11

Componentes GUI: Parte 1

OBJETIVOS

- Neste capítulo, você aprenderá:
- Os princípios do projeto de interfaces gráficas com o usuário (*graphical user interfaces* – GUIs).
- Como construir GUIs e tratar eventos gerados por interações de usuário com GUIs.
- Como entender os pacotes que contêm componentes GUI, interfaces e classes de tratamento de evento.
- Como criar e manipular botões, rótulos, listas, campos de texto e painéis.
- Como tratar eventos de mouse e eventos de teclado.
- Como utilizar gerenciadores de layout para organizar componentes GUI.

11.1 Introdução

- **Interface gráfica com o usuário (*graphical user interface* – GUI):**
  - **Apresenta um mecanismo amigável ao usuário para interagir com uma aplicação.**
  - **Freqüentemente contém barra de título, barra de menus que contém menus, botões e caixas de combinação.**
  - **É construída a partir de componentes GUI.**

Observação sobre aparência e comportamento 11.1

Interfaces com o usuário consistentes permitem que o usuário aprenda mais rápido novos aplicativos.

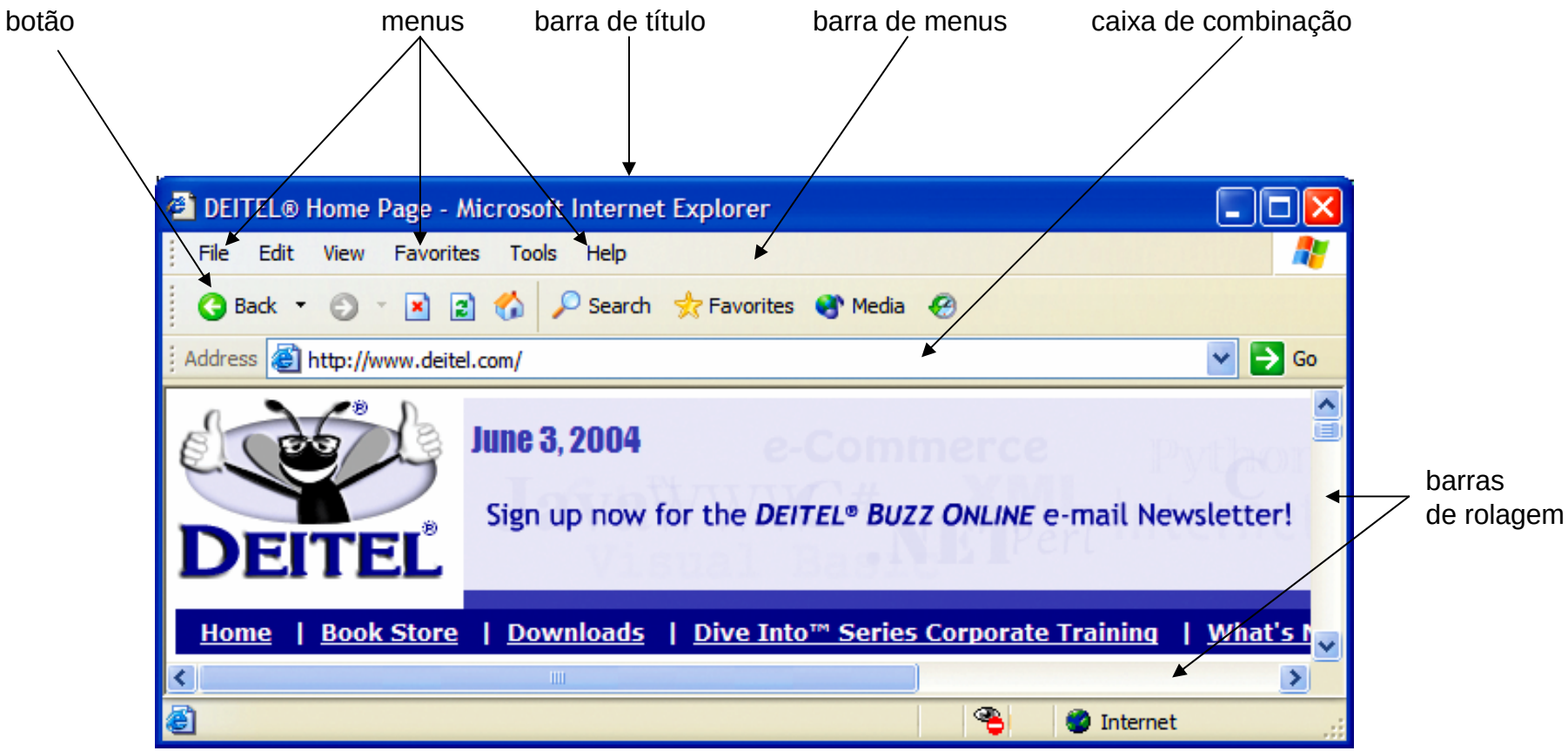


Figura 11.1 | Janela do Internet Explorer com componentes GUI.

11.2 Entrada/saída baseada em GUI simples com `JOptionPane`

- **Caixas de diálogo:**
  - **Utilizadas pelas aplicações para interagir com o usuário.**
  - **Fornecidas pela classe `JOptionPane` do Java (pacote `javax.swing`).**
    - **Contém diálogos de entrada e diálogos de mensagem.**

Resumo

```java
// Fig. 11.2: Addition.java
// Programa de adição que utiliza JOptionPane para entrada e saída.
import javax.swing.JOptionPane; // programa utiliza JOptionPane

public class Addition
{
   public static void main( String args[] )
   {
      // obtém a entrada de usuário a partir dos diálogos de entrada JOptionPane
      String firstNumber =
         JOptionPane.showInputDialog( "Enter first integer" );
      String secondNumber =
          JOptionPane.showInputDialog( "Enter second integer" );

      // converte String em valores int para utilização em um cálculo
      int number1 = Integer.parseInt( firstNumber );
      int number2 = Integer.parseInt( secondNumber );

      int sum = number1 + number2; // adiciona números

      // exibe o resultado em um diálogo de mensagem JOptionPane
      JOptionPane.showMessageDialog( null, "The sum is " + sum,
         "Sum of Two Integers", JOptionPane.PLAIN_MESSAGE );
   } // fim do método  main
} // fim da classe Addition
```

Mostra o diálogo de entrada para receber o primeiro inteiro

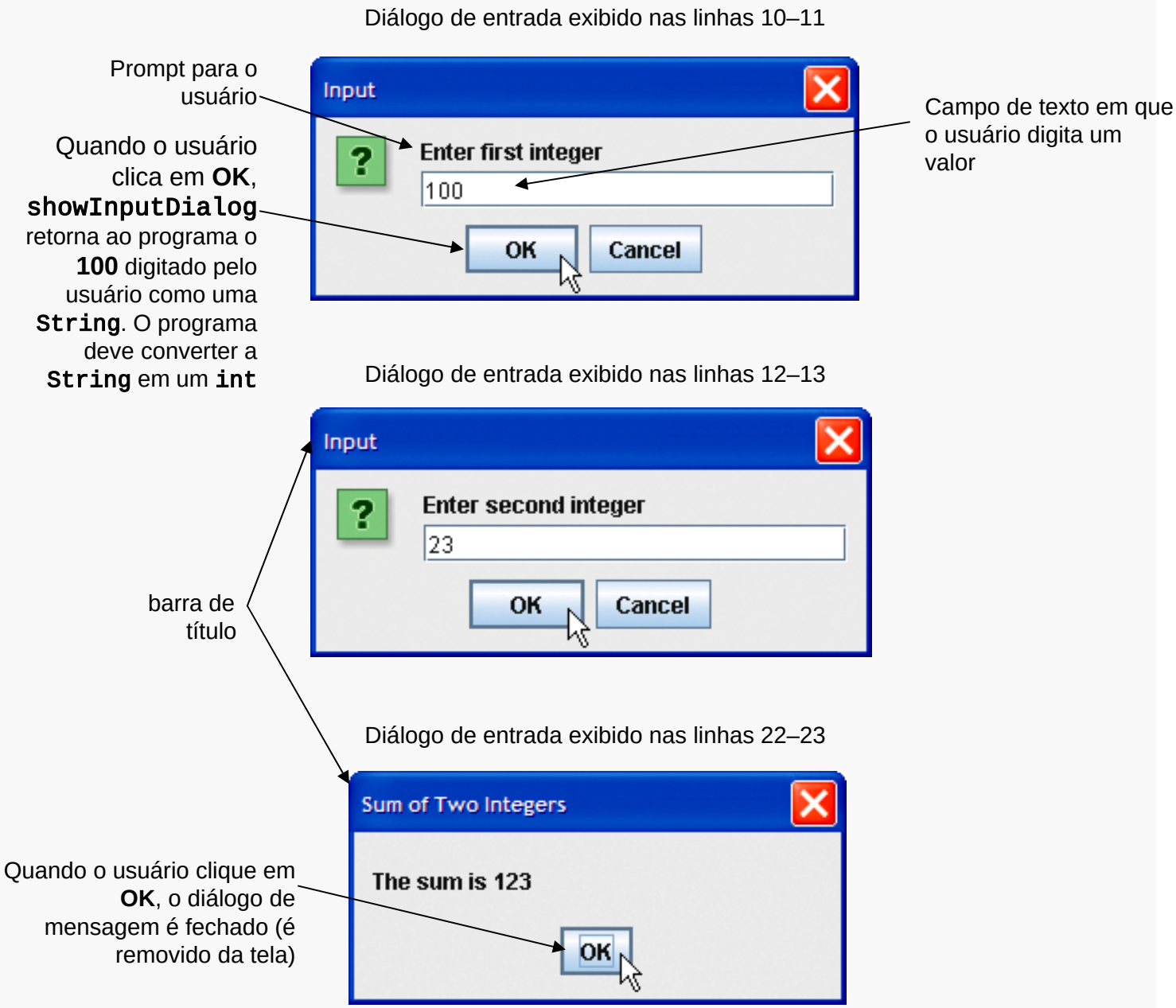
Mostra o diálogo de entrada para receber o segundo inteiro

Mostra o diálogo de mensagem para gerar a saída da soma para o usuário

Resumo

`Addition.java`

(2 de 2)



Observação sobre aparência e comportamento 11.2

O prompt em um diálogo de entrada emprega maiúsculas e minúsculas *no estilo de frases* — um estilo que emprega a maiúscula inicial apenas na primeira palavra da frase a menos que a palavra seja um nome próprio (por exemplo, Deitel).

Observação sobre aparência e comportamento 11.3

Em geral, a barra de título de uma janela adota o uso de letras maiúsculas e minúsculas de *título de livro* — um estilo que emprega a inicial maiúscula em cada palavra significativa no texto e não termina com pontuação (por exemplo, Uso de Letras Maiúsculas e Minúsculas no Título de um Livro).

Tipo de diálogo de mensagem	Ícone	Descrição
`ERROR_MESSAGE`		Um diálogo que indica um erro para o usuário.
`INFORMATION_MESSAGE`		Um diálogo com uma mensagem informativa para o usuário.
`WARNING_MESSAGE`		Um diálogo que adverte o usuário de um problema potencial.
`QUESTION_MESSAGE`		Um diálogo que impõe uma pergunta ao usuário. Normalmente, esse diálogo exige uma resposta, como clicar em um botão `Yes` ou `No`.
`PLAIN_MESSAGE`	Nenhum ícone	Um diálogo que contém uma mensagem, mas nenhum ícone..

Figura 11.3 | Constantes `JOptionPane` `static` para diálogos de mensagem.

11.3 Visão geral de componentes Swing

- **Componentes Swing GUI:**
  - **Declarado no pacote `javax.swing`.**
  - **A maioria dos componentes Swing são componentes *Java puros* — escritos, manipulados e exibidos em Java.**
  - **Fazem parte das Java Foundation Classes (JFC) — bibliotecas do Java para desenvolvimento de GUI para múltiplas plataformas.**

Componente	Descrição
`JLabel`	Exibe texto não-editável ou ícones.
`JTextField`	Permite ao usuário inserir dados do teclado. Também pode ser utilizado para exibir texto editável ou não editável.
`JButton`	Desencadeia um evento quando o usuário clicar nele com o mouse.
`JCheckBox`	Especifica uma opção que pode ser ou não selecionada.
`JComboBox`	Fornece uma lista drop-down de itens a partir da qual o usuário pode fazer uma seleção clicando em um item ou possivelmente digitando na caixa.
`JList`	Fornece uma lista de itens a partir da qual o usuário pode fazer uma seleção clicando em qualquer item na lista. Múltiplos elementos podem ser selecionados.
`JPanel`	Fornece uma área em que os componentes podem ser colocados e organizados. Também pode ser utilizado como uma área de desenho para imagens gráficas.

Figura 11.4 | Alguns componentes GUI básicos.

Swing *versus* AWT

- **Abstract Window Toolkit (AWT):**
  - **Precursor do Swing.**
  - **Declarado no pacote `java.awt`.**
  - **Não fornece aparência e comportamento consistentes para diversas plataformas.**

Dica de portabilidade 11.1

Os componentes Swing são implementados no Java; desse modo, eles são mais portáveis e flexíveis do que os componentes Java GUI originais de pacotes `java.awt`, que foram baseados nos componentes GUI da plataforma subjacente. Por essa razão, os componentes Swing GUI geralmente são preferidos.

Componentes GUI leves *versus* pesados

- **Componentes leves:**
  - **Não associados diretamente a componentes GUI suportados pela plataforma subjacente.**
- **Componentes pesados:**
  - **Associados diretamente à plataforma local.**
  - **Componentes AWT.**
  - **Alguns componentes Swing.**

Observação sobre aparência e comportamento 11.4

A aparência e o comportamento de uma GUI definida com componentes GUI pesados no pacote `java.awt` podem variar entre plataformas. Como os componentes pesados são acoplados à GUI da plataforma local, a aparência e o comportamento variam entre plataformas.

Superclasses de componentes GUI leves do Swing

- **Classe `Component` (pacote `java.awt`):**
  - **Subclasse de `Object`.**
  - **Declara muitos comportamentos e atributos comuns a componentes GUI.**
- **Classe `Container` (pacote `java.awt`):**
  - **Subclasse de `Component`.**
  - **Organiza `Component`s.**
- **Classe `JComponent` (pacote `javax.swing`):**
  - **Subclasse de `Container`.**
  - **Superclasse de todos os componentes Swing leves.**

Observação de engenharia de software 11.1

Estude os atributos e comportamentos das classes na hierarquia de classe da Figura 11.5. Essas classes declaram os recursos que são comuns à maioria dos componentes Swing.

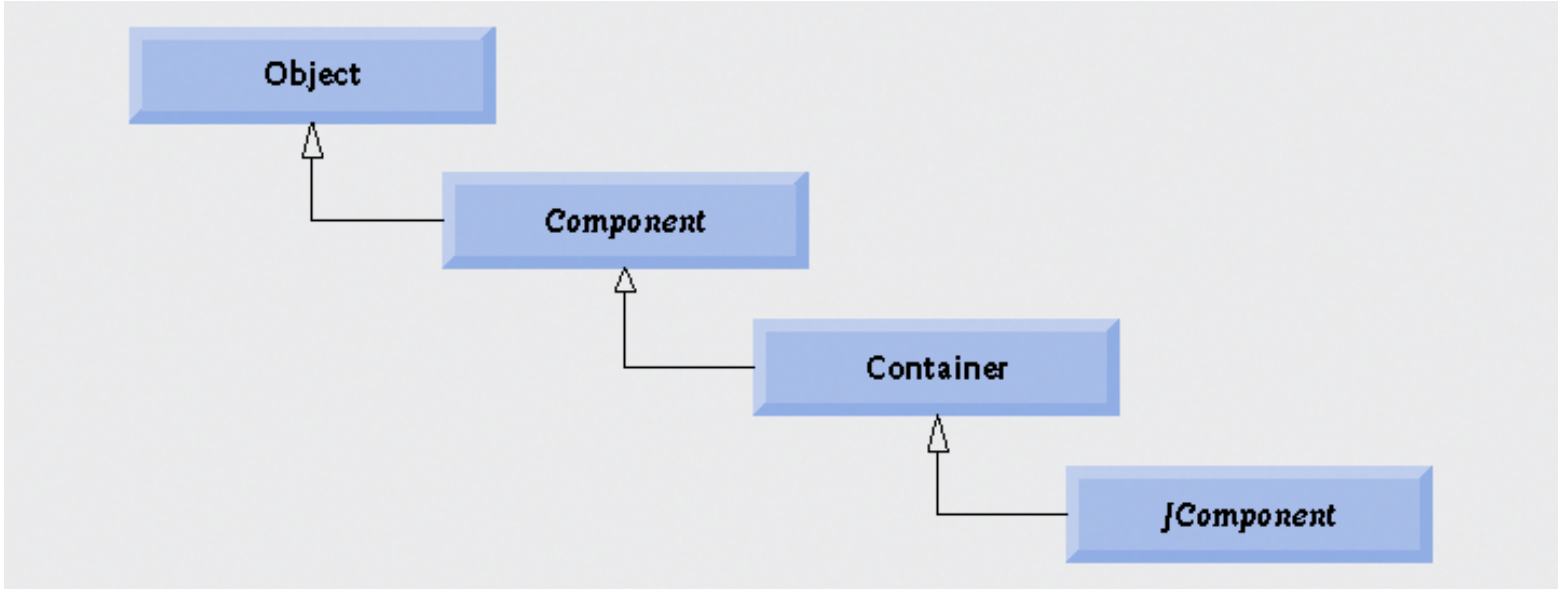


Figura 11.5 | Superclasses comuns de muitos dos componentes do Swing.

Superclasses de componentes GUI leves do Swing

- **Recursos dos componentes leves comuns:**
  - **Aparência e comportamento plugáveis para personalizar a aparência dos componentes.**
  - **Teclas de atalho (chamadas *mnemônicas*).**
  - **Capacidades comuns de tratamento de eventos.**
  - **Breves descrições do propósito de um componente GUI (chamadas *dicas de ferramenta*).**
  - **Suporte para *localização* de interface com o usuário.**

11.4 Exibição de texto e imagens em uma janela

- **Classe `Jframe`:**
  - **A maioria das janelas é uma instância ou subclasse dessa classe.**
  - **Fornece a barra de título.**
  - **Fornece botões para minimizar, maximizar e fechar a aplicação.**

Rotulando componentes GUI

- **Rótulo:**
  - **Instruções de texto ou informações que declaram o propósito de cada componente.**
  - **Criadas com a classe `Jlabel`.**

Observação sobre aparência e comportamento 11.5

Normalmente, o texto em um `JLabel` emprega maiúsculas e minúsculas no estilo de frases.

Especificando o layout

- **Organização dos contêineres:**
  - **Determina onde os componentes são colocados no contêiner.**
  - **Feita no Java com gerenciadores de layout.**
    - **Um dos quais é a classe `FlowLayout`.**
  - **Configure com o método `setLayout` da classe `Jframe`.**

Resumo

LabelFrame.java

(1 de 2)

```
1  // Fig. 11.6: LabelFrame.java
2  // Demonstrando a classe JLabel.
3  import java.awt.FlowLayout; // especifica como os componentes são organizados
4  import javax.swing.JFrame; // fornece recursos básicos de janela
5  import javax.swing.JLabel; // exibe texto e imagens
6  import javax.swing.SwingConstants; // constantes comuns utilizadas com Swing
7  import javax.swing.Icon; // interface utilizada para manipular imagens
8  import javax.swing.ImageIcon; // carrega imagens
9
10 public class LabelFrame extends JFrame
11 {
12    private JLabel label1; // JLabel apenas com texto
13    private JLabel label2; // JLabel construído com texto e ícone
14    private JLabel label3; // JLabel com texto e ícone adicionados
15
16    // Construtor LabelFrame adiciona JLabels a JFrame
17    public LabelFrame()
18    {
19       super( "Testing JLabel" );
20       setLayout( new FlowLayout() ); // configura o layout de frame
21
22       // Construtor JLabel com um argumento de string
23       label1 = new JLabel( "Label with text" );
24       label1.setToolTipText( "This is label1" );
25       add( label1 ); // adiciona label1 a JFrame
26
```

```
27       // construtor JLabel com string, Icon e argumentos de alinhamento
28       Icon bug = new ImageIcon( getClass().getResource(  "bug1.gif" ) );
29       label2 = new JLabel( "Label with text and icon", bug,
30          SwingConstants.LEFT );
31       label2.setToolTipText( "This is label2" );
32       add( label2 ); // adiciona label2 a JFrame
33
34       label3 = new JLabel(); // construtor JLabel sem argumentos
35       label3.setText( "Label with icon and text at bottom" );
36       label3.setIcon( bug ); // adiciona ícone a JLabel
37       label3.setHorizontalTextPosition( SwingConstants.CENTER );
38       label3.setVerticalTextPosition( SwingConstants.BOTTOM );
39       label3.setToolTipText( "This is label3" );
40       add( label3 ); // adiciona label3 a JFrame
41    } // fim do construtor LabelFrame
42 } // fim da classe LabelFrame
```

Resumo

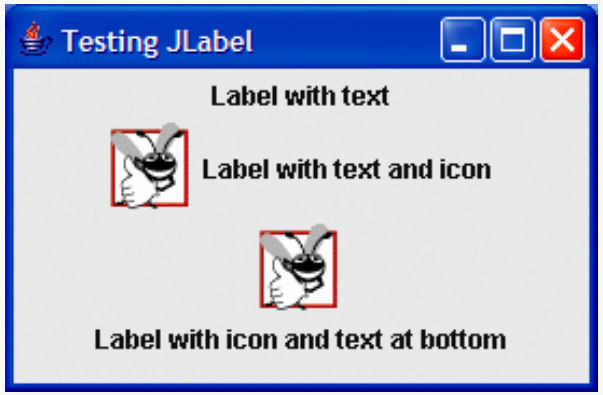
LabelFrame.java

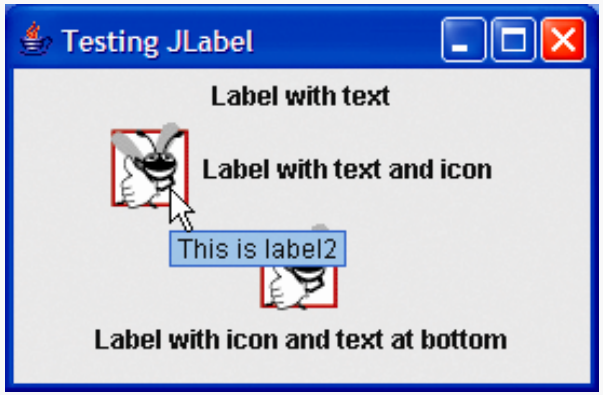
(2 de 2)

Resumo

LabelTest.java

```
1  // Fig. 11.7: LabelTest.java
2  // Testando LabelFrame.
3  import javax.swing.JFrame;
4
5  public class LabelTest
6  {
7     public static void main( String args[] )
8     {
9        LabelFrame labelFrame = new LabelFrame(); // cria LabelFrame
10       labelFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       labelFrame.setSize( 275, 180 ); // configura tamanho do frame
12       labelFrame.setVisible( true ); // exibe frame
13    } // fim de main
14 } // fim da classe LabelTest
```





Criando e anexando `label1`

- **Método `setToolTipText` da classe `Jcomponent`:**
  - **Especifica a dica de ferramenta.**
- **Método `add` da classe `Container`:**
  - **Adiciona um componente a um contêiner.**

Erro comum de programação 11.1

Se você não adicionar explicitamente um componente GUI a um contêiner, o componente GUI não será exibido quando o contêiner aparecer na tela.

Observação sobre a aparência e comportamento 11.6

Utilize as dicas de ferramenta para adicionar texto descritivo aos componentes GUI. Esse texto ajuda o usuário a determinar o propósito do componente GUI na interface com o usuário.

Criando e anexando `label2`

- **Interface `Icon`:**
  - **Pode ser adicionado a uma `JLabel` com o método `setIcon`.**
  - **Implementado pela classe `ImageIcon`.**
- **Interface `SwingConstants`:**
  - **Declara um conjunto de constantes inteiras comuns, como as utilizadas para configurar o alinhamento dos componentes.**
  - **Pode ser utilizada com os métodos `setHorizontalAlignment` e `setVerticalAlignment`.**

Criando e anexando `label3`

- **Outros métodos `Jlabel`:**
  - `getText` **e** `setText`
    - **Para configurar e recuperar o texto de um rótulo.**
  - `getIcon` **e** `setIcon`
    - **Para configurar e recuperar o ícone exibido no rótulo.**
  - `getHorizontalTextPosition` **e** `setHorizontalTextPosition`
    - **Para configurar e recuperar a posição horizontal do texto exibido no rótulo.**

Constante	Descrição
Constantes de posição horizontal	
`SwingConstants.LEFT`	Coloca o texto à esquerda.
`SwingConstants.CENTER`	Coloca o texto no centro.
`SwingConstants.RIGHT`	Coloca o texto à direita.
Constantes de posição vertical	
`SwingConstants.TOP`	Coloca o texto na parte superior.
`SwingConstants.CENTER`	Coloca o texto no centro.
`SwingConstants.BOTTOM`	Coloca o texto na parte inferior.

Figura 11.8 | Alguns componentes GUI básicos.

Criando e exibindo uma janela `LabelFrame`

- **Outros métodos `Jframe`:**
  - `setDefaultCloseOperation`
    - **Determina como a aplicação reage quando o usuário clica no botão de fechar.**
  - `setSize`
    - **Especifica a largura e altura da janela.**
  - `setVisible`
    - **Determina se a janela é exibida (`true`) ou não (`false`).**

11.5 Campos de texto e uma introdução ao tratamento de eventos com classes aninhadas

- **GUIs são baseadas em evento:**
  - **Uma interação com o usuário cria um evento.**
    - **Eventos comuns são clicar em um botão, digitar em um campo de texto, selecionar um item em um menu, fechar uma janela e mover o mouse.**
  - **O evento causa uma chamada a um método que chamou um handler de evento.**

11.5 Campos de texto e uma introdução ao tratamento de eventos com classes aninhadas. (*Continuação*)

- **Classe `JTextComponent`:**
  - **Superclasse de `JTextField`.**
    - **Superclasse de `JPasswordField`.**
      - **Adiciona o caractere de eco para ocultar a entrada de texto no componente.**
  - **Permite que o usuário insira texto no componente quando o componente tem o foco da aplicação.**

Resumo

TextFieldFrame .java

(1 de 3)

```
1  // Fig. 11.9: TextFieldFrame.java
2  // Demonstrando a classe JTextField.
3  import java.awt.FlowLayout;
4  import java.awt.event.ActionListener;
5  import java.awt.event.ActionEvent;
6  import javax.swing.JFrame;
7  import javax.swing.JTextField;
8  import javax.swing.JPasswordField;
9  import javax.swing.JOptionPane;
10
11 public class TextFieldFrame extends JFrame
12 {
13    private JTextField textField1; // campo de texto com tamanho configurado
14    private JTextField textField2; // campo de texto construído com texto
15    private JTextField textField3; // campo de texto com texto e tamanho
16    private JPasswordField passwordField; // campo de senha com texto
17
18    // Construtor TextFieldFrame adiciona JTextFields a JFrame
19    public TextFieldFrame()
20    {
21       super( "Testing JTextField and JPasswordField" );
22       setLayout( new FlowLayout() ); // configura layout de frame
23
24       // constrói textfield com 10 colunas
25       textField1 = new JTextField( 10 );
26       add( textField1 ); // adiciona textField1 a JFrame
27
```

Cria um novo `JTextField`

Resumo

TextFieldFrame.java

```
28       // constrói campo de texto com texto padrão
29       textField2 = new JTextField( "Enter text here" );
30       add( textField2 ); // adiciona textField2 a JFram
31
32       // constrói textfield com texto padrão e 21 colunas
33       textField3 = new JTextField( "Uneditable text field", 21 );
34       textField3.setEditable( false ); // desativa a edição
35       add( textField3 ); // adiciona textField3 ao JFra
36
37       // constrói passwordfield com o texto padrão
38       passwordField = new JPasswordField( "Hidden text" );
39       add( passwordField ); // adiciona passwordField a
40
41       // registra handlers de evento
42       TextFieldHandler handler = new TextFieldHandler();
43       textField1.addActionListener( handler );
44       textField2.addActionListener( handler );
45       textField3.addActionListener( handler );
46       passwordField.addActionListener( handler );
47    } // fim do construtor TextFieldFrame
48
49    // classe interna private para tratamento de evento
50    private class TextFieldHandler implements ActionListener
51    {
52       // processa eventos de campo de texto
53       public void actionPerformed( ActionEvent event )
54       {
55          String string = ""; // declara string a ser ex
56
```

Cria um novo `JTextField`

Cria um novo `JTextField` não editável

Cria um novo `JPasswordField`

Criar um handler de evento

Registra um handler de evento

Cria uma classe de handler de evento implementando a interface `ActionListener`

Declara o método `actionPerformed`

```
57          // usuário pressionou Enter no JTextField textField1
58          if ( event.getSource() == textField1 )
59             string = String.format( "textField1: %s",
60                event.getActionCommand() );
61
62          // usuário pressionou Enter no JTextField text
63          else if ( event.getSource() == textField2 )
64             string = String.format( "textField2: %s",
65                event.getActionCommand() );
66
67          // usuário pressionou Enter no JTextField text
68          else if ( event.getSource() == textField3 )
69             string = String.format( "textField3: %s",
70                event.getActionCommand() );
71
72          // usuário pressionou Enter no JTextField pass
73          else if ( event.getSource() == passwordField )
74             string = String.format( "passwordField: %s"
75                new String( passwordField.getPassword() ) );
76
77          // exibe conteúdo do JTextField
78          JOptionPane.showMessageDialog( null, string );
79       } // fim do método actionPerformed
80    } // fim da classe TextFieldHandler interna private
81 } // fim da classe TextFieldFrame
```

Testa se a origem do evento é o primeiro campo de texto

Obtém texto a partir do campo de texto

Testa se a origem do evento é o segundo campo de texto

(3 de 3)

Obtém texto a partir do campo de texto

Testa se a origem do evento é o terceiro campo de texto

Obtém texto a partir do campo de texto

Testa se a origem do evento é o campo de senha

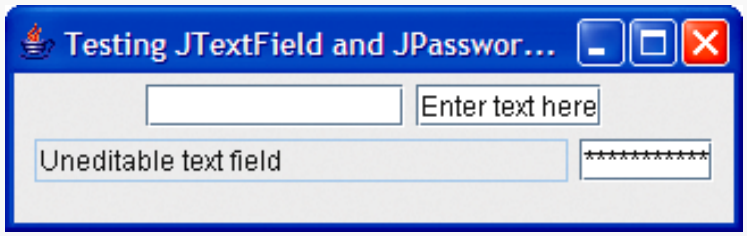
Obtém senha a partir do campo de senha

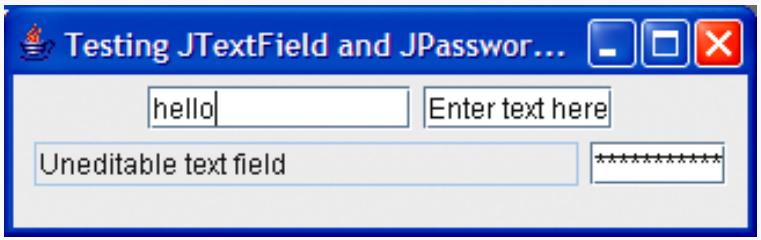
Resumo

TextFieldTest.java

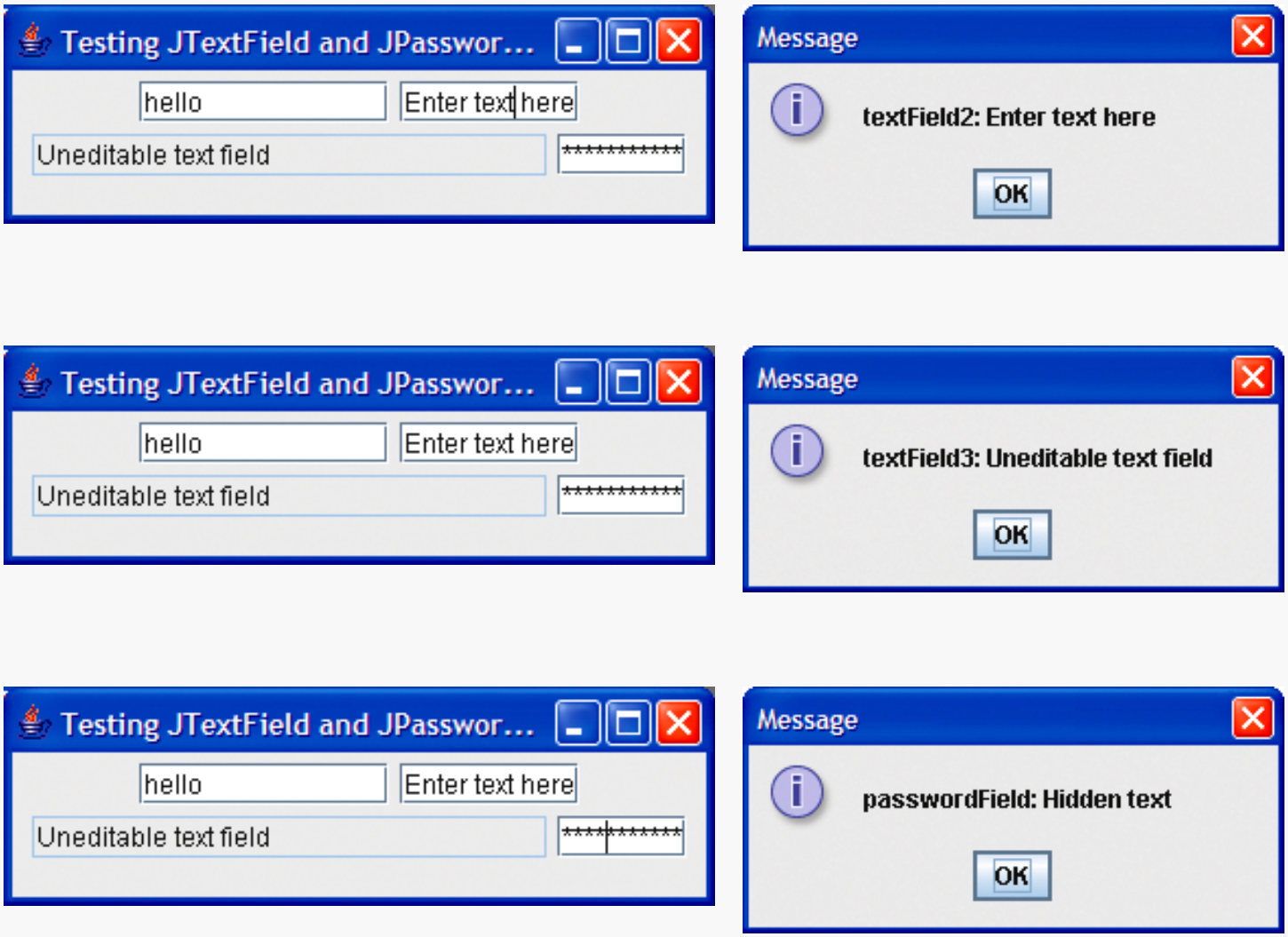
(1 de 2)

```
1  // Fig. 11.10: TextFieldTest.java
2  // Testando TextFieldFrame.
3  import javax.swing.JFrame;
4
5  public class TextFieldTest
6  {
7     public static void main( String args[] )
8     {
9        TextFieldFrame textFieldFrame = new TextFieldFrame();
10       textFieldFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       textFieldFrame.setSize( 325, 100 ); // configura tamanho do frame
12       textFieldFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe TextFieldTest
```





Resumo



`TextFieldTest.java`

(2 de 2)

Passos necessários para configurar o tratamento de evento de um componente GUI

- **Vários passos de codificação são requeridos para que uma aplicação responda a eventos:**
  - **Criar uma classe para o handler de evento.**
  - **Implementar uma interface ouvinte de evento apropriada.**
  - **Registrar o handler de evento.**

Utilizando uma classe aninhada para implementar um handler de evento

- **Classe de primeiro nível:**
  - **Não declarada dentro de uma outra classe.**
- **Classes aninhadas:**
  - **Declaradas dentro de uma outra classe.**
  - **Classes aninhadas não-`static` são chamadas classes internas.**
    - **Freqüentemente utilizadas para tratamento de eventos.**

Observação de engenharia de software 11.2

Uma classe interna tem permissão de acessar diretamente variáveis e métodos de sua classe de primeiro nível, mesmo se eles forem `private`.

Utilizando uma classe aninhada para implementar um handler de evento (*Cont.*)

- `JTextField` e `JPasswordField`:
  - **Pressionar Enter dentro de um desses campos causa um** `ActionEvent`.
    - **Processado pelos objetos que implementam a interface** `ActionListener`.

Registrando o handler de evento para cada campo de texto

- **Registrando um handler de evento.**
  - **Chama o método** `addActionListener` **para registrar um objeto** `ActionListener`.
  - `ActionListener` **ouve eventos no objeto.**

Observação de engenharia de software 11.3

O ouvinte de evento para um evento deve implementar a interface apropriada para o evento.

Erro comum de programação 11.2

Esquecer de registrar um objeto tratador de eventos para um tipo de evento de um componente GUI particular faz com que o tipo seja ignorado.

Detalhes do método `actionPerformed` da classe `TextFieldHandler`

- **Fonte do evento:**
  - **Componente a partir do qual o evento se origina.**
  - **Pode ser determinado utilizando o método `getSource`.**
  - **O texto em um `JTextField` pode ser adquirido utilizando `getActionCommand`.**
  - **O texto em um `JPasswordField` pode ser adquirido utilizando `getPassword`.**

11.6 Tipos comuns de eventos GUI e interfaces ouvintes

- **Tipos de eventos:**
  - **Todos são subclasses de `AWTEvent`.**
  - **Alguns declarados no pacote `java.awt.event`.**
  - **Aqueles específicos a componentes Swing declarados no `javax.swing.event`.**

11.6 Tipos comuns de eventos GUI e interfaces ouvintes (*Continuação*)

- **Modelo de evento de delegação:**
  - **A origem do evento é o componente com o qual o usuário interage.**
  - **O objeto do evento é criado e contém as informações sobre o evento que aconteceu.**
  - **O ouvinte de evento é notificado quando um evento acontece.**

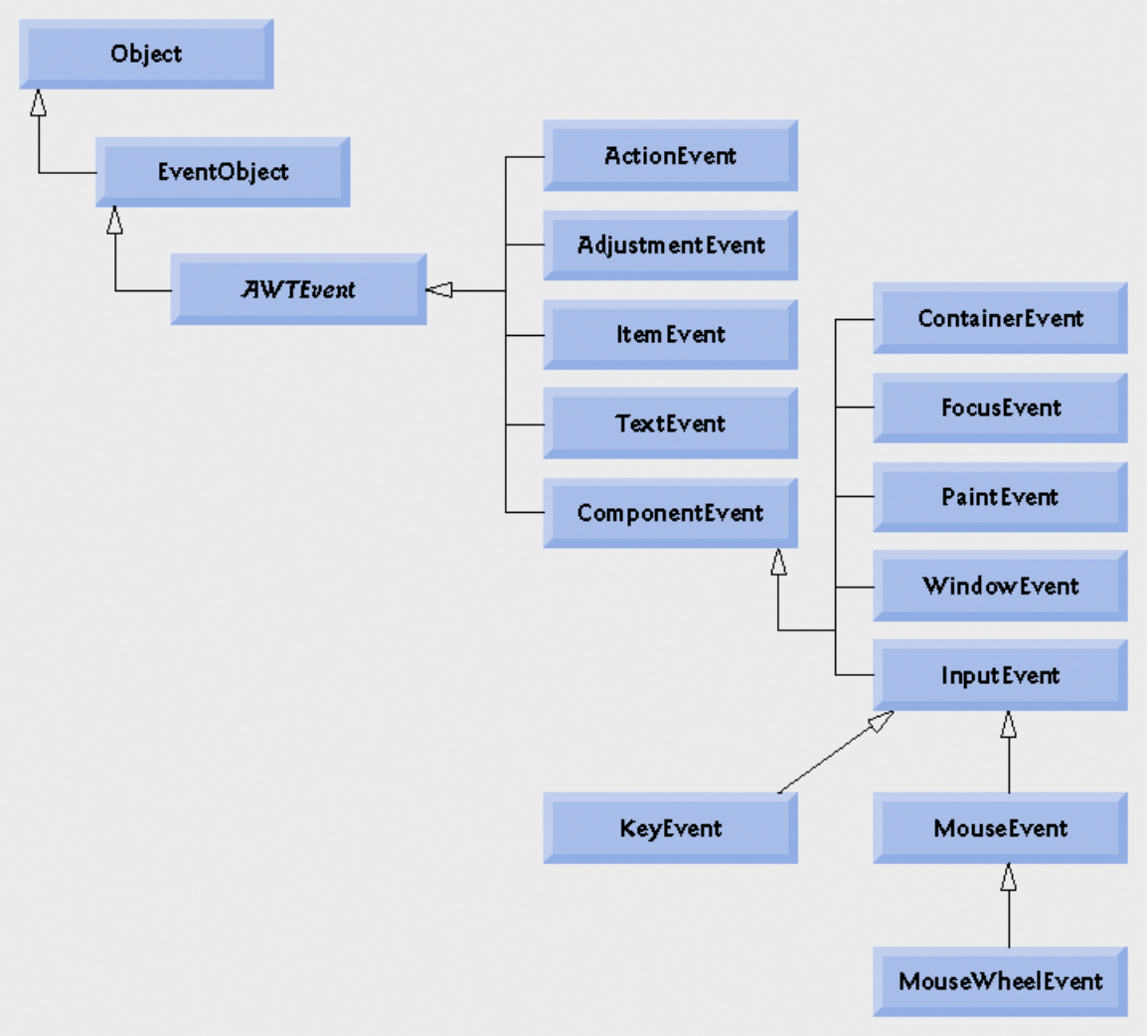


Figura 11.11 | Algumas classes de evento do pacote `java.awt.event`.

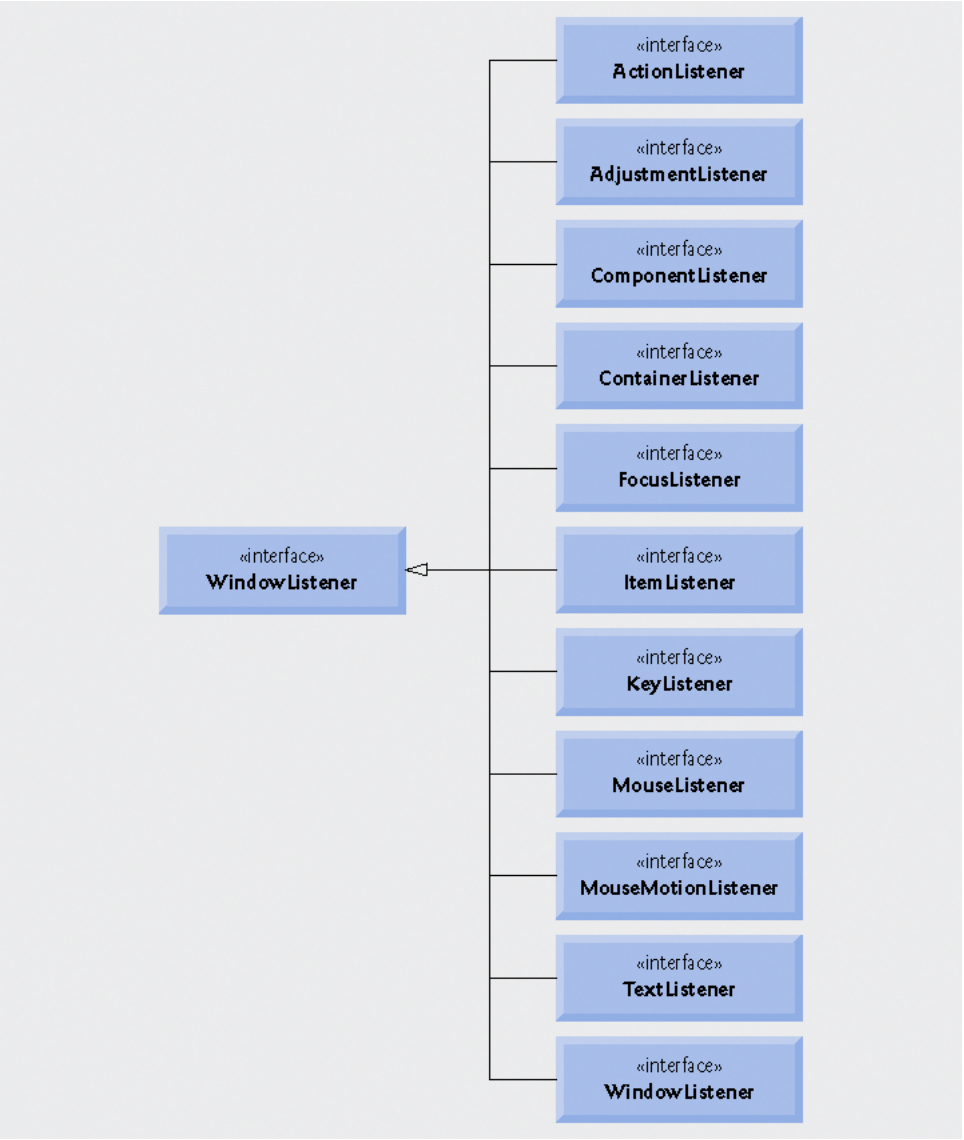


Figura 11.12 | Algumas interfaces ouvintes de eventos comuns do pacote `java.awt.event`.

11.7 Como o tratamento de evento funciona

- **Perguntas remanescentes:**
  - **Como o handler de evento ficou registrado?**
  - **Como o componente GUI sabe chamar `actionPerformed` em vez de algum outro método de tratamento de evento?**

Registrando eventos

- **Cada `JComponent` tem uma variável de instância `listenerList`:**
  - **Objeto do tipo `EventListenerList`.**
  - **Mantém referências a todos os seus ouvintes registrados.**

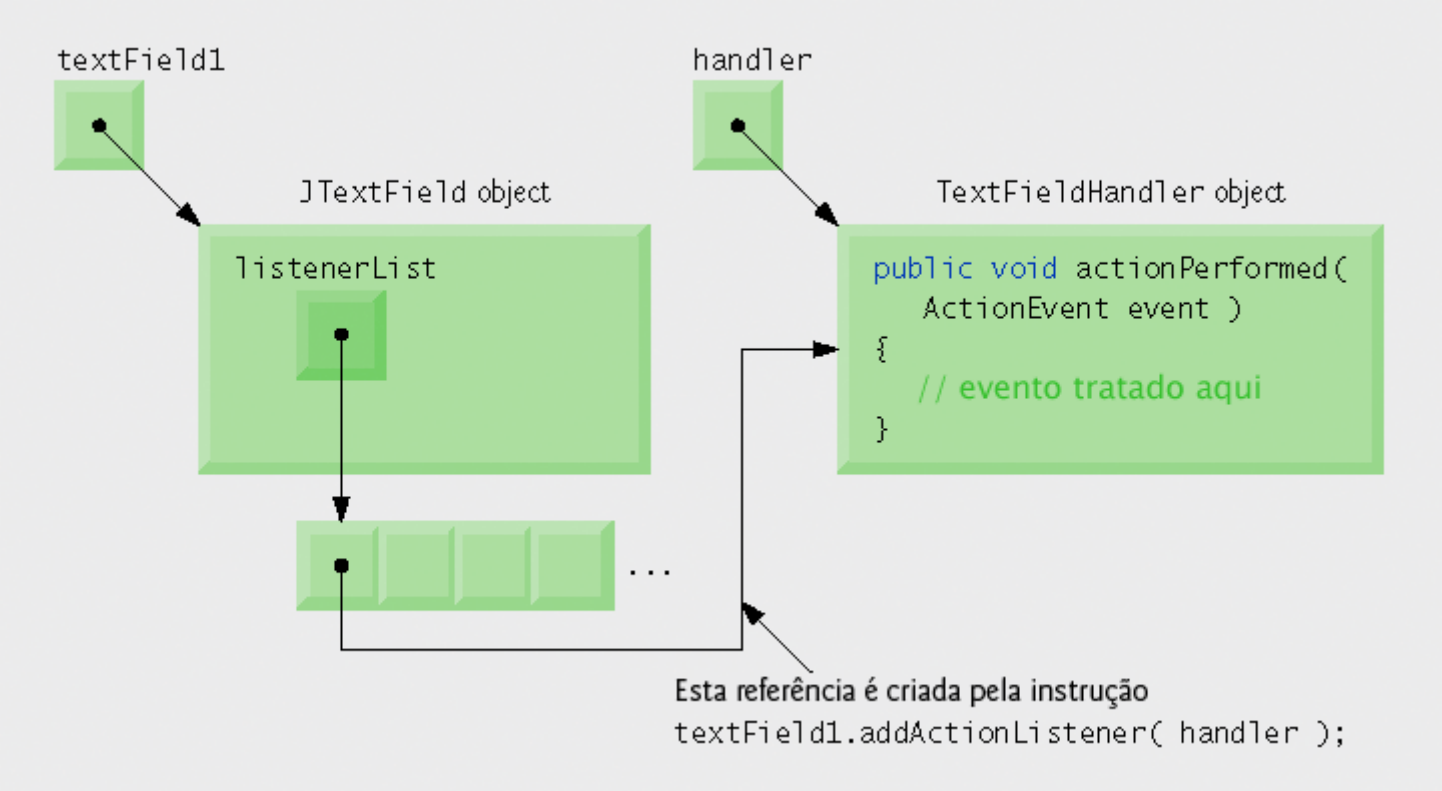


Figura 11.13 | Registro de evento para `JTextField textField1`.

Invocação de handler de evento

- **Eventos são despachados somente aos ouvintes dos eventos que correspondem ao tipo de evento.**
  - **Eventos têm um ID de evento único que especifica o tipo de evento.**
- **`MouseEvents` são tratados por `MouseListeners` e `MouseMotionsListeners`.**
- **`KeyEvents` são tratados por `KeyListeners`.**

11.8 JButton

- **Botão:**
  - **O usuário do componente clica para desencadear uma ação específica.**
  - **Pode ser botão de comando, caixa de seleção, botão de alternância ou botão de opção.**
  - **Os tipos de botões são subclasses da classe `AbstractButton`.**

Observação sobre aparência e comportamento 11.7

Em geral, os botões utilizam letras maiúsculas e minúsculas no estilo de título de livro.

11.8 JButton (*Continuação*)

- **Botão de comando:**
  - **Gera um `ActionEvent` quando é clicado.**
  - **Criado com a classe `Jbutton`.**
  - **O texto na face do botão é chamado rótulo do botão.**

Observação sobre aparência e comportamento 11.8

Ter mais de um `JButton` com o mesmo rótulo torna os `JButton`s ambíguos para o usuário. Forneça um rótulo único para cada botão.

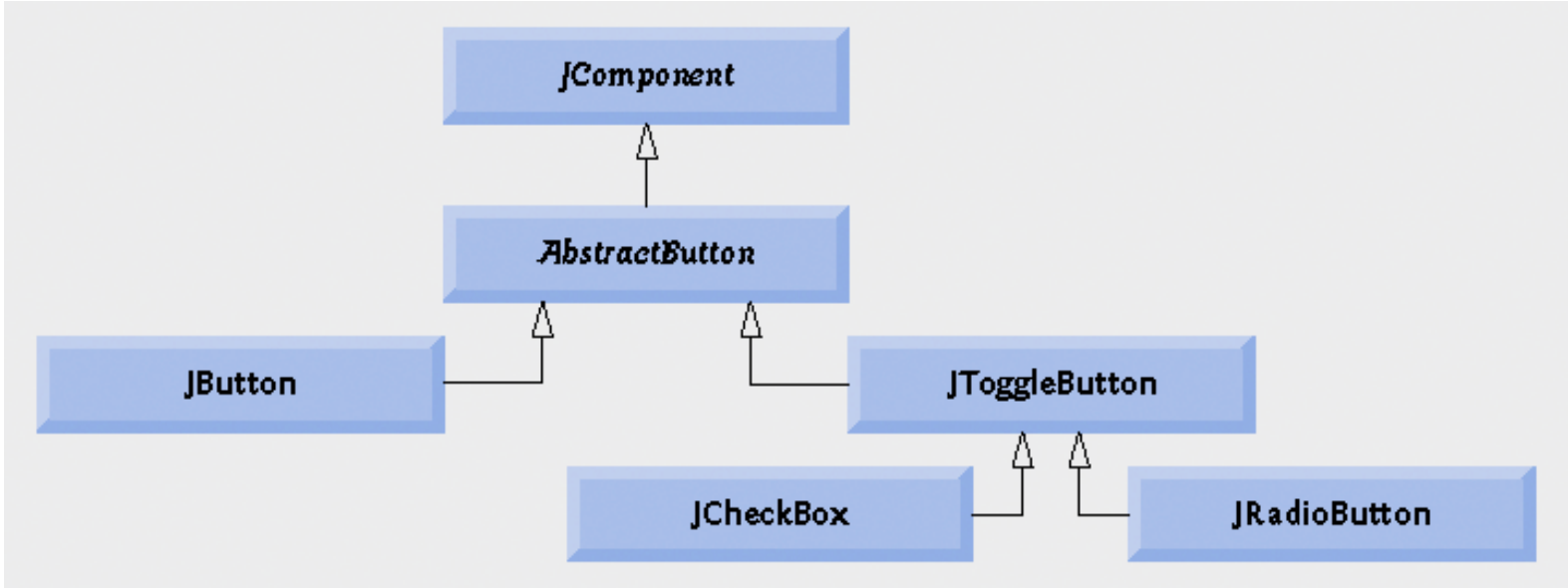


Figura 11.14 | Hierarquia do botão Swing.

Resumo

ButtonFrame.java

(1 de 2)

```
1  // Fig. 11.15: ButtonFrame.java
2  // Criando JButtons.
3  import java.awt.FlowLayout;
4  import java.awt.event.ActionListener;
5  import java.awt.event.ActionEvent;
6  import javax.swing.JFrame;
7  import javax.swing.JButton;
8  import javax.swing.Icon;
9  import javax.swing.ImageIcon;
10 import javax.swing.JOptionPane;
11
12 public class ButtonFrame extends JFrame
13 {
14    private JButton plainJButton; // botão apenas com texto
15    private JButton fancyJButton; // botão com ícones
16
17    // ButtonFrame adiciona JButtons ao JFrame
18    public ButtonFrame()
19    {
20       super( "Testing Buttons" );
21       setLayout( new FlowLayout() ); // configura o layout do frame
22
23       plainJButton = new JButton( "Plain Button" ); // botão com texto
24       add( plainJButton ); // adiciona plainJButton ao JFrame
25
26       Icon bug1 = new ImageIcon( getClass().getResource( "bug1.gif" ) );
27       Icon bug2 = new ImageIcon( getClass().getResource( "bug2.gif" ) );
28       fancyJButton = new JButton( "Fancy Button", bug1 );
29       fancyJButton.setRolloverIcon( bug2 ); // configura i
30       add( fancyJButton ); // adiciona fancyJButton ao JFrame
```

Declara duas variáveis de instância `JButton`

Cria um novo `JButton`

Cria dois `ImageIcon`s

Cria um novo `JButton`

Configura o ícone de rollover para `JButton`

Resumo

```
31
32       // cria novo ButtonHandler para tratamento de evento de botão
33       ButtonHandler handler = new ButtonHandler();
34       fancyJButton.addActionListener( handler );
35       plainJButton.addActionListener( handler );
36    } // fim do construtor ButtonFrame
37
38    // classe interna para tratamento de evento de botã
39    private class ButtonHandler implements ActionListener
40    {
41       // trata evento de botão
42       public void actionPerformed( ActionEvent event )
43       {
44          JOptionPane.showMessageDialog( ButtonFrame.this, String.format(
45             "You pressed: %s", event.getActionCommand() ) );
46       } // fim do método actionPerformed
47    } // fim da classe ButtonHandler interna private
48 } // fim da classe ButtonFrame
```

BbuttonFrame.java

(2 de 2)

Cria um handler para botões

Registra um handler de evento

A classe interna implementa `ActionListener`

Acessa a instância da classe externa utilizando essa referência

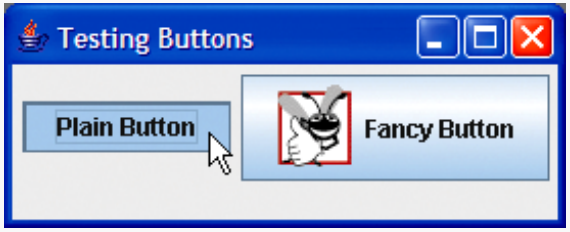
Obtém o texto do `JButton` pressionado

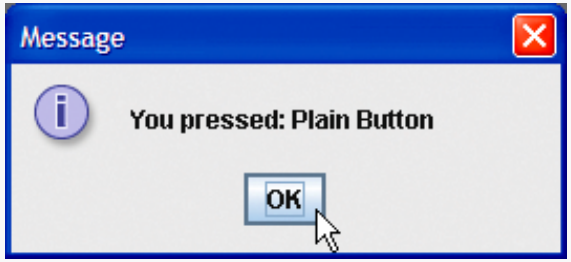
Resumo

ButtonTest.java

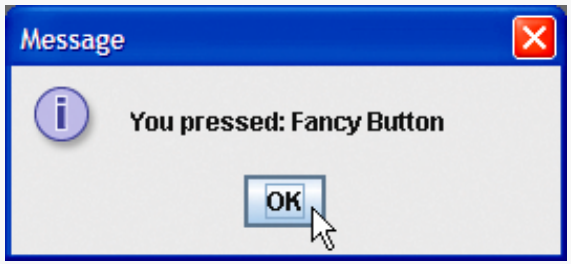
(1 de 2)

```
1  // Fig. 11.16: ButtonTest.java
2  // Testando ButtonFrame.
3  import javax.swing.JFrame;
4
5  public class ButtonTest
6  {
7     public static void main( String args[] )
8     {
9        ButtonFrame buttonFrame = new ButtonFrame(); // cria ButtonFrame
10       buttonFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       buttonFrame.setSize( 275, 110 ); // configura tamanho do frame
12       buttonFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe ButtonTest
```





Resumo



`ButtonTest.java`

(2 de 2)

11.8 JButton

- **JButtons podem ter um ícone de rollover.**
  - **Aparece quando o mouse é posicionado sobre um botão.**
  - **Adicionado a um JButton com o método setRolloverIcon.**

Observação sobre aparência e comportamento 11.9

Como a classe `AbstractButton` suporta exibição de texto e imagens em um botão, todas as subclasses de `AbstractButton` também suportam exibição de texto e imagens.

Observação sobre aparência e comportamento 11.10

Utilizar ícones rollover para `JButtons` fornece aos usuários um feedback visual que indica que, quando eles clicam no mouse enquanto o cursor está posicionado sobre o botão, uma ação ocorrerá.

Observação de engenharia de software 11.4

Quando utilizada em uma classe interna, a palavra-chave `this` referencia o objeto de classe interna atual sendo manipulado. Um método de classe interna pode utilizar `this` do seu objeto de classe externa precedendo `this` com o nome de classe externa e um ponto, como em `ButtonFrame.this`.

11.9 Botões que mantêm o estado

- **Botões de estado:**
  - **O Swing contém três tipos de botões de estado:**
  - `JToggleButton`, `JCheckBox` **e** `JRadioButton`.
  - `JCheckBox` **e** `JRadioButton` **são subclasses de** `JToggleButton`.

11.9.1 JCheckBox

- JCheckBox:
  - **Contém um rótulo de caixa de seleção que aparece à direita da caixa de seleção por padrão.**
  - **Gera um `ItemEvent` quando é clicado.**
    - `ItemEvent`s **são tratados por um** `ItemListener`.
    - **Passado para o método** `itemStateChanged`.
  - **O método** `isSelected` **retorna se uma caixa de seleção está selecionada (**`true`**) ou não (**`false`**).**

Resumo

CheckBoxFrame .java

(1 de 3)

```
1  // Fig. 11.17: CheckBoxFrame.java
2  // Criando botões JCheckBox.
3  import java.awt.FlowLayout;
4  import java.awt.Font;
5  import java.awt.event.ItemListener;
6  import java.awt.event.ItemEvent;
7  import javax.swing.JFrame;
8  import javax.swing.JTextField;
9  import javax.swing.JCheckBox;
10
11 public class CheckBoxFrame extends JFrame
12 {
13    private JTextField textField; // exibe o texto na alteração de fontes
14    private JCheckBox boldJCheckBox; // para aplicar/remover seleção de negrito
15    private JCheckBox italicJCheckBox; // para aplicar/remover seleção de itálico
16
17    // construtor CheckBoxFrame adiciona JCheckBoxes a JFrame
18    public CheckBoxFrame()
19    {
20       super( "JCheckBox Test" );
21       setLayout( new FlowLayout() ); // configura o layout do frame
22
23       // configura JTextField e sua fonte
24       textField = new JTextField( "Watch the font style change", 20 );
25       textField.setFont( new Font( "Serif", Font.PLAIN, 14 ) );
26       add( textField ); // adiciona textField a JFrame
27
```

Declara duas variáveis de instância JCheckBox

Configura a origem do campo de texto

Resumo

```
28       boldJCheckBox = new JCheckBox( "Bold" ); // cria caixa de seleção de negrito
29       italicJCheckBox = new JCheckBox( "Italic" ); // cria itálico
30       add( boldJCheckBox ); // adiciona caixa de seleção para negrito a JFrame
31       add( italicJCheckBox ); // adiciona caixa de sel
32
33       // registra listeners para JCheckBoxes
34       CheckBoxHandler handler = new CheckBoxHandler();
35       boldJCheckBox.addItemListener( handler );
36       italicJCheckBox.addItemListener( handler );
37    } // fim do construtor CheckBoxFrame
38
39    // classe interna private para tratamento de evento
40    private class CheckBoxHandler implements ItemListener
41    {
42       private int valBold = Font.PLAIN; // controla o
43       private int valItalic = Font.PLAIN; // controla
44
45       // responde aos eventos de caixa de seleção
46       public void itemStateChanged( ItemEvent event )
47       {
48          // processa aos eventos de caixa de seleção d
49          if ( event.getSource() == boldJCheckBox )
50             valBold =
51                boldJCheckBox.isSelected() ? Font.BOLD : Font.PLAIN;
52
```

CheckBoxFrame

(2 de 2)

Cria dois `JCheckBox`es

Cria um handler de evento

Registra um handler de evento com `JCheckBox`es

A classe interna implementa `ItemListener`

O método `itemStateChanged` é chamado quando uma `JCheckBox` é clicada

Testa se `JCheckBox` está selecionada

```
53          // processa eventos de caixa de seleção de itálico
54          if ( event.getSource() == italicJCheckBox )
55             valItalic =
56                italicJCheckBox.isSelected() ? Font.ITALIC : Font.PLAIN;
57
58          // configura fonte do campo de texto
59          textField.setFont(
60             new Font( "Serif", valBold + valItalic, 14 ) );
61       } // fim do método itemStateChanged
62    } // fim da classe CheckBoxHandler interna private
63 } // fim da classe CheckBoxFrame
```

Testa a origem do evento

O método `isSelected` retorna se `JCheckBox` está selecionada

Resumo

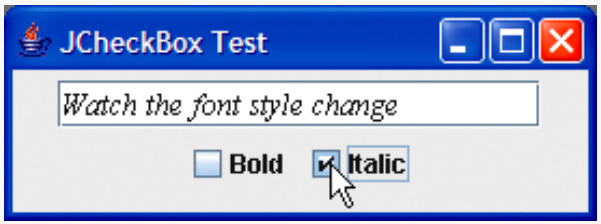
`oxFrame`

(3 de 3)

Resumo

CheckBoxTest.java

```
1  // Fig. 11.18: CheckBoxTest.java
2  // Testando CheckBoxFrame.
3  import javax.swing.JFrame;
4
5  public class CheckBoxTest
6  {
7     public static void main( String args[] )
8     {
9        CheckBoxFrame checkBoxFrame = new CheckBoxFrame();
10       checkBoxFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       checkBoxFrame.setSize( 275, 100 ); // configura o tamanho do frame
12       checkBoxFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe CheckBoxTest
```



11.9.2 JRadioButton

- JRadioButton:
  - **Tem dois estados – *selecionado e não selecionado*.**
  - **Normalmente aparece em um grupo no qual somente um botão de opção pode ser selecionado de cada vez.**
    - **Grupo mantido por um objeto** ButtonGroup.
      - **Declara o método** add **para adicionar um** JRadioButton **ao grupo.**
  - **Normalmente, representa opções mutuamente exclusivas.**

Erro comum de programação 11.3

Adicionar um objeto `ButtonGroup` (ou um objeto de nenhuma outra classe que não deriva de `Component`) a um contêiner resulta em um erro de compilação.

Resumo

RadioButtonFrame.java

(1 de 3)

```
1  // Fig. 11.19: RadioButtonFrame.java
2  // Criando botões de opção utilizando ButtonGroup e JRadioButton.
3  import java.awt.FlowLayout;
4  import java.awt.Font;
5  import java.awt.event.ItemListener;
6  import java.awt.event.ItemEvent;
7  import javax.swing.JFrame;
8  import javax.swing.JTextField;
9  import javax.swing.JRadioButton;
10 import javax.swing.ButtonGroup;
11
12 public class RadioButtonFrame extends JFrame
13 {
14    private JTextField textField; // usado para exibir
15    private Font plainFont; // fonte para texto simples
16    private Font boldFont; // fonte para texto negrito
17    private Font italicFont; // fonte para texto itálico
18    private Font boldItalicFont; // fonte para texto negrito e itálico
19    private JRadioButton plainJRadioButton; // seleciona texto simples
20    private JRadioButton boldJRadioButton; // seleciona texto negrito
21    private JRadioButton italicJRadioButton; // seleciona texto itálico
22    private JRadioButton boldItalicJRadioButton; // negrito e itálico
23    private ButtonGroup radioGroup; // buttongroup para armazenar botões de opção
24
25    // construtor RadioButtonFrame adiciona JRadioButtons ao JFrame
26    public RadioButtonFrame()
27    {
28       super( "RadioButton Test" );
29       setLayout( new FlowLayout() ); // configura layout do frame
30
```

Declara quatro `JRadioButtons` e um `ButtonGroup` para gerenciá-los

Resumo

RadioButtonFrame.java

(2 de 3)

```
31       textField = new JTextField( "Watch the font style change", 25 );
32       add( textField ); // adiciona textField ao JFrame
33
34       // cria botões de opção
35       plainJRadioButton = new JRadioButton( "Plain", true );
36       boldJRadioButton = new JRadioButton( "Bold", false );
37       italicJRadioButton = new JRadioButton( "Italic", false );
38       boldItalicJRadioButton = new JRadioButton( "Bold/Italic", false );
39       add( plainJRadioButton ); // adiciona botão simples ao JFrame
40       add( boldJRadioButton ); // adiciona botão de negrito ao JFrame
41       add( italicJRadioButton ); // adiciona botão de
42       add( boldItalicJRadioButton ); // adiciona botão de itálico e negrito
43
44       // cria relacionamento lógico entre JRadioButtons
45       radioGroup = new ButtonGroup(); // cria ButtonGroup
46       radioGroup.add( plainJRadioButton ); // adiciona
47       radioGroup.add( boldJRadioButton ); // adiciona
48       radioGroup.add( italicJRadioButton ); // adiciona itálico ao grupo
49       radioGroup.add( boldItalicJRadioButton ); // adiciona negrito e itálico
50
51       // cria objetos de fonte
52       plainFont = new Font( "Serif", Font.PLAIN, 14 );
53       boldFont = new Font( "Serif", Font.BOLD, 14 );
54       italicFont = new Font( "Serif", Font.ITALIC, 14 );
55       boldItalicFont = new Font( "Serif", Font.BOLD + Font.ITALIC, 14 );
56       textField.setFont( plainFont ); // configura fonte inicial à simples
57
```

Cria os quatro `JRadioButtons`

Cria o `ButtonGroup`

Adiciona cada `JRadioButton` ao `ButtonGroup`

Resumo

RadioButtonFrame.java

(3 de 3)

```
58       // registra eventos para JRadioButtons
59       plainJRadioButton.addItemListener(
60          new RadioButtonHandler( plainFont ) );
61       boldJRadioButton.addItemListener(
62          new RadioButtonHandler( boldFont ) );
63       italicJRadioButton.addItemListener(
64          new RadioButtonHandler( italicFont ) );
65       boldItalicJRadioButton.addItemListener(
66          new RadioButtonHandler( boldItalicFont ) );
67    } // fim do construtor RadioButtonFrame
68
69    // classe interna private para tratar eventos de botão de opção
70    private class RadioButtonHandler implements ItemListener
71    {
72       private Font font; // fonte associada com esse l
73
74       public RadioButtonHandler( Font f )
75       {
76          font = f; // configura a fonte desse listene
77       } // fim do construtor RadioButtonHandler
78
79       // trata eventos de botão de opção
80       public void itemStateChanged( ItemEvent event )
81       {
82          textField.setFont( font ); // configura fonte de textField
83       } // fim do método itemStateChanged
84    } // fim da classe RadioButtonHandler interna private
85 } // fim da classe RadioButtonFrame
```

Registra um handler de evento com cada `JRadioButton`

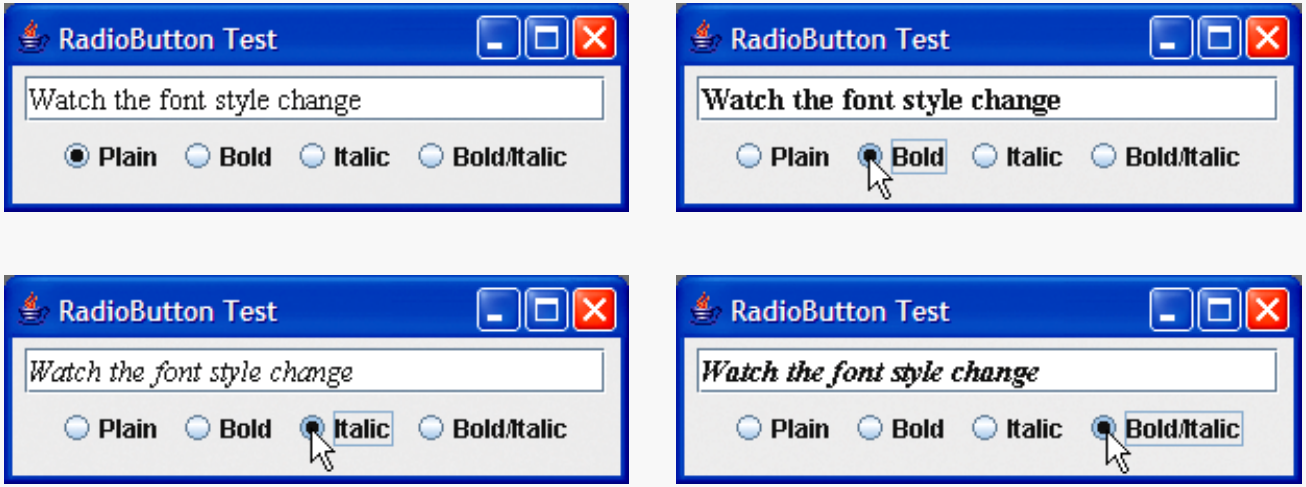
A classe interna do handler de evento implementa `ItemListener`

Quando o botão de opção é selecionado, a origem do campo de texto é configurada com o valor passado para o construtor

Resumo

RadioButtonTest.java

```
1  // Fig. 11.20: RadioButtonTest.java
2  // Testando RadioButtonFrame.
3  import javax.swing.JFrame;
4
5  public class RadioButtonTest
6  {
7     public static void main( String args[] )
8     {
9        RadioButtonFrame radioButtonFrame = new RadioButtonFrame();
10       radioButtonFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       radioButtonFrame.setSize( 300, 100 ); // configura tamanho do frame
12       radioButtonFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe RadioButtonTest
```



11.10 JComboBox e utilização de uma classe interna anônima para tratamento de eventos

- **Caixa de combinação:**
  - **Às vezes, também chamada *lista drop-down*.**
  - **Implementada pela classe `JComboBox`.**
  - **Cada item na lista tem um índice.**
  - **`setMaximumRowCount` configura o número máximo de linhas mostradas de cada vez.**
  - **`JComboBox` fornece uma barra de rolagem e setas para cima e para baixo para percorrer a lista.**

Observação sobre aparência e comportamento 11.11

Configure a contagem máxima de linha para uma `JComboBox` com um número de linhas que impede a lista de expandir-se para fora dos limites da janela em que ela é utilizada.

Essa configuração irá assegurar que a lista seja exibida corretamente quando for expandida pelo usuário.

Utilizando uma classe interna anônima para tratamento de evento

- **Classe interna anônima:**
  - **Forma especial de classe interna.**
  - **Declarada sem nome.**
  - **Em geral, aparece dentro de uma chamada de método.**
  - **Tem acesso limitado a variáveis locais.**

Resumo

ComboBoxFrame .java

(1 de 2)

```
1  // Fig. 11.21: ComboBoxFrame.java
2  // Usando a JComboBox para selecionar uma image para exibição.
3  import java.awt.FlowLayout;
4  import java.awt.event.ItemListener;
5  import java.awt.event.ItemEvent;
6  import javax.swing.JFrame;
7  import javax.swing.JLabel;
8  import javax.swing.JComboBox;
9  import javax.swing.Icon;
10 import javax.swing.ImageIcon;
11
12 public class ComboBoxFrame extends JFrame
13 {
14    private JComboBox imagesJComboBox; // caixa de combinação p/ armazenar nomes de ícones
15    private JLabel label; // rótulo para exibir ícone selecionado
16
17    private String names[] =
18       { "bug1.gif", "bug2.gif",  "travelbug.gif", "buganim.gif" };
19    private Icon icons[] = {
20       new ImageIcon( getClass().getResource( names[ 0 ] ) ),
21       new ImageIcon( getClass().getResource( names[ 1 ] ) ),
22       new ImageIcon( getClass().getResource( names[ 2 ] ) ),
23       new ImageIcon( getClass().getResource( names[ 3 ] ) ) };
24
25    // construtor ComboBoxFrame adiciona JComboBox ao JFrame
26    public ComboBoxFrame()
27    {
28       super( "Testing JComboBox" );
29       setLayout( new FlowLayout() ); // configura layout do frame
30
```

Declara a variável de instância `JComboBox`

```
31       imagesJComboBox = new JComboBox( names ); // configure a JComboBox
32       imagesJComboBox.setMaximumRowCount( 3 ); // exib
33
34       imagesJComboBox.addItemListener(
35          new ItemListener() // classe interna anônima
36          {
37             // trata evento JComboBox
38             public void itemStateChanged( ItemEvent event )
39             {
40                // determina se caixa de seleção está marcada ou não
41                if ( event.getStateChange() == ItemEvent.SELECTED )
42                   label.setIcon( icons[
43                      imagesJComboBox.getSelectedIndex
44             } // fim do método itemStateChanged
45          } // fim da classe interna anônima
46       ); // fim da chamada para addItemListener
47
48       add( imagesJComboBox ); // adiciona a caixa de combinação ao JFrame
49       label = new JLabel( icons[ 0 ] ); // exibe o primeiro ícone
50       add( label ); // adiciona rótulo ao JFrame
51    } // fim do construtor ComboBoxFrame
52 } // fim da classe ComboBoxFrame
```

Cria `JComboBox` e configura a contagem máxima de linhas

Cria a classe interna anônima como o handler de evento

Declara o método `itemStateChanged`

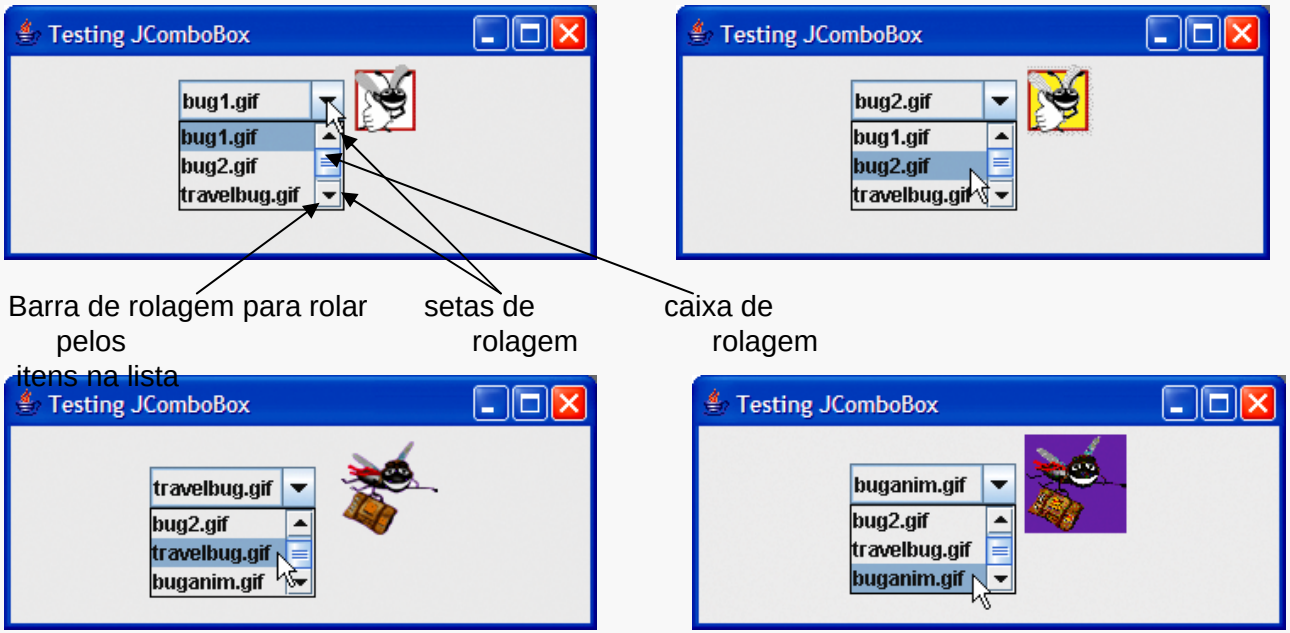
Testa a alteração de estado da `JComboBox`

O método `getSelectedIndex` localiza o item selecionado

Resumo

ComboBoxTest.java

```
1  // Fig. 11.22: ComboBoxTest.java
2  // Testando ComboBoxFrame.
3  import javax.swing.JFrame;
4
5  public class ComboBoxTest
6  {
7     public static void main( String args[] )
8     {
9        ComboBoxFrame comboBoxFrame = new ComboBoxFrame();
10       comboBoxFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       comboBoxFrame.setSize( 350, 150 ); // configura o tamanho do frame
12       comboBoxFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe ComboBoxTest
```



Observação de engenharia de software 11.5

Uma classe interna anônima declarada em um método pode acessar as variáveis de instância e métodos do objeto de classe de primeiro nível que a declararam, bem como as variáveis locais `final` do método, mas não pode acessar variáveis não-`final` do método.

Observação de engenharia de software 11.6

Como qualquer outra classe, quando uma classe interna anônima implementa uma interface, a classe deve implementar cada método na interface.

11.11 `JList`

- **Lista:**
  - **Exibe uma série de itens dentre os quais usuário pode selecionar um ou mais.**
  - **Implementada pela classe `Jlist`.**
  - **Permite listas de seleção única ou listas de múltipla seleção.**
  - **Um `ListSelectionEvent` ocorre quando um item é selecionado.**
    - **Tratado por um `ListSelectionListener` e passado para o método `valueChanged`.**

Resumo

ListFrame.java

(1 de 2)

```
1  // Fig. 11.23: ListFrame.java
2  // Selecionando cores a partir de uma JList.
3  import java.awt.FlowLayout;
4  import java.awt.Color;
5  import javax.swing.JFrame;
6  import javax.swing.JList;
7  import javax.swing.JScrollPane;
8  import javax.swing.event.ListSelectionListener;
9  import javax.swing.event.ListSelectionEvent;
10 import javax.swing.ListSelectionModel;
11
12 public class ListFrame extends JFrame
13 {
14    private JList colorJList; // lista para exibir cores
15    private final String colorNames[] = { "Black", "Blue", "Cyan",
16       "Dark Gray", "Gray", "Green", "Light Gray", "Magenta",
17       "Orange", "Pink", "Red", "White", "Yellow" };
18    private final Color colors[] = { Color.BLACK, Color.BLUE, Color.CYAN,
19       Color.DARK_GRAY, Color.GRAY, Color.GREEN, Color.LIGHT_GRAY,
20       Color.MAGENTA, Color.ORANGE, Color.PINK, Color.RED, Color.WHITE,
21       Color.YELLOW };
22
23    // construtor ListFrame adiciona JScrollPane que contém JList ao JFrame
24    public ListFrame()
25    {
26       super( "List Test" );
27       setLayout( new FlowLayout() ); // configura o layout de frame
28
```

Declara a variável de instância `JList`

Resumo

```
29       colorJList = new JList( colorNames ); // cria com colorNames
30       colorJList.setVisibleRowCount( 5 ); // exibe cinco
                                                          vez
31
32       // não permite múltiplas seleções
33       colorJList.setSelectionMode( ListSelectionModel.SINGLE_SELECTION );
34
35       // adiciona um JScrollPane que contém JList ao f
                                                          va
36       add( new JScrollPane( colorJList ) );
37
38       colorJList.addListSelectionListener(
39          new ListSelectionListener() // classe interna
40          {
41             // trata eventos de seleção de lista
42             public void valueChanged( ListSelectionEvent event )
43             {
44                getContentPane().setBackground(
45                   colors[ colorJList.getSelectedIndex() ] );
46             } // fim do método valueChanged
47          } // fim da classe interna anônima
48       ); // fim da chamada para addListSelectionListener
49    } // fim do construtor ListFrame
50 } // fim da classe ListFrame
```

Cria `JList`

Configura o modo de seleção da `JList`

(2 de 2)

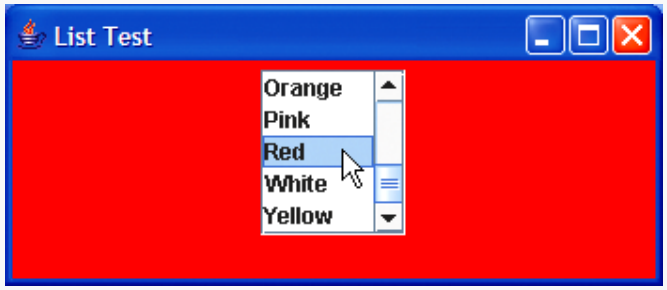
Adiciona `JList` a `ScrollPane` e a adiciona à aplicação

Obtém o índice do item selecionado

Resumo

ListTest.java

```
1  // Fig. 11.24: ListTest.java
2  // Selecionando cores a partir de uma JList.
3  import javax.swing.JFrame;
4
5  public class ListTest
6  {
7     public static void main( String args[] )
8     {
9        ListFrame listFrame = new ListFrame(); // cria ListFrame
10       listFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       listFrame.setSize( 350, 150 ); // configura tamanho do frame
12       listFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe ListTest
```



11.12 Listas de seleção múltipla

- **Lista de seleção múltipla:**
  - **Permite que usuários selecionem vários itens.**
  - **Seleção de um único intervalo que permite apenas um intervalo contínuo de itens.**
  - **Seleção de múltiplos intervalos que permite que qualquer conjunto de elementos seja selecionado.**

Resumo

Multiple SelectionFrame .java

(1 de 3)

```
1  // Fig. 11.25: MultipleSelectionFrame.java
2  // Copiando itens de uma List para a outra.
3  import java.awt.FlowLayout;
4  import java.awt.event.ActionListener;
5  import java.awt.event.ActionEvent;
6  import javax.swing.JFrame;
7  import javax.swing.JList;
8  import javax.swing.JButton;
9  import javax.swing.JScrollPane;
10 import javax.swing.ListSelectionModel;
11
12 public class MultipleSelectionFrame extends JFrame
13 {
14    private JList colorJList; // lista para armazenar nomes de cores
15    private JList copyJList; // lista para copiar nomes de cores no
16    private JButton copyJButton; // botão para copiar nomes selecionados
17    private final String colorNames[] = { "Black", "Blue", "Cyan",
18       "Dark Gray", "Gray", "Green", "Light Gray", "Magenta", "Orange",
19       "Pink", "Red", "White", "Yellow" };
20
21    // construtor MultipleSelectionFrame
22    public MultipleSelectionFrame()
23    {
24       super( "Multiple Selection Lists" );
25       setLayout( new FlowLayout() ); // configura layout do frame
26
```

Resumo

```
27       colorJList = new JList( colorNames ); // armazena nomes de todas as cores
28       colorJList.setVisibleRowCount( 5 ); // mostra cinco linhas
29       colorJList.setSelectionMode(
30          ListSelectionModel.MULTIPLE_INTERVAL_SELECTION );
31       add( new JScrollPane( colorJList ) ); // adicion
32
33       copyJButton = new JButton( "Copy >>>" ); // cria botão de cópia
34       copyJButton.addActionListener(
35
36          new ActionListener() // classe interna anônima
37          {
38             // trata evento de botão
39             public void actionPerformed( ActionEvent event )
40             {
41                // coloca valores selecionados na copyJList
42                copyJList.setListData( colorJList.getSelectedValues() );
43             } // fim do método actionPerformed
44          } // fim da classe interna anônima
45       ); // fim da chamada para addActionListener
46
```

Utiliza uma lista de seleção de múltiplos intervalos

Utiliza os métodos `setListData` e `getSelectedValues` para copiar valores de uma `JList` para outra

e SelectionFrame .java

(2 de 3)

```
47       add( copyJButton ); // adiciona botão de cópia ao JFrame
48
49       copyJList = new JList(); // cria lista p/ armaze
50       copyJList.setVisibleRowCount( 5 ); // mostra 5
51       copyJList.setFixedCellWidth( 100 ); // configura largura
52       copyJList.setFixedCellHeight( 15 ); // configura altura
53       copyJList.setSelectionMode(
54          ListSelectionModel.SINGLE_INTERVAL_SELECTION
55       add( new JScrollPane( copyJList ) ); // adiciona lista com scrollpane
56    } // fim do construtor MultipleSelectionFrame
57 } // fim da classe MultipleSelectionFrame
```

Configura a largura da célula para apresentação

Configura a altura da célula para apresentação

Configura o modelo de seleção como seleção de um único intervalo

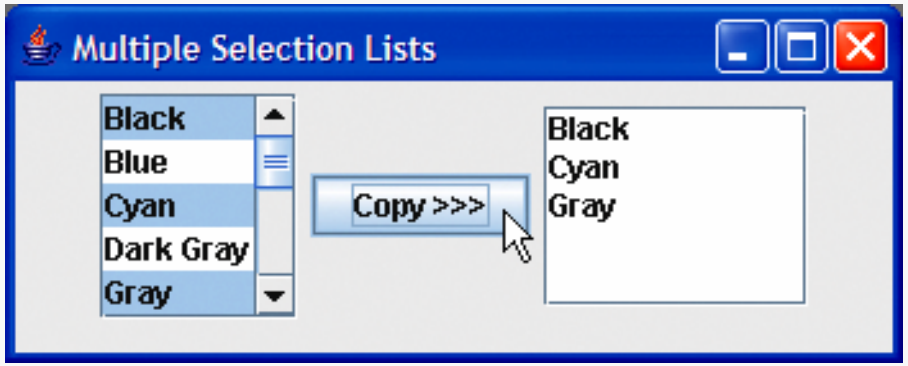
Resumo

Multiple
onFrame

Resumo

MultipleSelectionTest.java

```
1  // Fig. 11.26: MultipleSelectionTest.java
2  // Testando MultipleSelectionFrame.
3  import javax.swing.JFrame;
4
5  public class MultipleSelectionTest
6  {
7     public static void main( String args[] )
8     {
9        MultipleSelectionFrame multipleSelectionFrame =
10          new MultipleSelectionFrame();
11       multipleSelectionFrame.setDefaultCloseOperation(
12          JFrame.EXIT_ON_CLOSE );
13       multipleSelectionFrame.setSize( 350, 140 ); // configura o tamanho do frame
14       multipleSelectionFrame.setVisible( true ); // exibe o frame
15    } // fim de main
16 } // fim da classe MultipleSelectionTest
```



11.13 Tratamento de evento de mouse

- **Eventos de mouse:**
  - **Cria um objeto** `MouseEvent`.
  - **Tratado por** `MouseListeners` **e** `MouseMotionListeners`**.**
  - `MouseInputListener` **combina as duas interfaces.**
  - **A interface** `MouseWheelListener` **declara o método** `mouseWheelMoved` **para tratar** `MouseWheelEvents`.

Métodos de interface `MouseListener` e `MouseMotionListener`

Métodos de interface ***`MouseListener`***

`public void mousePressed( MouseEvent event )`

Chamado quando um botão do mouse é pressionado enquanto o cursor de mouse estiver sobre um componente.

`public void mouseClicked( MouseEvent event )`

Chamado quando um botão do mouse é pressionado e liberado enquanto o cursor do mouse pairar sobre um componente. Esse evento é sempre precedido por uma chamada para `mousePressed`.

`public void mouseReleased( MouseEvent event )`

Chamado quando um botão do mouse é liberado depois de ser pressionado. Esse evento sempre é precedido por uma chamada para mousePressed e um ou mais chamadas para `mouseDragged`.

`public void mouseEntered( MouseEvent event )`

Chamado quando o cursor do mouse entra nos limites de um componente.

Figura 11.27 | Métodos de interface `MouseListener` e `MouseMotionListener`. (Parte 1 de 2.)

Métodos de interface `MouseListener` e `MouseMotionListener`	
`public void mouseExited( MouseEvent event )`	
	Chamado quando o cursor do mouse deixa os limites de um componente.
Métodos de interface `MouseMotionListener`	
`public void mouseDragged( MouseEvent event )`	
	Chamado quando o botão do mouse é pressionado enquanto o cursor de mouse estiver sobre um componente e o mouse é movido enquanto o botão do mouse permanecer pressionado. Esse evento é sempre precedido por uma chamada para `mousePressed`. Todos os eventos de arrastar são enviados para o componente em que o usuário começou a arrastar o mouse.
`public void mouseMoved( MouseEvent event )`	
	Chamado quando o mouse é movido quando o cursor de mouse estiver sobre um componente. Todos os eventos de movimento são enviados para o componente sobre o qual o mouse atualmente está posicionado.

Figura 11.27 | Métodos de interface `MouseListener` e `MouseMotionListener`. (Parte 2 de 2.)

Observação sobre a aparência e comportamento 11.12

As chamadas de método para `mouseDragged` e `mouseReleased` são enviadas ao `MouseMotionListener` do `Component` em que uma operação de arrastar do mouse se iniciou. De maneira semelhante, a chamada de método `mouseReleased` no fim de uma operação de arrastar é enviada para o `MouseListener` de `Component` em que a operação de arrastar se iniciou.

Resumo

MouseTrackerFrame.java

(1 de 4)

```
1  // Fig. 11.28: MouseTrackerFrame.java
2  // Demonstrando eventos de mouse.
3  import java.awt.Color;
4  import java.awt.BorderLayout;
5  import java.awt.event.MouseListener;
6  import java.awt.event.MouseMotionListener;
7  import java.awt.event.MouseEvent;
8  import javax.swing.JFrame;
9  import javax.swing.JLabel;
10 import javax.swing.JPanel;
11
12 public class MouseTrackerFrame extends JFrame
13 {
14    private JPanel mousePanel; // painel em que eventos de mouse ocorrerão
15    private JLabel statusBar; // rótulo que exibe informações sobre evento
16
17    // construtor MouseTrackerFrame configura GUI e
18    // registra handlers de evento de mouse
19    public MouseTrackerFrame()
20    {
21       super( "Demonstrating Mouse Events" );
22
23       mousePanel = new JPanel(); // cria painel
24       mousePanel.setBackground( Color.WHITE ); // configura cor de fundo
25       add( mousePanel, BorderLayout.CENTER ); // adiciona
26
27       statusBar = new JLabel( "Mouse outside JPanel" );
28       add( statusBar, BorderLayout.SOUTH ); // adiciona rótulo ao JFrame
29
```

Cria `JPanel` para capturar eventos de mouse

Configura o fundo como branco

Cria `JLabel` e o adiciona à aplicação

```
30       // cria e registra listener para mouse e eventos de movimento de mouse
31       MouseHandler handler = new MouseHandler();
32       mousePanel.addMouseListener( handler );
33       mousePanel.addMouseMotionListener( handler );
34    } // fim do construtor MouseTrackerFrame
35
36    private class MouseHandler implements MouseListener,
37       MouseMotionListener
38    {
39       // handlers de evento MouseListener
40       // trata evento quando o mouse é liberado logo depois de pressionado
41       public void mouseClicked( MouseEvent event )
42       {
43          statusBar.setText( String.format( "Clicked at [%d, %d]",
44             event.getX(), event.getY() ) );
45       } // fim do método mouseClicked
46
47       // trata evento quando mouse é pressionado
48       public void mousePressed( MouseEvent event )
49       {
50          statusBar.setText( String.format( "Pressed at [%d, %d]",
51             event.getX(), event.getY() ) );
52       } // fim do método mousePressed
53
54       // trata evento quando mouse é liberado depois da operação de arrastar
55       public void mouseReleased( MouseEvent event )
56       {
57          statusBar.setText( String.format( "Released at [%d, %d]",
58             event.getX(), event.getY() ) );
59       } // fim do método mouseReleased
```

Cria handler de evento para eventos de mouse

Registra um handler de evento

Implementa interfaces ouvintes de mouse

(2 de 4)

Declara o método `mouseClicked`

Determina a localização do clique de mouse

Declara o método `mousePressed`

Declara o método `mouseReleased`

Resumo

MouseTracker

(3 de 4)

```
60
61       // trata evento quando mouse entra na área
62       public void mouseEntered( MouseEvent event )
63       {
64          statusBar.setText( String.format( "Mouse entered at [%d, %d]",
65             event.getX(), event.getY() ) );
66          mousePanel.setBackground( Color.GREEN );
67       } // fim do método mouseEntered
68
69       // trata evento quando mouse sai da área
70       public void mouseExited( MouseEvent event )
71       {
72          statusBar.setText( "Mouse outside JPanel" );
73          mousePanel.setBackground( Color.WHITE );
74       } // fim do método mouseExited
75
```

Declara o método `mouseEntered`

Configura o segundo plano de `JPanel`

Declara o método `mouseExited`

Configura o segundo plano de `JPanel`

Resumo

MouseTracker Frame.java

(4 de 4)

```
76       // MouseMotionListener event handlers
77       // trata evento MouseMotionListener
78       public void mouseDragged( MouseEvent event )
79       {
80          statusBar.setText( String.format( "Dragged at [%d, %d]",
81             event.getX(), event.getY() ) );
82       } // fim do método mouseDragged
83
84       // trata evento quanto usuário move o mouse
85       public void mouseMoved( MouseEvent event )
86       {
87          statusBar.setText( String.format( "Moved at [%d, %d]",
88             event.getX(), event.getY() ) );
89       } // fim do método mouseMoved
90    } // fim da classe MouseHandler interna
91 } // fim da classe MouseTrackerFrame
```

Declara o método `mouseDragged`

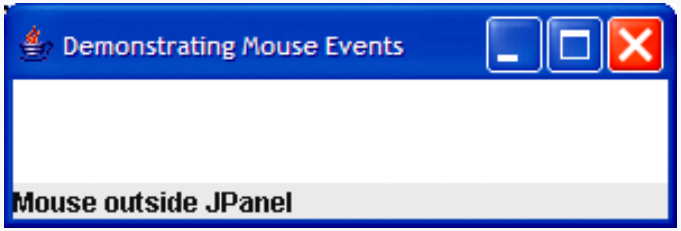
Declara o método `mouseMoved`

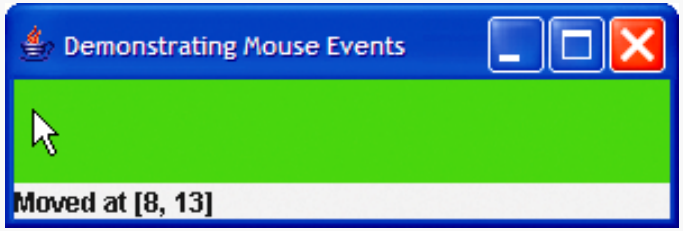
Resumo

MouseTracker Frame.java

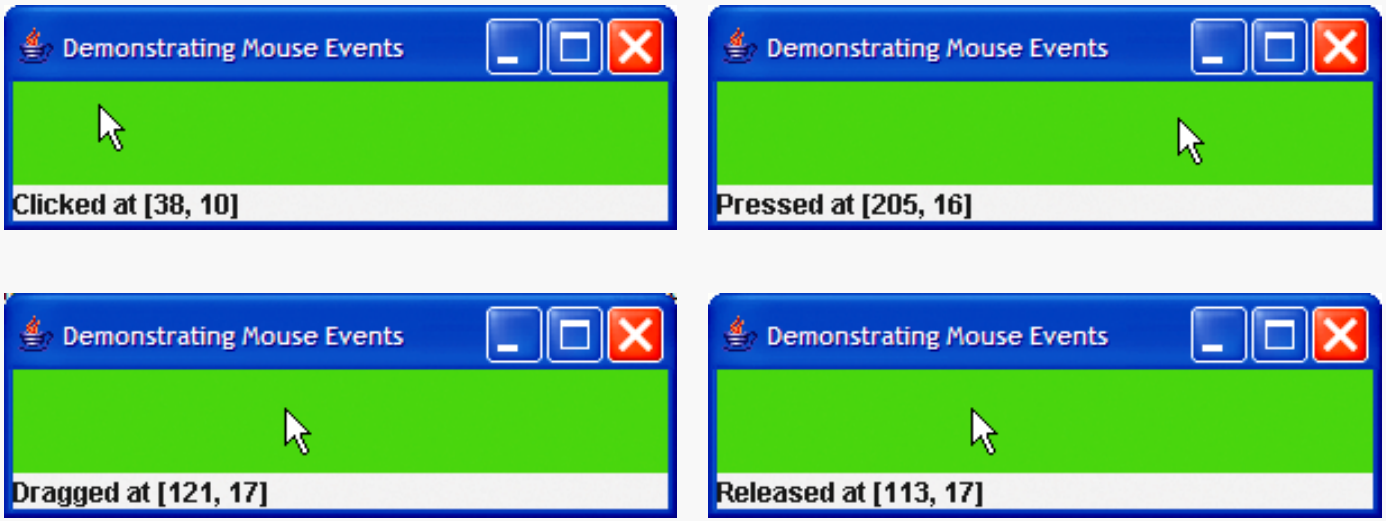
(1 de 2)

```
1  // Fig. 11.29: MouseTrackerFrame.java
2  // Testando MouseTrackerFrame.
3  import javax.swing.JFrame;
4
5  public class MouseTracker
6  {
7     public static void main( String args[] )
8     {
9        MouseTrackerFrame mouseTrackerFrame = new MouseTrackerFrame();
10       mouseTrackerFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       mouseTrackerFrame.setSize( 300, 100 ); // configura o tamanho do frame
12       mouseTrackerFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe MouseTracker
```





Resumo



`MouseTracker Frame.java`

(2 de 2)

11.14 Classes adaptadoras

- **Classe adaptadora:**
  - **Implementa interface ouvinte de evento.**
  - **Fornece implementação-padrão para todos os métodos de tratamento de eventos.**

Observação de engenharia de software 11.7

Quando uma classe implementa uma interface, a classe tem um relacionamento '*é um*' com essa interface. Todas as subclasses diretas e indiretas dessa classe herdam essa interface. Portanto, um objeto de uma classe que estende uma classe adaptadora de evento *é um* objeto do tipo ouvinte de eventos correspondente (por exemplo, um objeto de uma subclasse de `MouseAdapter` é um `MouseListener`).

Herdando `MouseAdapter`

- `MouseAdapter`:
  - **Classe adaptadora para as interfaces `MouseListener` e `MouseMotionListener`.**
  - **Estender a classe permite sobrescrever somente os métodos que você deseja utilizar.**

Erro comum de programação 11.4

Se você estender uma classe adaptadora e digitar incorretamente o nome do método que você está sobrescrevendo, o método simplesmente torna-se outro método na classe. Esse é um erro de lógica difícil de ser detectado, visto que o programa chamará a versão vazia do método herdado da classe adaptadora.

Classe adaptadora de evento em `java.awt.event`	Implementa interface
`ComponentAdapter`	`ComponentListener`
`ContainerAdapter`	`ContainerListener`
`FocusAdapter`	`FocusListener`
`KeyAdapter`	`KeyListener`
`MouseAdapter`	`MouseListener`
`MouseMotionAdapter`	`MouseMotionListener`
`WindowAdapter`	`WindowListener`

Figura 11.30 | Classes adaptadoras de evento e as interfaces que elas implementam no pacote `java.awt.event`.

Resumo

MouseDetails Frame.java

(1 de 2)

```
1  // Fig. 11.31: MouseDetailsFrame.java
2  // Demonstrando cliques de mouse e distinguindo entre botões do mouse.
3  import java.awt.BorderLayout;
4  import java.awt.Graphics;
5  import java.awt.event.MouseAdapter;
6  import java.awt.event.MouseEvent;
7  import javax.swing.JFrame;
8  import javax.swing.JLabel;
9
10 public class MouseDetailsFrame extends JFrame
11 {
12    private String details; // representação String
13    private JLabel statusBar; // JLabel que aparece no botão de janela
14
15    // construtor configura barra de título String e registra o listener de mouse
16    public MouseDetailsFrame()
17    {
18       super( "Mouse clicks and buttons" );
19
20       statusBar = new JLabel( "Click the mouse" );
21       add( statusBar, BorderLayout.SOUTH );
22       addMouseListener( new MouseClickHandler() ); // adiciona handler
23    } // fim do construtor MouseDetailsFrame
24
```

Registra um handler de evento

Resumo

MouseDetails Frame.java

(2 de 2)

```
25    // classe interna para tratar eventos de mouse
26    private class MouseClickHandler extends MouseAdapter
27    {
28       // trata evento de clique de mouse e determina qual botão foi pressionado
29       public void mouseClicked( MouseEvent event )
30       {
31          int xPos = event.getX(); // obtém posição x do mouse
32          int yPos = event.getY(); // obtém posição y do mouse
33
34          details = String.format( "Clicked %d time(s)",
35             event.getClickCount() );
36
37          if ( event.isMetaDown() ) // botão direito do mouse
38             details += " with right mouse button";
39          else if ( event.isAltDown() ) // botão do meio do mouse
40             details += " with center mouse button";
41          else // botão esquerdo do mouse
42             details += " with left mouse button";
43
44          statusBar.setText( details ); // exibe mensagem na statusBar
45       } // fim do método mouseClicked
46    } // fim da classe interna private MouseClickHandler
47 } // fim da classe MouseDetailsFrame
```

Obtém o número de vezes que o botão do mouse foi clicado

Testa se o botão direito do mouse foi clicado

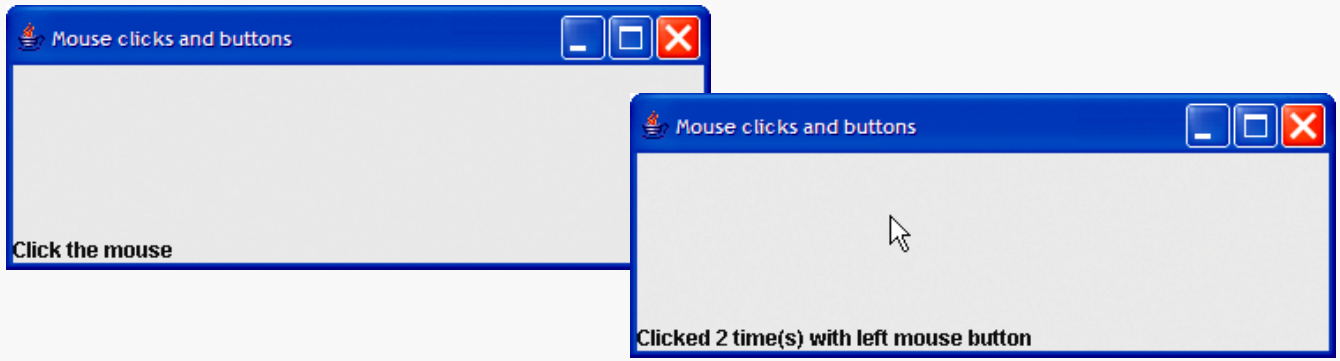
Testa se o botão do meio do mouse foi clicado

Resumo

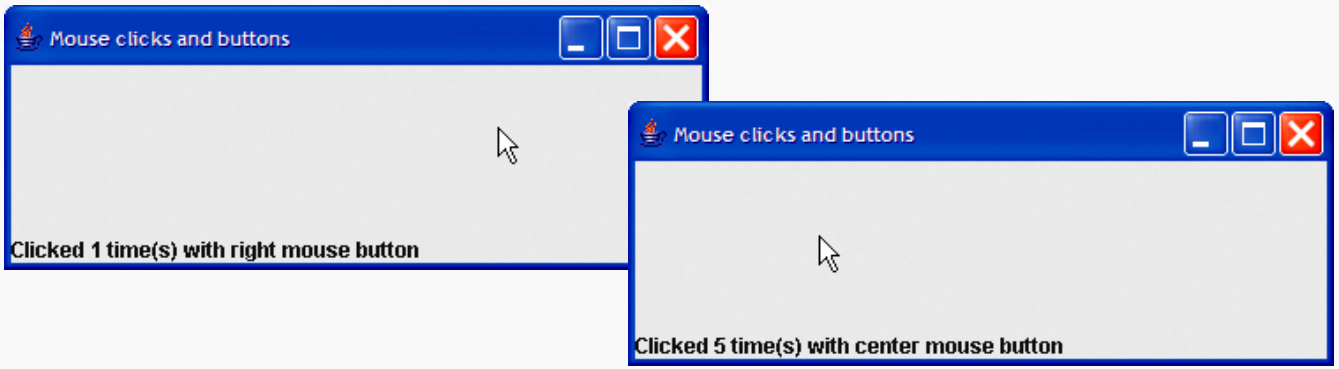
MouseDetails .java

(1 de 2)

```
1  // Fig. 11.32: MouseDetails.java
2  // Testando MouseDetailsFrame.
3  import javax.swing.JFrame;
4
5  public class MouseDetails
6  {
7     public static void main( String args[] )
8     {
9        MouseDetailsFrame mouseDetailsFrame = new MouseDetailsFrame();
10       mouseDetailsFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       mouseDetailsFrame.setSize( 400, 150 ); // configura o tamanho do frame
12       mouseDetailsFrame.setVisible( true ); // obtém o frame
13    } // fim de main
14 } // fim da classe MouseDetails
```



Resumo



`MouseDetails.java`

(2 de 2)

11.16 Tratamento de eventos de teclado

- **Interface** `KeyListener`:
  - **Para tratar eventos de teclado —** `KeyEvents`**.**
  - **Declara os métodos** `keyPressed`**,** `keyReleased` **e** `keyTyped`, sendo que cada um recebe um `KeyEvent` como seu argumento.

Resumo

KeyDemoFrame.java

(1 de 3)

```
1  // Fig. 11.36: KeyDemoFrame.java
2  // Demonstrando os eventos de pressionamento de tecla.
3  import java.awt.Color;
4  import java.awt.event.KeyListener;
5  import java.awt.event.KeyEvent;
6  import javax.swing.JFrame;
7  import javax.swing.JTextArea;
8
9  public class KeyDemoFrame extends JFrame implements KeyListener
10 {
11    private String line1 = ""; // primeira linha de textarea
12    private String line2 = ""; // segunda linha de tex
13    private String line3 = ""; // terceira linha de textarea
14    private JTextArea textArea; // textarea a exibir saída
15
16    // construtor KeyDemoFrame
17    public KeyDemoFrame()
18    {
19       super( "Demonstrating Keystroke Events" );
20
21       textArea = new JTextArea( 10, 15 ); // configura JTextArea
22       textArea.setText( "Press any key on the keyboard..." );
23       textArea.setEnabled( false ); // disativa texta
24       textArea.setDisabledTextColor( Color.BLACK ); /
25       add( textArea ); // adiciona textarea ao JFrame
26
27       addKeyListener( this ); // permite que o frame processe eventos de teclado
28    } // fim do construtor KeyDemoFrame
29
```

Implementa a interface `KeyListener`

Configura a cor de segundo plano

Registra a própria aplicação como um handler de evento

Resumo

```
30    // trata pressionamento de qualquer tecla
31    public void keyPressed( KeyEvent event )
32    {
33       line1 = String.format( "Key pressed: %s",
34          event.getKeyText( event.getKeyCode() ) ); // gera saída de tecla pressionada
35       setLines2and3( event ); // configura a saída da
36    } // fim do método keyPressed
37
38    // trata liberação de qualquer tecla
39    public void keyReleased( KeyEvent event )
40    {
41       line1 = String.format( "Key released: %s",
42          event.getKeyText( event.getKeyCode() ) ); // gera saída de tecla liberada
43       setLines2and3( event ); // configura a saída da
44    } // fim do método keyReleased
45
46    // trata pressionamento de qualquer tecla de ação
47    public void keyTyped( KeyEvent event )
48    {
49       line1 = String.format( "Key typed: %s", event.getKeyChar() );
50       setLines2and3( event ); // configura saída das linhas dois e três
51    } // fim do método keyTyped
52
```

Declara o método `keyPressed`

Obtém o código da tecla pressionada

Frame
.java

(2 de 3)

Declara o método `keyReleased`

Obtém o código da tecla liberada

Declara o método `keyTyped`

Gera saída do caractere digitado

Resumo

```
53    // configura segunda e terceira linhas de saída
54    private void setLines2and3( KeyEvent event )
55    {
56       line2 = String.format( "This key is %san action key",
57          ( event.isActionKey() ? "" : "not " ) );
58
59       String temp = event.getKeyModifiersText( event.getModifiers() );
60
61       line3 = String.format( "Modifier keys pressed: %
62          ( temp.equals( "" ) ? "none" : temp ) ); // e
63
64       textArea.setText( String.format( "%s\n%s\n%s\n",
65          line1, line2, line3 ) ); // gera saída de três linhas de texto
66    } // fim do método setLines2and3
67 } // fim da classe KeyDemoFrame
```

Testa se era uma tecla de ação

KeyDemoFrame.java

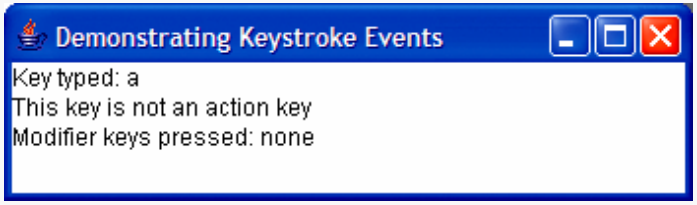
Determina quaisquer modificadores pressionados

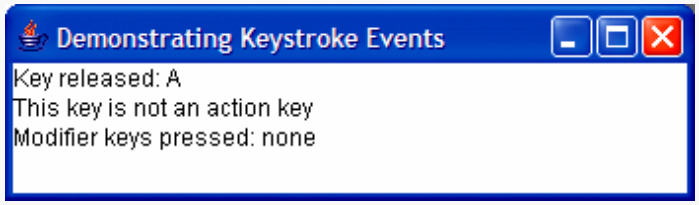
Resumo

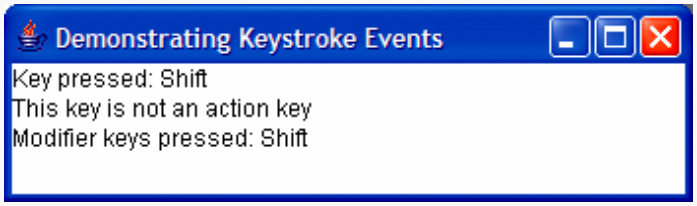
KeyDemo.java

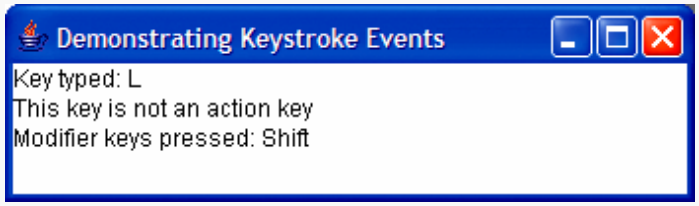
(1 de 2)

```
1  // Fig. 11.37: KeyDemo.java
2  // Testando KeyDemoFrame.
3  import javax.swing.JFrame;
4
5  public class KeyDemo
6  {
7     public static void main( String args[] )
8     {
9        KeyDemoFrame keyDemoFrame = new KeyDemoFrame();
10       keyDemoFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       keyDemoFrame.setSize( 350, 100 ); // configura o tamanho do frame
12       keyDemoFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe KeyDemo
```









Resumo

`KeyDemo.java`

(2 de 2)

11.17 Gerenciadores de layout

- **Gerenciadores de layout:**
  - **Fornecidos para organizar componentes GUI em um contêiner.**
  - **Fornecem as capacidades básicas de layout.**
  - **Implementam a interface** `LayoutManager`.

Observação sobre aparência e comportamento 11.16

A maioria dos ambientes de programação do Java fornece ferramentas de desenho GUI que ajudam um programador a projetar uma GUI graficamente; as ferramentas de desenho então escrevem o código Java para criar a GUI. Essas ferramentas costumam fornecer maior controle sobre o tamanho, a posição e o alinhamento de componentes GUI do que os gerenciadores de layouts predefinidos.

Observação sobre aparência e comportamento 11.17

É possível configurar o layout de um `Container` como `null`, o que indica que nenhum gerenciador de layout deve ser utilizado. Em um `Container` sem gerenciador de layout, o programador deve posicionar e dimensionar os componentes no contêiner dado e cuidar para que, em eventos de redimensionamento, todos os componentes sejam reposicionados conforme necessário. Os eventos de redimensionamento de um componente podem ser processados por um `ComponentListener`.

11.17.1 `FlowLayout`

- `FlowLayout`:
  - **É o gerenciador de layout mais simples.**
  - **Os componentes GUI são colocados em um contêiner da esquerda para a direita na ordem em que eles são adicionados ao contêiner.**
  - **Os componentes podem ser alinhados à esquerda, centralizados ou alinhados à esquerda.**

Gerenciador de layout	Descrição
`FlowLayout`	Padrão para `javax.swing.JPanel`. Coloca os componentes seqüencialmente (da esquerda para a direita) na ordem que foram adicionados. Também é possível especificar a ordem dos componentes utilizando o método `Container` method `add`, que aceita um Component e uma posição de índice do tipo inteiro como argumentos.
`BorderLayout`	Padrão para `JFrame`s (e outras janelas). Organiza os componentes em cinco áreas: `NORTH`, `SOUTH`, `EAST`, `WEST` e `CENTER`.
`GridLayout`	Organiza os componentes nas linhas e colunas.

Figura 11.38 | Gerenciadores de layout.

Resumo

FlowLayoutFrame.java

(1 de 3)

```
1  // Fig. 11.39: FlowLayoutFrame.java
2  // Demonstrando os alinhamentos de FlowLayout.
3  import java.awt.FlowLayout;
4  import java.awt.Container;
5  import java.awt.event.ActionListener;
6  import java.awt.event.ActionEvent;
7  import javax.swing.JFrame;
8  import javax.swing.JButton;
9
10 public class FlowLayoutFrame extends JFrame
11 {
12    private JButton leftJButton; // botão para configurar alinhamento à esquerda
13    private JButton centerJButton; // botão para configurar alinhamento centralizado
14    private JButton rightJButton; // botão para configurar alinhamento à direita
15    private FlowLayout layout; // objeto de layout
16    private Container container; // contêiner para configurar layout
17
18    // configura GUI e registra listeners de botão
19    public FlowLayoutFrame()
20    {
21       super( "FlowLayout Demo" );
22
23       layout = new FlowLayout(); // cria FlowLayout
24       container = getContentPane(); // obtém contêiner para layout
25       setLayout( layout ); // configura layout do frame
26
```

Cria `FlowLayout`

Configura o layout da aplicação

Resumo

```
27       // configura leftJButton e registra listener
28       leftJButton = new JButton( "Left" ); // cria botão Left
29       add( leftJButton ); // adiciona o botão Left ao frame
30       leftJButton.addActionListener(
31
32          new ActionListener() // classe interna anôni
33          {
34             // processa o evento leftJButton
35             public void actionPerformed( ActionEvent event )
36             {
37                layout.setAlignment( FlowLayout.LEFT );
38
39                // realinha os componentes anexados
40                layout.layoutContainer( container );
41             } // fim do método actionPerformed
42          } // fim da classe interna anônima
43       ); // fim da chamada para addActionListener
44
45       // configura centerJButton e registra listener
46       centerJButton = new JButton( "Center" ); // cria botão Center
47       add( centerJButton ); // adiciona botão Center ao frame
48       centerJButton.addActionListener(
49
50          new ActionListener() // classe interna anôni
51          {
52             // processa evento centerJButton
53             public void actionPerformed( ActionEvent
54             {
55                layout.setAlignment( FlowLayout.CENTER );
56
```

Adiciona `JButton`; `FlowLayout` tratará o posicionamento

`FlowLayoutFrame.java`

(2 de 3)

Configura o alinhamento à esquerda

Ajusta o layout

Adiciona `JButton`; `FlowLayout` tratará o posicionamento

Configura o alinhamento no centro

Resumo

FlowLayoutFrame.java

(3 de 3)

```
57                // realinha os componentes anexados
58                layout.layoutContainer( container );
59             } // fim do método actionPerformed
60          } // fim da classe interna anônima
61       ); // fim da chamada para addActionListener
62
63       // configura rightJButton e registra listener
64       rightJButton = new JButton( "Right" ); // cria botão Right
65       add( rightJButton ); // adiciona botão Right ao frame
66       rightJButton.addActionListener(
67
68          new ActionListener() // classe interna anôni
69          {
70             // processa o evento rightJButton
71             public void actionPerformed( ActionEvent event )
72             {
73                layout.setAlignment( FlowLayout.RIGHT );
74
75                // realinha os componentes anexados
76                layout.layoutContainer( container );
77             } // fim do método actionPerformed
78          } // fim da classe interna anônima
79       ); // fim da chamada para addActionListener
80    } // fim do construtor FlowLayoutFrame
81 } // fim da classe FlowLayoutFrame
```

Ajusta o layout

Adiciona `JButton`; `FlowLayout` tratará o posicionamento

Configura o alinhamento à direita

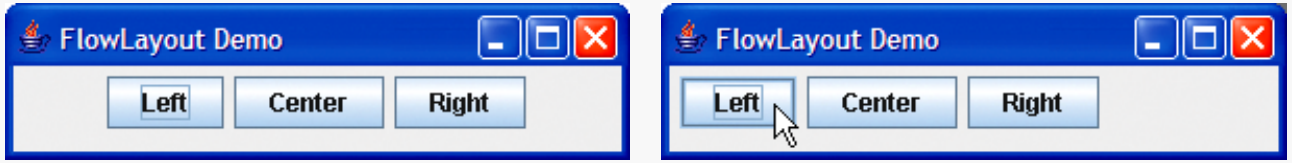
Ajusta o layout

Resumo

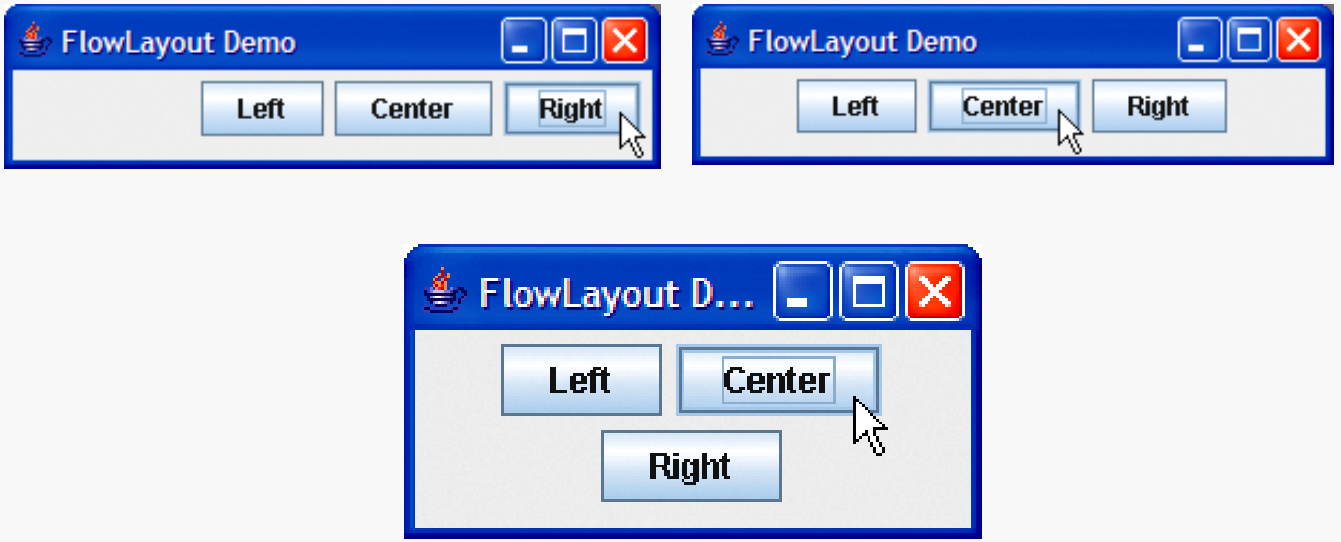
FlowLayoutDemo.java

(1 de 2)

```
1  // Fig. 11.40: FlowLayoutDemo.java
2  // Testando FlowLayoutFrame.
3  import javax.swing.JFrame;
4
5  public class FlowLayoutDemo
6  {
7     public static void main( String args[] )
8     {
9        FlowLayoutFrame flowLayoutFrame = new FlowLayoutFrame();
10       flowLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       flowLayoutFrame.setSize( 300, 75 ); // configura o tamanho do frame
12       flowLayoutFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe FlowLayoutDemo
```



Resumo



`FlowLayoutDemo.java`

(2 de 2)

11.17.2 BorderLayout

- BorderLayout:
  - **Organiza os componentes em cinco regiões — norte, sul, leste, oeste e centro.**
  - **Implementa a interface** LayoutManager2.
  - **Fornece o espaçamento da lacuna horizontal e o espaçamento da lacuna vertical.**

Observação sobre aparência e comportamento 11.18

Todo contêiner pode ter apenas um gerenciador de layout. Os contêineres separados no mesmo aplicativo podem utilizar diferentes gerenciadores de layout.

Observação sobre a aparência e comportamento 11.19

Se nenhuma região for especificada ao adicionar um `Component` para um `BorderLayout`, o gerenciador de layout assume que o `Component` deve ser adicionado à região `BorderLayout.CENTER`.

Erro comum de programação 11.6

Quando mais de um componente é adicionado a uma região em um `BorderLayout`, somente o último componente adicionado a essa região será exibido. Não há nenhum erro que indica esse problema.

Resumo

BorderLayout Frame.java

(1 de 2)

```
1  // Fig. 11.41: BorderLayoutFrame.java
2  // Demonstrando BorderLayout.
3  import java.awt.BorderLayout;
4  import java.awt.event.ActionListener;
5  import java.awt.event.ActionEvent;
6  import javax.swing.JFrame;
7  import javax.swing.JButton;
8
9  public class BorderLayoutFrame extends JFrame implements ActionListener
10 {
11    private JButton buttons[]; // array de botões para ocultar partes
12    private final String names[] = { "Hide North", "Hide South",
13       "Hide East", "Hide West", "Hide Center" };
14    private BorderLayout layout; // objeto borderlayout
15
16    // configura GUI e tratamento de evento
17    public BorderLayoutFrame()
18    {
19       super( "BorderLayout Demo" );
20
21       layout = new BorderLayout( 5, 5 ); // 5 pixel g
22       setLayout( layout ); // configura o layout de frame
23       buttons = new JButton[ names.length ]; // configura o tamanho do array
24
25       // cria JButtons e registra listeners para eles
26       for ( int count = 0; count < names.length; count++ )
27       {
28          buttons[ count ] = new JButton( names[ count ] );
29          buttons[ count ].addActionListener( this );
30       } // fim de for
```

Declara a variável de instância `BorderLayout`

Cria `BorderLayout`

Configura o layout

Registra um handler de evento

Resumo

BorderLayout

(2 de 2)

```
31
32       add( buttons[ 0 ], BorderLayout.NORTH ); // adiciona botão para o norte
33       add( buttons[ 1 ], BorderLayout.SOUTH ); // adiciona botão para o sul
34       add( buttons[ 2 ], BorderLayout.EAST ); // adiciona botão para o leste
35       add( buttons[ 3 ], BorderLayout.WEST ); // adiciona botão para o oeste
36       add( buttons[ 4 ], BorderLayout.CENTER ); // adiciona botão para o centro
37    } // fim do construtor BorderLayoutFrame
38
39    // trata eventos de botão
40    public void actionPerformed( ActionEvent event )
41    {
42       // verifica a origem de evento e o painel de conteúdo de layout correspondentemente
43       for ( JButton button : buttons )
44       {
45          if ( event.getSource() == button )
46             button.setVisible( false ); // oculta bot
47          else
48             button.setVisible( true ); // mostra outros botões
49       } // fim de for
50
51       layout.layoutContainer( getContentPane() ); // painel de conteúdo de layout
52    } // fim do método actionPerformed
53 } // fim da classe BorderLayoutFrame
```

Adiciona botões à aplicação utilizando as constantes do gerenciador de layout

Torna o botão invisível

Torna o botão visível

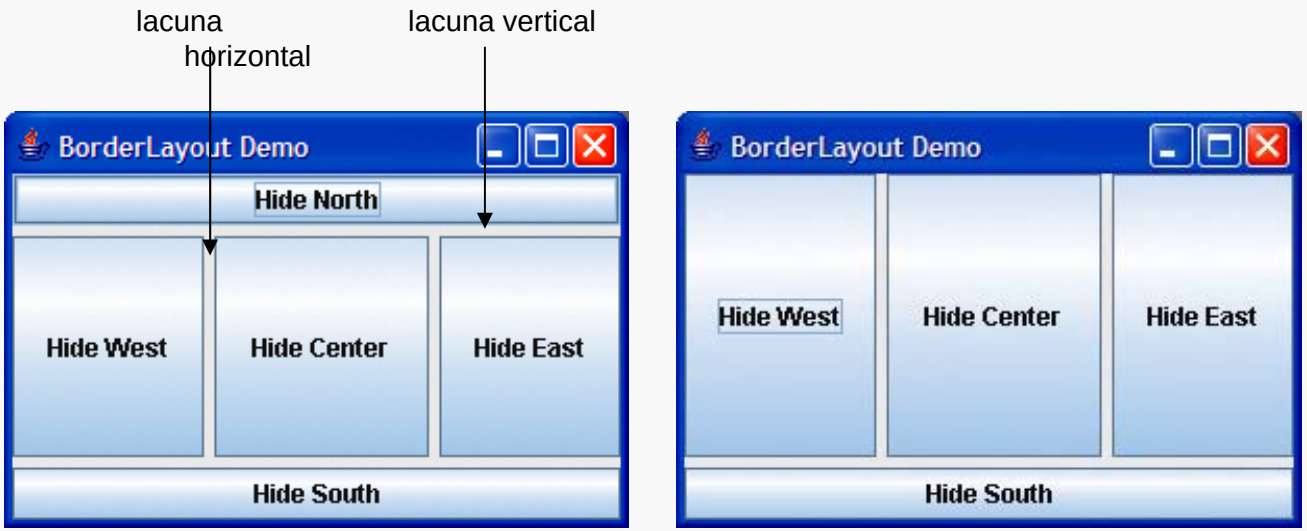
Atualiza o layout

Resumo

BorderLayoutDemo.java

(1 de 2)

```
1  // Fig. 11.42: BorderLayoutDemo.java
2  // Testando BorderLayoutFrame.
3  import javax.swing.JFrame;
4
5  public class BorderLayoutDemo
6  {
7     public static void main( String args[] )
8     {
9        BorderLayoutFrame borderLayoutFrame = new BorderLayoutFrame();
10       borderLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       borderLayoutFrame.setSize( 300, 200 ); // configura o tamanho do frame
12       borderLayoutFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe BorderLayoutDemo
```



Resumo

`BorderLayout Demo.java`

(2 de 2)

11.17.3 `GridLayout`

- `GridLayout`:
  - **Divide o contêiner em uma grade.**
  - **Todos os componentes têm a mesma largura e altura.**

Resumo

GridLayout Frame.java

(1 de 2)

```
1  // Fig. 11.43: GridLayoutFrame.java
2  // Demonstrando GridLayout.
3  import java.awt.GridLayout;
4  import java.awt.Container;
5  import java.awt.event.ActionListener;
6  import java.awt.event.ActionEvent;
7  import javax.swing.JFrame;
8  import javax.swing.JButton;
9
10 public class GridLayoutFrame extends JFrame implements ActionListener
11 {
12    private JButton buttons[]; // array de botões
13    private final String names[] =
14       { "one", "two", "three", "four", "five", "six"
15    private boolean toggle = true; // alterna entre do
16    private Container container; // contêiner do frame
17    private GridLayout gridLayout1; // primeiro gridlayout
18    private GridLayout gridLayout2; // segundo gridlayout
19
20    // construtor sem argumento
21    public GridLayoutFrame()
22    {
23       super( "GridLayout Demo" );
24       gridLayout1 = new GridLayout( 2, 3, 5, 5 ); // 2 por 3; lacunas de 5
25       gridLayout2 = new GridLayout( 3, 2 ); // 3 por 2; sem lacunas
26       container = getContentPane(); // obtém painel de conteúdo
27       setLayout( gridLayout1 ); // configura layout do JFrame
28       buttons = new JButton[ names.length ];
29
```

Declara duas variáveis de instância `GridLayout`

Cria `GridLayout`

Configura o layout

Resumo

```
30       for ( int count = 0; count < names.length; count++ )
31       {
32          buttons[ count ] = new JButton( names[ count ] );
33          buttons[ count ].addActionListener( this ); // registra listener
34          add( buttons[ count ] ); // adiciona botão ao JFrame
35       } // fim de for
36    } // fim do construtor GridLayoutFrame
37
38    // trata eventos de botão alternando entre layouts
39    public void actionPerformed( ActionEvent event )
40    {
41       if ( toggle )
42          container.setLayout( gridLayout2 ); // confi
43       else
44          container.setLayout( gridLayout1 ); // configura layout como primeiro
45
46       toggle = !toggle; // alterna para valor oposto
47       container.validate(); // refaz o layout do contêiner
48    } // fim do método actionPerformed
49 } // fim da classe GridLayoutFrame
```

Adiciona o botão ao `JFrame`

Utiliza o segundo layout

Utiliza o primeiro layout

Atualiza o layout

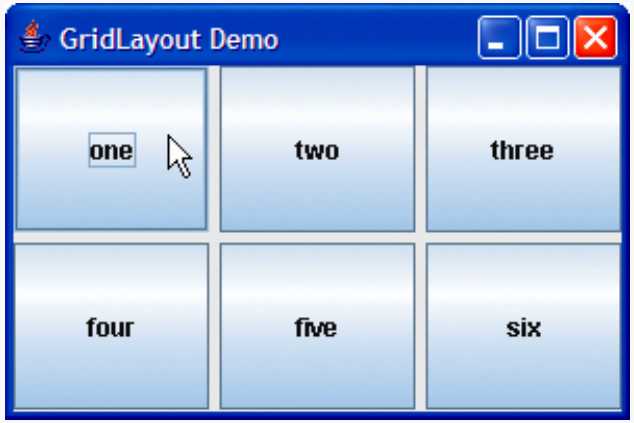
`GridLayoutFrame.java`

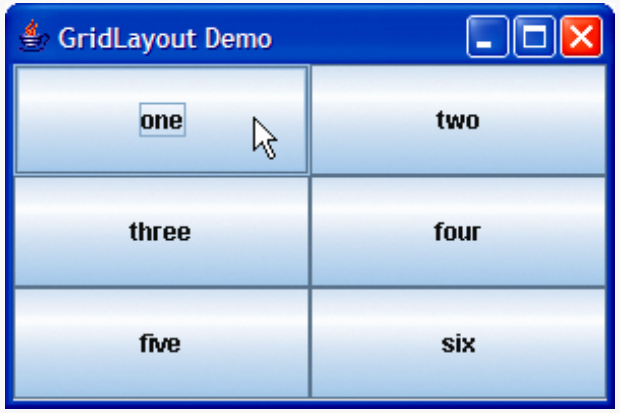
(2 de 2)

Resumo

GridLayoutDemo.java

```
1  // Fig. 11.44: GridLayoutDemo.java
2  // Testando GridLayoutFrame.
3  import javax.swing.JFrame;
4
5  public class GridLayoutDemo
6  {
7     public static void main( String args[] )
8     {
9        GridLayoutFrame gridLayoutFrame = new GridLayoutFrame();
10       gridLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       gridLayoutFrame.setSize( 300, 200 ); // configura o tamanho do frame
12       gridLayoutFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe GridLayoutDemo
```





11.18 Utilizando painéis para gerenciar layouts mais complexos

- **GUIs complexas freqüentemente requerem múltiplos painéis para organizar seus componentes adequadamente.**

Resumo

PanelFrame.java

(1 de 2)

```
1  // Fig. 11.45: PanelFrame.java
2  // Utilizando um JPanel para ajudar a fazer o layout dos componentes.
3  import java.awt.GridLayout;
4  import java.awt.BorderLayout;
5  import javax.swing.JFrame;
6  import javax.swing.JPanel;
7  import javax.swing.JButton;
8
9  public class PanelFrame extends JFrame
10 {
11    private JPanel buttonJPanel; // painel para armazenar botões
12    private JButton buttons[]; // array de botões
13
14    // construtor sem argumentos
15    public PanelFrame()
16    {
17       super( "Panel Demo" );
18       buttons = new JButton[ 5 ]; // cria array de botões
19       buttonJPanel = new JPanel(); // configura painel
20       buttonJPanel.setLayout( new GridLayout( 1, buttons.length ) );
21
```

Declara um `JPanel` para conter os botões

Cria o `JPanel`

Configura o layout

Resumo

```
22       // cria e adiciona botões
23       for ( int count = 0; count < buttons.length; count++ )
24       {
25          buttons[ count ] = new JButton( "Button " + ( count + 1 ) );
26          buttonJPanel.add( buttons[ count ] ); // adiciona botão ao painel
27       } // end for
28
29       add( buttonJPanel, BorderLayout.SOUTH ); // adiciona painel ao JFrame
30    } // fim do construtor PanelFrame
31 } // fim da classe PanelFrame
```

Adiciona um botão ao painel

Adiciona o painel à aplicação

rame.java

(2 de 2)

Resumo

PanelDemo.java

```
1  // Fig. 11.46: PanelDemo.java
2  // Testando PanelFrame.
3  import javax.swing.JFrame;
4
5  public class PanelDemo extends JFrame
6  {
7     public static void main( String args[] )
8     {
9        PanelFrame panelFrame = new PanelFrame();
10       panelFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       panelFrame.setSize( 450, 200 ); // configura o tamanho do frame
12       panelFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe PanelDemo
```

11.19 JTextArea

- JTextArea:
  - **Fornece uma área para manipular múltiplas linhas de texto.**
- **Contêiner** Box:
  - **Subclasse de** Container.
  - **Utiliza um gerenciador de layout** BoxLayout.

Observação sobre aparência e comportamento 11.20

Para fornecer a funcionalidade de mudança de linha automática para uma `JTextArea`, invoque o método `JTextArea` `setLine-Wrap` com um argumento `true`.

Resumo

TextAreaFrame.java

(1 de 2)

```
1  // Fig. 11.47: TextAreaFrame.java
2  // Copiando texto selecionado de uma textarea para a outra.
3  import java.awt.event.ActionListener;
4  import java.awt.event.ActionEvent;
5  import javax.swing.Box;
6  import javax.swing.JFrame;
7  import javax.swing.JTextArea;
8  import javax.swing.JButton;
9  import javax.swing.JScrollPane;
10
11 public class TextAreaFrame extends JFrame
12 {
13    private JTextArea textArea1; // exibe string demo
14    private JTextArea textArea2; // texto destacado é copiado aqui
15    private JButton copyJButton; // começa a copiar o texto
16
17    // construtor sem argumentos
18    public TextAreaFrame()
19    {
20       super( "TextArea Demo" );
21       Box box = Box.createHorizontalBox(); // cria box
22       String demo = "This is a demo string to\n" +
23          "illustrate copying text\nfrom one textarea to \n" +
24          "another textarea using an\nexternal event\n";
25
26       textArea1 = new JTextArea( demo, 10, 15 ); // cria textarea1
27       box.add( new JScrollPane( textArea1 ) ); // adiciona scrollpane
28
```

Declara as variáveis de instância `JTextArea`

Cria um contêiner `Box`

Cria uma área de texto e a adiciona à caixa

Resumo

TextAreaFrame.java

(2 de 2)

```
29       copyJButton = new JButton( "Copy >>>" ); // cria botão de cópia
30       box.add( copyJButton ); // adiciona o botão de cópia à box
31       copyJButton.addActionListener(
32
33          new ActionListener() // classe interna anônima
34          {
35             // configura texto em textArea2 como texto selecionado de textArea1
36             public void actionPerformed( ActionEvent event )
37             {
38                textArea2.setText( textArea1.getSelectedText() );
39             } // fim do método actionPerformed
40          } // fim da classe interna anônima
41       ); // fim da chamada para addActionListener
42
43       textArea2 = new JTextArea( 10, 15 ); // cria segunda textarea
44       textArea2.setEditable( false ); // desativa a edição
45       box.add( new JScrollPane( textArea2 ) ); // adiciona scrollpane
46
47       add( box ); // adiciona box ao frame
48    } // fim do construtor TextAreaFrame
49 } // fim da classe TextAreaFrame
```

Adiciona o botão à caixa

Copia o texto selecionado de uma área de texto para outra

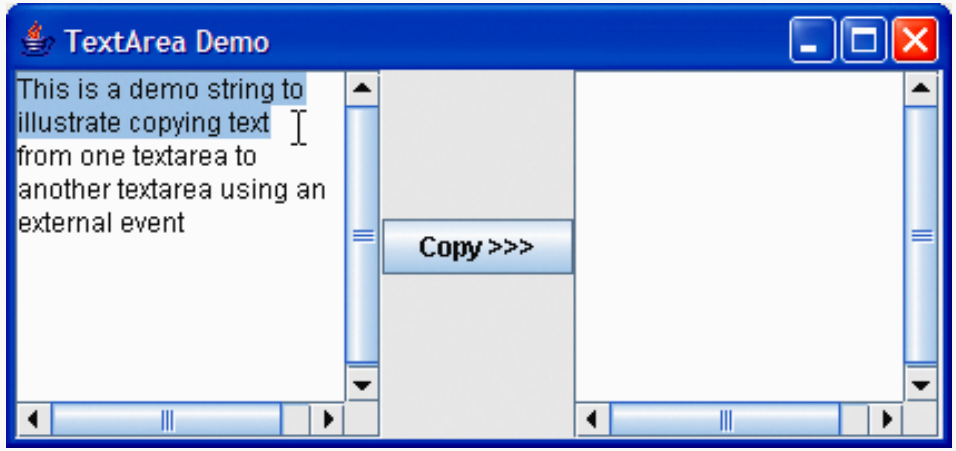
Cria uma segunda área de texto e a adiciona à caixa

Resumo

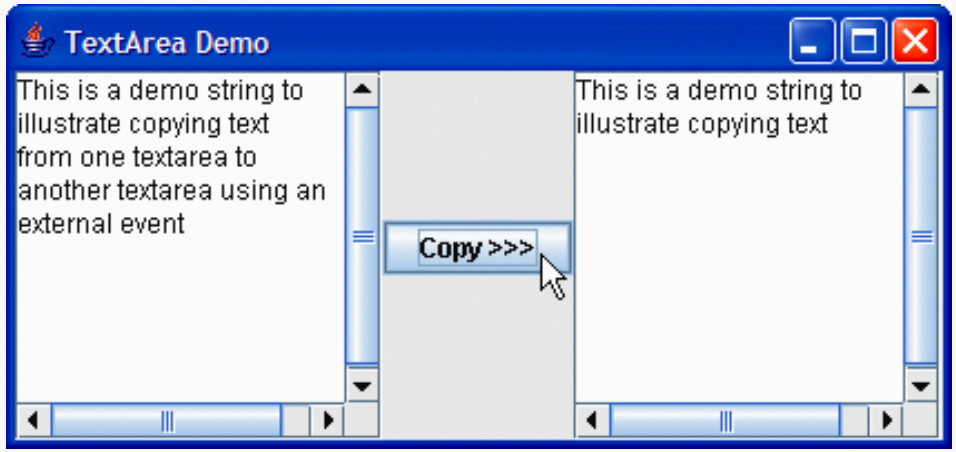
TextAreaDemo .java

(1 de 2)

```
1  // Fig. 11.48: TextAreaDemo.java
2  // Copiando texto selecionado de uma textarea para a outra.
3  import javax.swing.JFrame;
4
5  public class TextAreaDemo
6  {
7     public static void main( String args[] )
8     {
9        TextAreaFrame textAreaFrame = new TextAreaFrame();
10       textAreaFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       textAreaFrame.setSize( 425, 200 ); // configura o tamanho do frame
12       textAreaFrame.setVisible( true ); // exibe o frame
13    } // fim de main
14 } // fim da classe TextAreaDemo
```



Resumo



`TextAreaDemo.java`

(2 de 2)

- `JScrollPane` **tem diretivas de barra de rolagem:**
  - **Diretivas horizontais:**
    - **Sempre (**`HORIZONTAL_SCROLLBAR_ALWAYS`**).**
    - **Conforme necessário (**`HORIZONTAL_SCROLLBAR_AS_NEEDED`**).**
    - **Nunca (**`HORIZONTAL_SCROLLBAR_NEVER`**).**
  - **Diretivas verticais:**
    - **Sempre (**`VERTICAL_SCROLLBAR_ALWAYS`**).**
    - **Conforme necessário (**`VERTICAL_SCROLLBAR_AS_NEEDED`**).**
    - **Nunca (**`VERTICAL_SCROLLBAR_NEVER`**).**