim da classe BoxLayoutFrame
```

Figura 25.16 | Gerenciador de layout `BoxLayout`. (Parte 4 de 4.)


```
1  // Figura 25.17: 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 de main
14 } // fim da classe BoxLayoutDemo
```

Figura 25.17 | Classe de teste para BoxLayoutFrame. (Parte 1 de 2.)

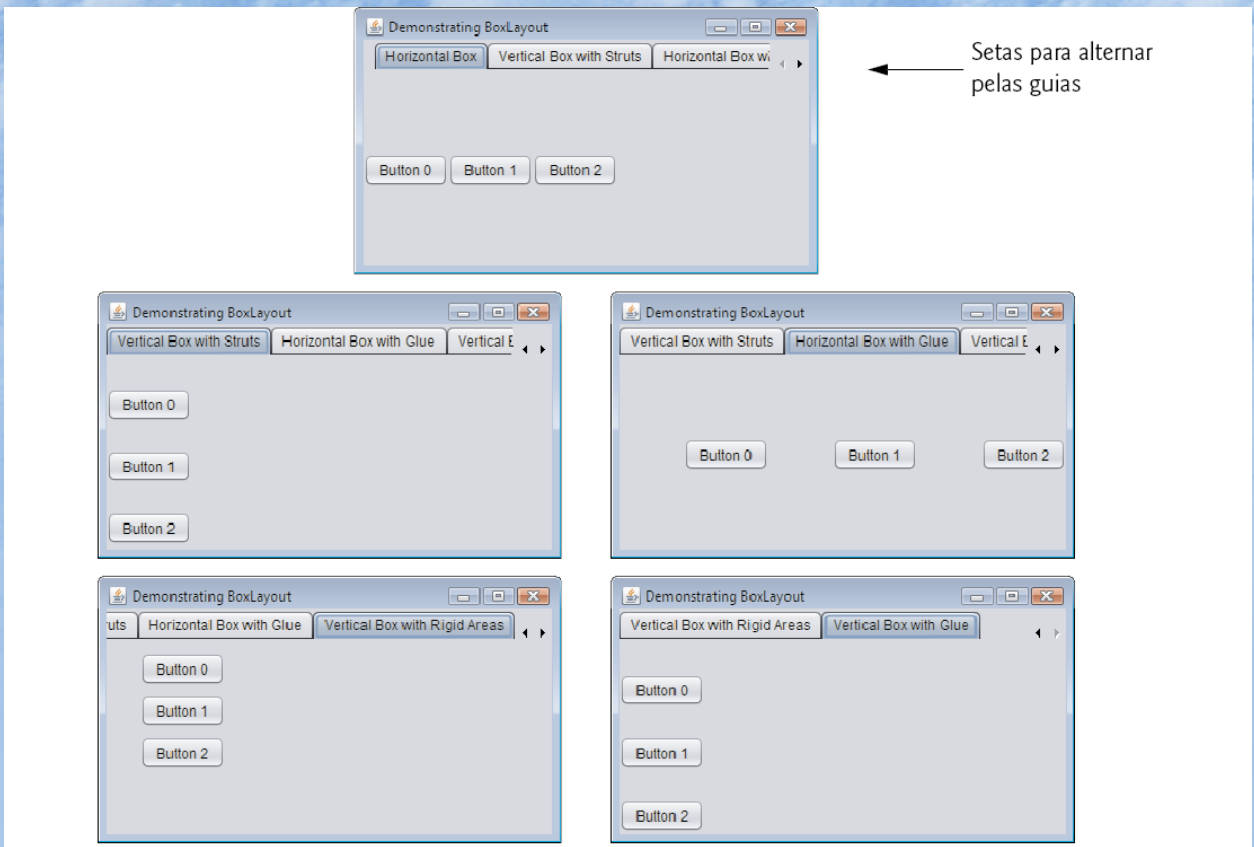


Figura 25.17 | Classe de teste para `BoxLayoutFrame`. (Parte 2 de 2.)

- O método `static` `Box` **createVerticalBox** retorna referências aos contêineres `Box` com um `BoxLayout` vertical no qual componentes GUI são organizados de cima para baixo.
- Antes de adicionar cada botão, a linha 33 adiciona uma **estrutura vertical** ao contêiner com um método `static` **createVerticalBox** da classe `Box`.
- Uma **estrutura vertical** é um componente GUI invisível que tem uma altura fixa em pixels e é utilizada para garantir uma quantidade fixa de espaço entre componentes GUI.
- Criada com o método `static` `Box` **createVerticalStrut**.
- Quando o contêiner é redimensionado, a distância entre os componentes GUI separados por estruturas (struts) não muda.
- A classe `Box` também declara o método **createHorizontalStrut** para `BoxLayout`s horizontais.

- A **cola horizontal** é um componente GUI invisível que pode ser utilizado entre componentes GUI de tamanho fixo para ocupar espaço adicional.
- Criado com o método `static` da classe `Box` **createVerticalStrut.**
- Quando o contêiner é redimensionado, componentes separados por componentes de cola permanecem no mesmo tamanho, mas a cola se expande ou se contrai para ocupar o espaço entre eles.
- A classe `Box` também declara o método **createVerticalGlue** para `BoxLayouts` verticais.

- Uma **área rígida** é um componente GUI invisível que tem sempre a largura e altura fixas em pixels.
- Criada com o método `static` da classe `Box` **createRigidArea**
- O construtor `BoxLayout` recebe uma referência ao contêiner cujo layout ele controla, e uma constante indicando se o layout é horizontal (**BoxLayout.X_AXIS**) ou vertical (**BoxLayout.Y_AXIS**).
- O método `static` da classe `Box` **createGlue** cria um componente que expande ou contrai com base no tamanho de `Box`.
- **JTabbedPane.TOP** enviado ao construtor indica que as guias devem aparecer na parte superior do `JTabbedPane`.
- **JTabbedPane.SCROLL_TAB_LAYOUT** especifica que as guias devem recorrer para uma nova linha se houver muitos espaços para se ajustarem em uma única linha.

COMO PROGRAMAR

8ª edição

- Um dos gerenciadores de layout predefinidos mais poderosos **é GridBagLayout** (no pacote `java.awt`).
- Semelhante ao `GridLayout`, mas muito mais flexível.
- Os componentes podem variar de tamanho e podem ser adicionados em qualquer ordem.

- O primeiro passo na utilização de `GridBagLayout` é determinar a aparência da GUI.
- Desenhe a GUI e depois desenhe uma grade sobre ela, dividindo os componentes em linhas e colunas.
- Os números iniciais de linhas e colunas devem ser 0, de modo que o gerenciador de layout `GridBagLayout` possa utilizar os números de linha e coluna para posicionar adequadamente os componentes na grade.
- A Figura 25.18 demonstra como desenhar linhas e colunas sobre uma GUI.

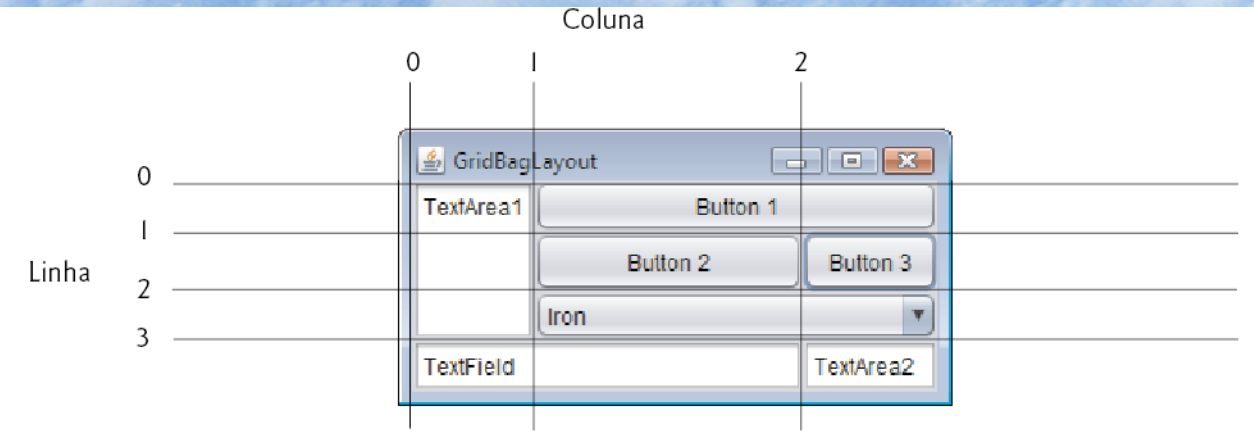


Figura 25.18 | Projetando uma GUI que utilizará `GridBagLayout`.

- Um objeto **GridBagConstraints** descreve como um componente é posicionado em um `GridBagLayout`.
- Vários campos `GridBagConstraints` estão resumidos na Figura 25.19.
- O método `GridBagLayout` **setConstraints** aceita um argumento `Component` e um argumento `GridBagConstraints`.

Campo	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 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 25.19 | Campos `GridBagConstraints`.

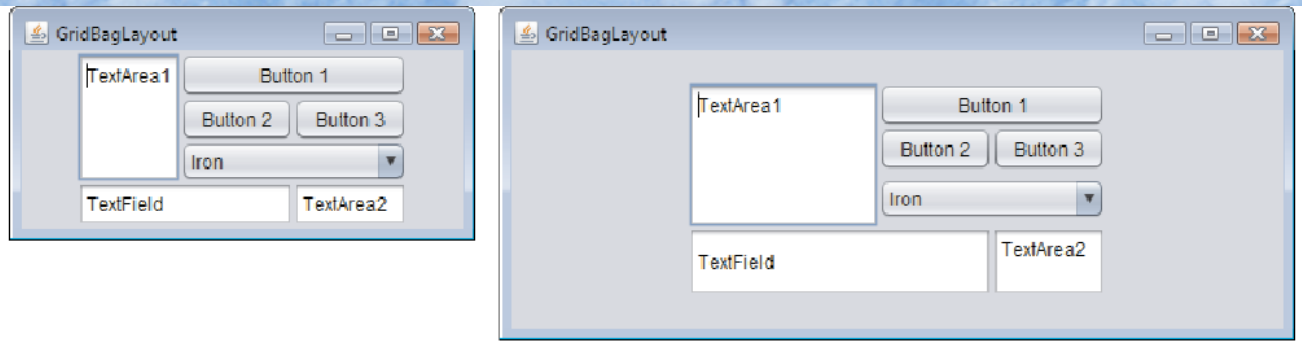


Figura 25.20 | `GridBagLayout` com os pesos configurados como zero.

```
1  // Figura 25.21: 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 quadro
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
```

Figura 25.21 | Gerenciador de layout `GridBagLayout`. (Parte 1 de 4.)

```
25          // cria componentes GUI
26          JTextArea textArea1 = new JTextArea( "TextArea1", 5, 10 );
27          JTextArea textArea2 = new JTextArea( "TextArea2", 2, 2 );
28
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 padrão
43          constraints.fill = GridBagConstraints.HORIZONTAL;
44          addComponent( button1, 0, 1, 2, 1 );
45
46          // weightx e weighty para comboBox são 0: o padrão
47          // fill é HORIZONTAL
48          addComponent( comboBox, 2, 1, 2, 1 );
```

Figura 25.21 | Gerenciador de layout `GridBagLayout`. (Parte 2 de 4.)

```
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
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 );
```

Figura 25.21 | Gerenciador de layout `GridBagLayout`. (Parte 3 de 4.)

```
66       } // fim do construtor GridBagFrame
67
68       // método para configurar restrições em
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; // configure gridheight
76          layout.setConstraints( component, constraints ); // configura constraints
77          add( component ); // adiciona componente
78       } // fim do método addComponent
79    } // fim da classe GridBagFrame
```

Figura 25.21 | Gerenciador de layout `GridBagLayout`. (Parte 4 de 4.)


```
1  // Figura 25.22: 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 o tamanho do frame
12       gridBagFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe GridBagDemo
```

Figura 25.22 | Classe de teste para `GridBagFrame`. (Parte 1 de 2.)

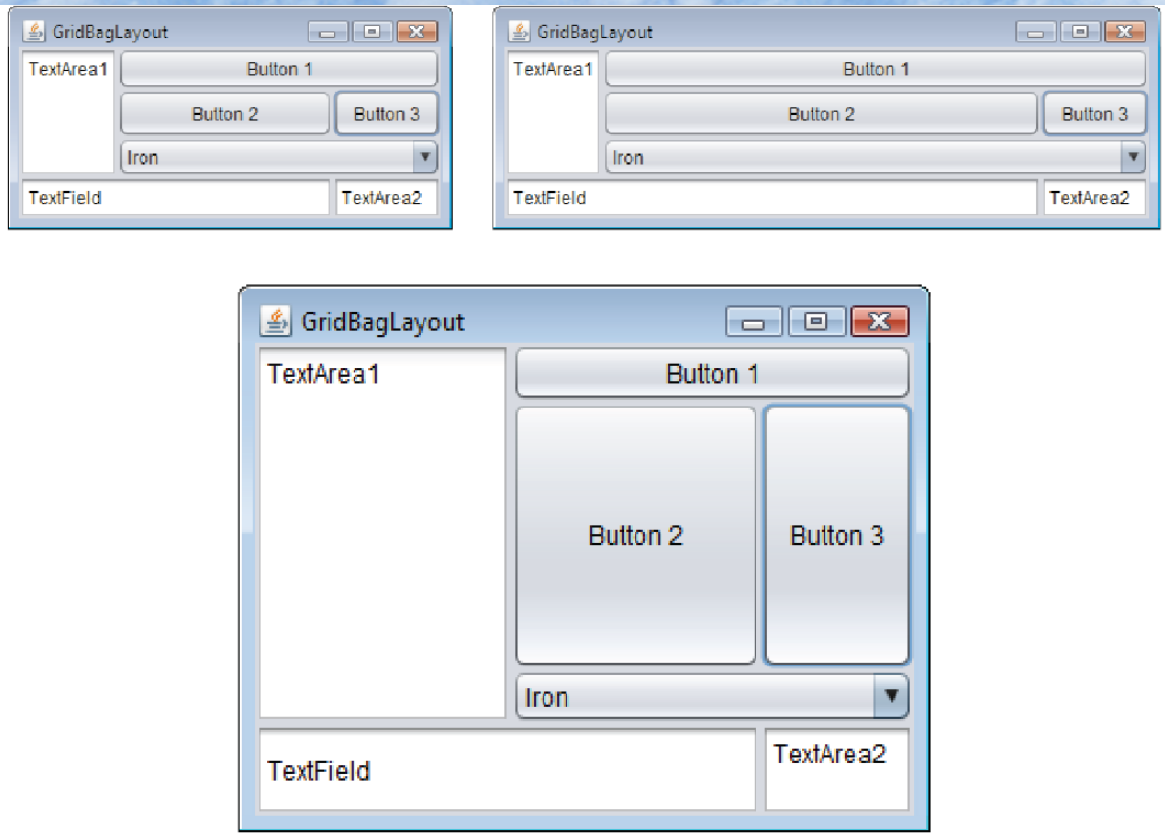


Figura 25.22 | Classe de teste para `GridBagFrame`. (Parte 2 de 2.)

- Uma variação de `GridBagLayout` usa as constantes `GridBagConstraints` **RELATIVE** e **REMAINDER**.
- `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.


```
1  // Figura 25.23: 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 quadro
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 o layout de frame
23       constraints = new GridBagConstraints(); // instancia restrições
```

Figura 25.23 | Constantes `GridBagConstraints` `RELATIVE` e `REMAINDER`. (Parte 1 de 4.)


```
24
25          // cria componentes GUI
26          String[] metals = { "Copper", "Aluminum", "Silver" };
27          JComboBox comboBox = new JComboBox( metals );
28
29          JTextField textField = new JTextField( "TextField" );
30
31          String[] fonts = { "Serif", "Monospaced" };
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 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
```

Figura 25.23 | Constantes `GridBagConstraints` `RELATIVE` e `REMAINDER`. (Parte 2 de 4.)


```
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
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
```

Figura 25.23 | Constantes `GridBagConstraints` `RELATIVE` e `REMAINDER`. (Parte 3 de 4.)


```
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
```

Figura 25.23 | Constantes `GridBagConstraints` `RELATIVE` e `REMAINDER`. (Parte 4 de 4.)


```
1  // Figura 25.24: GridBagDemo2.java
2  // Demonstrando as constantes 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 de main
14 } // fim da classe GridBagDemo2
```

Figura 25.24 | Classe de teste para `GridBagDemo2`. (Parte 1 de 2.)

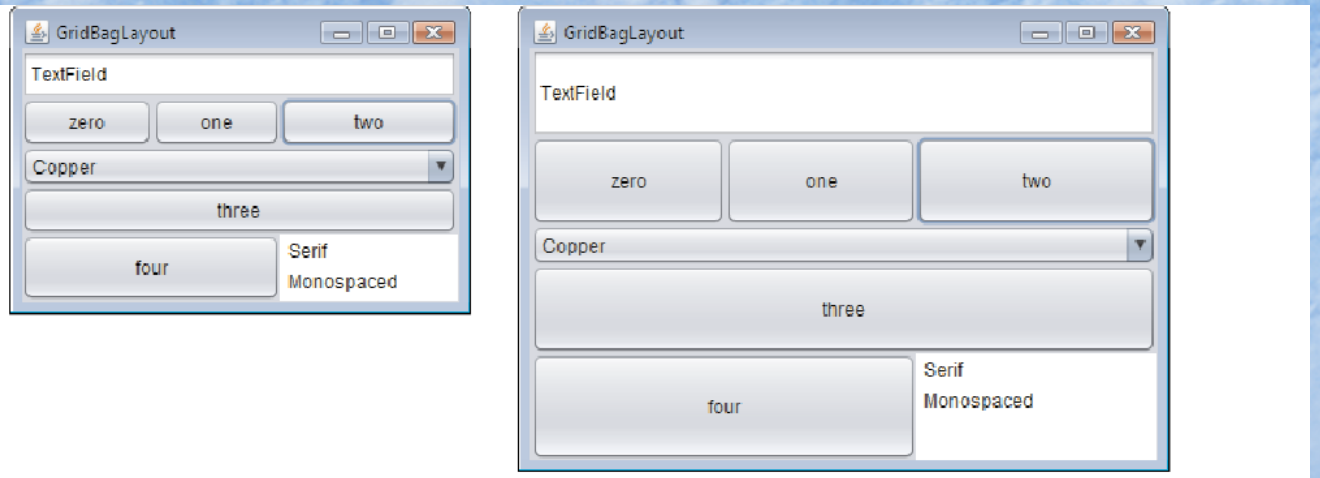


Figura 25.24 | Classe de teste para `GridBagDemo2`. (Parte 2 de 2.)