Java – Aula 03

Continuação de interfaces gráficas – GUI Swing

29/08/2012

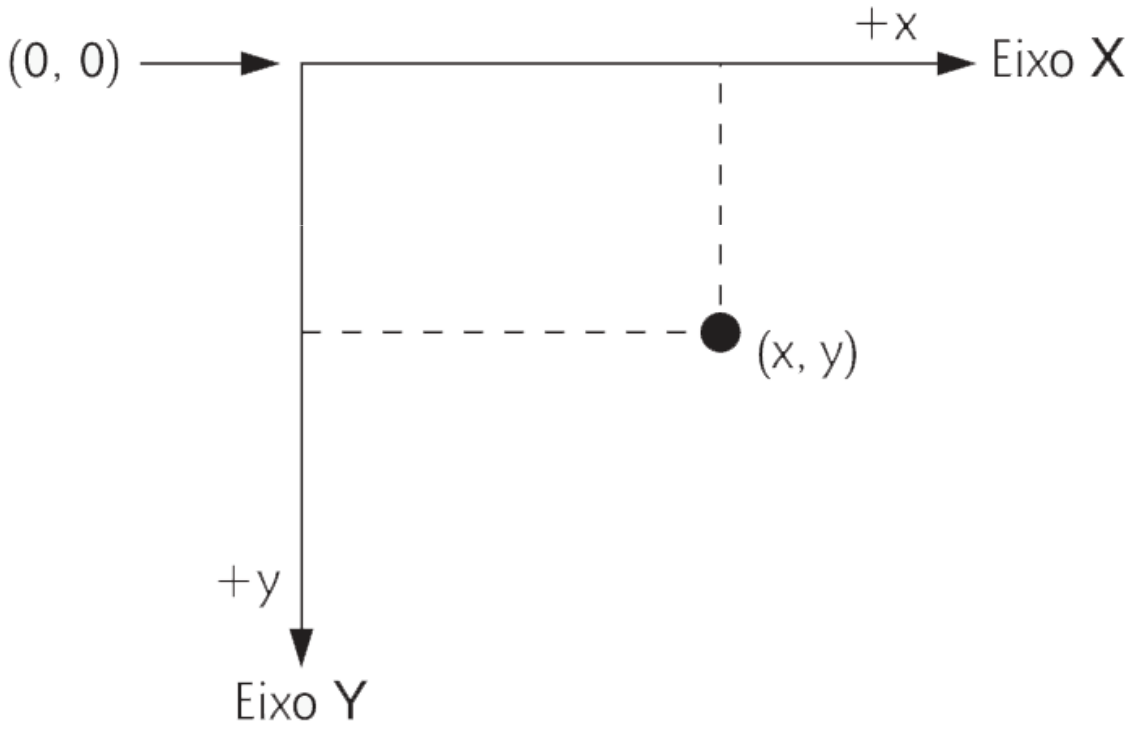
Celso Olivete Júnior

olivete@fct.unesp.br

GUIs e imagens gráficas: Criando desenhos simples

- Sistema de coordenadas do Java:
  - Definido pelas coordenadas x e coordenadas y.
    - Também conhecidas como coordenadas horizontais e verticais.
    - São medidas ao longo do eixo X e do eixo Y.
  - Unidades coordenadas são medidas em pixels.
- Classe Graphics no pacote java.awt
  - Fornece os métodos para desenhar texto e formas.
- Classe JPanel no pacote javax.swing
  - Fornece uma área para desenhar.
- Classe Jframe pode ser utilizada para o desenho, mas não é conveniente, porque o espaço da barra de título precisa ser levado em conta no cálculo dascoordenadas y.

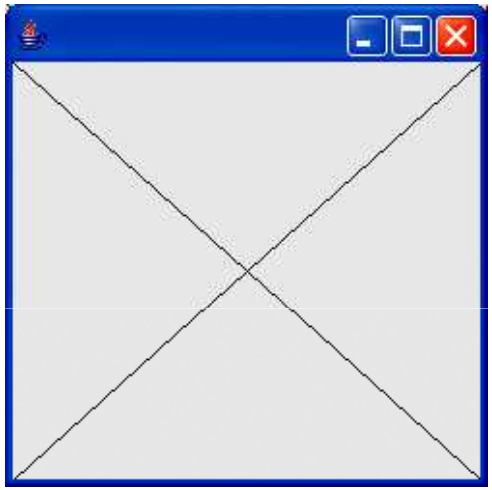
Sistema de coordenadas do Java: unidades em *pixels*

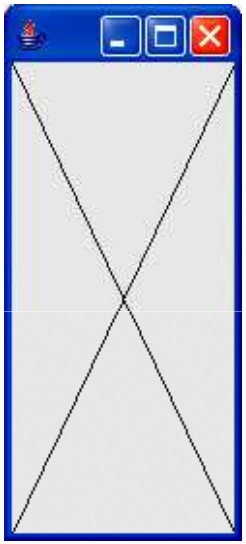


Criando desenhos simples (Continuação)

- Herança:
  - Palavra-chave extends.
  - A subclasse herda da superclasse:
    - A subclasse tem os dados e os métodos que a superclasse tem, bem como aqueles que ela define para si própria.

Exemplo: Linhas que se estendem a partir de um canto





Exemplo

```
1  // Fig. 4.19: DrawPanel.java
2  // Desenha duas linhas que se cruzam em um painel.
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class DrawPanel extends JPanel
7  {
8     // desenha um X a partir dos cantos do painel
9     public void paintComponent( Graphics g )
10    {
11       // chama paintComponent para assegurar que o painel é exibido corretamente
12       super.paintComponent( g );
13
14       int width = getWidth(); // largura total
15       int height = getHeight(); // altura total
16
17       // desenha uma linha a partir do canto superior esquerdo até o inferior direito
18       g.drawLine( 0, 0, width, height );
19
20       // desenha uma linha a partir do canto inferior esquerdo até o superior direito
21       g.drawLine( 0, height, width, 0 );
22    } // fim do método paintComponent
23 } // fim da classe DrawPanel
```

Exemplo

```
1  // Fig. 4.19: DrawPanel.java
2  // Desenha duas linhas que se cruzam em um pa
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class DrawPanel extends JPanel
7  {
8     // desenha um X a partir dos cantos do painel
9     public void paintComponent( Graphics g )
10    {
11       // chama paintComponent para assegurar que o painel é exibido corretamente
12       super.paintComponent( g );
13
14       int width = getWidth(); // largura total
15       int height = getHeight(); // altura total
16
17       // desenha uma linha a partir do canto superior esquerdo até o inferior direito
18       g.drawLine( 0, 0, width, height );
19
20       // desenha uma linha a partir do canto inferior esquerdo até o superior direito
21       g.drawLine( 0, height, width, 0 );
22    } // fim do método paintComponent
23 } // fim da classe DrawPanel
```

Importa as classes `java.awt.Graphics` e `javax.swing.JPanel`

Exemplo

```
1  // Fig. 4.19: DrawPanel.java
2  // Desenha duas linhas que se cruzam em um pa
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class DrawPanel extends JPanel
7  {
8     // desenha um X a partir dos cantos do painel
9     public void paintComponent( Graphics g )
10    {
11       // chama paintComponent para assegurar que o painel é exibido corretamente
12       super.paintComponent( g );
13
14       int width = getWidth(); // largura total
15       int height = getHeight(); // altura total
16
17       // desenha uma linha a partir do canto superior esquerdo até o inferior direito
18       g.drawLine( 0, 0, width, height );
19
20       // desenha uma linha a partir do canto inferior esquerdo até o superior direito
21       g.drawLine( 0, height, width, 0 );
22    } // fim do método paintComponent
23 } // fim da classe DrawPanel
```

Importa as classes **java.awt.Graphics** e **javax.swing.JPanel**

A classe **DrawPanel** estende a classe **JPanel**

Exemplo

```
1  // Fig. 4.19: DrawPanel.java
2  // Desenha duas linhas que se cruzam em um pa
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class DrawPanel extends JPanel
7  {
8     // desenha um X a partir dos cantos do painel
9     public void paintComponent( Graphics g )
10    {
11       // chama paintComponent para assegurar que o painel é exibido corretamente
12       super.paintComponent( g );
13
14       int width = getWidth(); // largura total
15       int height = getHeight(); // altura total
16
17       // desenha uma linha a partir do canto superior esquerdo até o inferior direito
18       g.drawLine( 0, 0, width, height );
19
20       // desenha uma linha a partir do canto inferior esquerdo até o superior direito
21       g.drawLine( 0, height, width, 0 );
22    } // fim do método paintComponent
23 } // fim da classe DrawPanel
```

Importa as classes **java.awt.Graphics** e **javax.swing.JPanel**

A classe **DrawPanel** estende a classe **JPanel**

Declara o método **paintComponent** – chamado quando o componente é criado

Todo desenho é feito neste método

Exemplo

```
1  // Fig. 4.19: DrawPanel.java
2  // Desenha duas linhas que se cruzam em um pa
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class DrawPanel extends JPanel
7  {
8     // desenha um X a partir dos cantos do painel
9     public void paintComponent( Graphics g )
10    {
11       // chama paintComponent para assegurar que o painel é exibido corretamente
12       super.paintComponent( g );
13
14       int width = getWidth(); // largura total
15       int height = getHeight(); // altura total
16
17       // desenha uma linha a partir do canto superior esquerdo até o inferior direito
18       g.drawLine( 0, 0, width, height );
19
20       // desenha uma linha a partir do canto inferior esquerdo até o superior direito
21       g.drawLine( 0, height, width, 0 );
22    } // fim do método paintComponent
23 } // fim da classe DrawPanel
```

Importa as classes **java.awt.Graphics** e **javax.swing.JPanel**

A classe **DrawPanel** estende a classe **JPanel**

Declara o método **paintComponent** – chamado quando o componente é criado

Todo desenho é feito neste método

Recupera a largura e altura do **JPanel**

Exemplo

```
1  // Fig. 4.19: DrawPanel.java
2  // Desenha duas linhas que se cruzam em um pa
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class DrawPanel extends JPanel
7  {
8     // desenha um X a partir dos cantos do painel
9     public void paintComponent( Graphics g )
10    {
11       // chama paintComponent para assegurar que o p
12       super.paintComponent( g );
13
14       int width = getWidth(); // largura total
15       int height = getHeight(); // altura total
16
17       // desenha uma linha a partir do canto superior esquerdo até o
18       g.drawLine( 0, 0, width, height );
19
20       // desenha uma linha a partir do canto inferior esquerdo até o superior direito
21       g.drawLine( 0, height, width, 0 );
22    } // fim do método paintComponent
23 } // fim da classe DrawPanel
```

Importa as classes **java.awt.Graphics** e **javax.swing.JPanel**

A classe **DrawPanel** estende a classe **JPanel**

Declara o método **paintComponent – chamado quando o componente é criado**

Todo desenho é feito neste método

Recupera a largura e altura do **JPanel**

Desenha as duas linhas

Criando desenhos simples (Continuação)

- Classe JFrame do pacote javax.swing:
  - Permite ao programador criar uma janela.
  - Método setDefaultCloseOperation:
    - Passa JFrame.EXIT_ON_CLOSE como seu argumento para configurar a aplicação a terminar quando o usuário fecha a janela.
  - Método Add:
    - Anexa uma JPanel a Jframe.
  - Método setSize:
    - Configura a largura (primeiro argumento) e altura (segundo argumento) da Jframe.

Exemplo

```
1  // Fig. 4.20: DrawPanelTest.java
2  // Aplicativo para exibir uma DrawPanel.
3  import javax.swing.JFrame;
4
5  public class DrawPanelTest
6  {
7     public static void main( String args[] )
8     {
9        // cria um painel que contém nosso desenho
10       DrawPanel panel = new DrawPanel();
11
12       // cria um novo quadro para conter o painel
13       JFrame application = new JFrame();
14
15       // configura o frame para ser encerrado quando ele é fechado
16       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
17
18       application.add( panel ); // adiciona o painel ao frame
19       application.setSize( 250, 250 ); // configura o tamanho do frame
20       application.setVisible( true ); // torna o frame visível
21    } // fim de main
22 } // fim da classe DrawPanelTest
```

Importa a classe **JFrame** da classe **javax.swing**

Cria objetos **DrawPanel** e **JFrame**

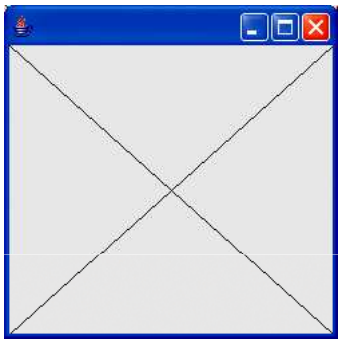
Configura a aplicação para terminar quando o usuário fecha a janela

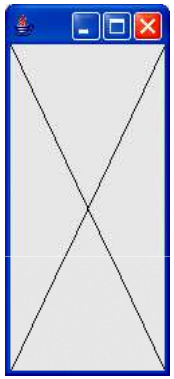
Adiciona a **DrawPanel** a **JFrame**

Configura o tamanho e exibe o **JFrame**

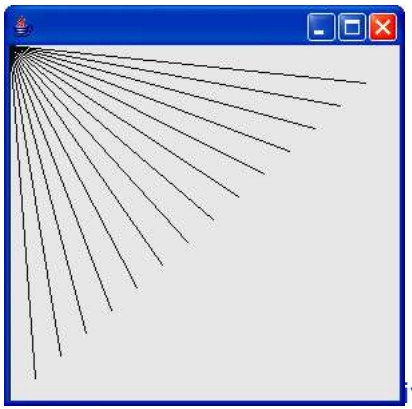
Exemplo: Linhas que se estendem a partir de um canto

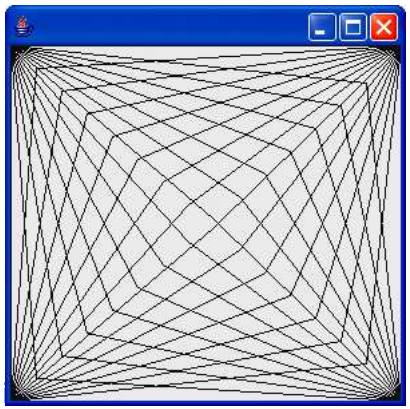
Resultado do programa anterior:





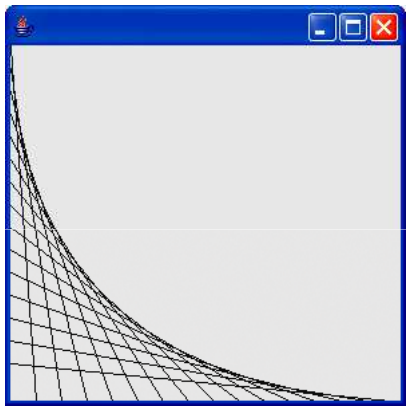
Altere o programa para obter os seguintes resultados:

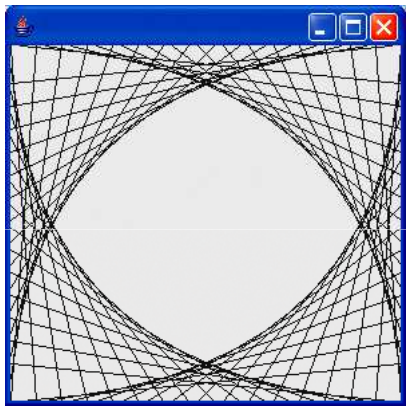




Desenho de linhas com loops e drawLine

Altere o programa para obter os seguintes resultados:





Criando desenhos simples (Continuação)

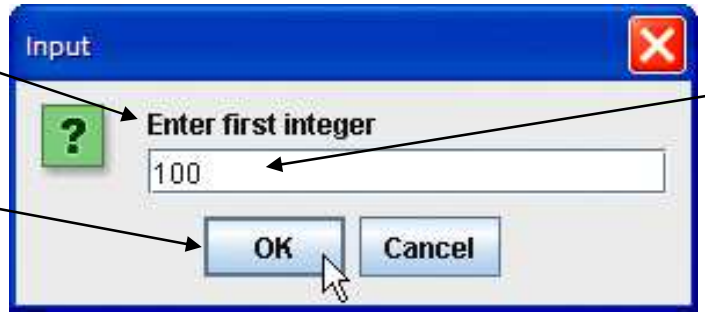
- A classe Jpanel:
  - Cada JPanel tem um método paintComponent:
    - paintComponent é chamado sempre que o sistema precisa exibir o Jpanel.
  - Métodos getWidth e getHeight:
    - Retornam a largura e a altura da JPanel, respectivamente.
  - Método drawLine:
    - Desenha uma linha a partir das coordenadas definidas pelos seus dois primeiros argumentos de acordo com as coordenadas definidas pelos seus dois segundos argumentos

JOptionPane

Diálogo de entrada exibido nas linhas 10–11

Prompt para o usuário

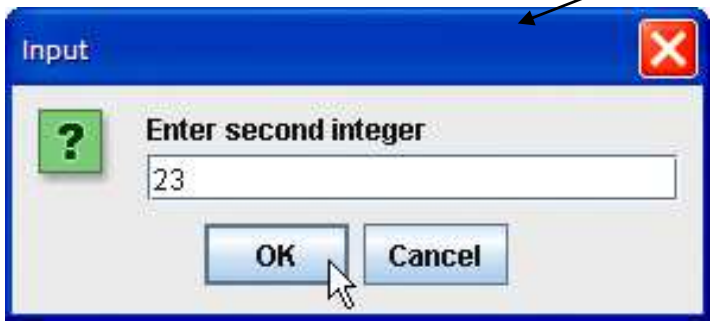
Quando o usuário clica em **OK**, **showInputDialog** retorna ao programa o **100** digitado pelo usuário como uma **String**. O programa deve converter a **String** em um **int**



Campo de texto em que o usuário digita um valor

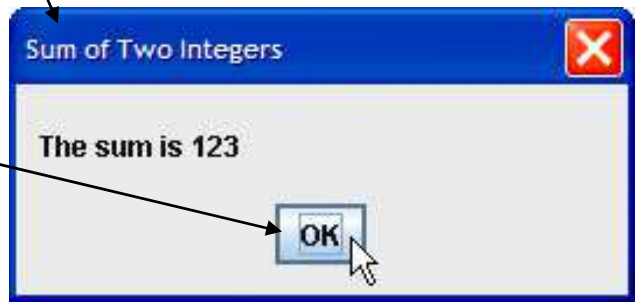
barra de título

Diálogo de entrada exibido nas linhas 12–13



Quando o usuário clique em **OK**, o diálogo de mensagem é fechado (é removido da tela)

Diálogo de entrada exibido nas linhas 22–23



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.

Exemplo

```
1  // Fig. 11.2: Addition.java
2  // Programa de adição que utiliza JOptionPane para entrada e saída.
3  import javax.swing.JOptionPane; // programa utiliza JOptionPane
4
5  public class Addition
6  {
7     public static void main( String args[] )
8     {
9        // obtém a entrada de usuário a partir dos diálogos de entrada JOptionPane
10       String firstNumber =
11          JOptionPane.showInputDialog( "Enter first integer" );
12       String secondNumber =
13           JOptionPane.showInputDialog( "Enter second integer" );
14
15       // converte String em valores int para utilização em um cálculo
16       int number1 = Integer.parseInt( firstNumber );
17       int number2 = Integer.parseInt( secondNumber );
18
19       int sum = number1 + number2; // adiciona números
20
21       // exibe o resultado em um diálogo de mensagem JOptionPane
22       JOptionPane.showMessageDialog( null, "The sum is " + sum,
23          "Sum of Two Integers", JOptionPane.PLAIN_MESSAGE );
24    } // fim do método  main
25 } // fim da classe Addition
```

Mostra o diálogo de entrada para receber o primeiro inteiro

Exemplo

```
1  // Fig. 11.2: Addition.java
2  // Programa de adição que utiliza JOptionPane para entrada e saída.
3  import javax.swing.JOptionPane; // programa utiliza JOptionPane
4
5  public class Addition
6  {
7     public static void main( String args[] )
8     {
9        // obtém a entrada de usuário a partir dos diálogos de entrada JOptionPane
10       String firstNumber =
11          JOptionPane.showInputDialog( "Enter first integer" );
12       String secondNumber =
13           JOptionPane.showInputDialog( "Enter second integer" );
14
15       // converte String em valores int para utilização em um cálculo
16       int number1 = Integer.parseInt( firstNumber );
17       int number2 = Integer.parseInt( secondNumber );
18
19       int sum = number1 + number2; // adiciona números
20
21       // exibe o resultado em um diálogo de mensagem JOptionPane
22       JOptionPane.showMessageDialog( null, "The sum is " + sum,
23          "Sum of Two Integers", JOptionPane.PLAIN_MESSAGE );
24    } // fim do método  main
25 } // fim da classe Addition
```

Mostra o diálogo de entrada para receber o primeiro inteiro

Mostra o diálogo de entrada para receber o segundo inteiro

Exemplo

```
1  // Fig. 11.2: Addition.java
2  // Programa de adição que utiliza JOptionPane para entrada e saída.
3  import javax.swing.JOptionPane; // programa utiliza JOptionPane
4
5  public class Addition
6  {
7     public static void main( String args[] )
8     {
9        // obtém a entrada de usuário a partir dos diálogos de entrada JOptionPane
10       String firstNumber =
11          JOptionPane.showInputDialog( "Enter first integer" );
12       String secondNumber =
13           JOptionPane.showInputDialog( "Enter second integer" );
14
15       // converte String em valores int para utilização em um cálculo
16       int number1 = Integer.parseInt( firstNumber );
17       int number2 = Integer.parseInt( secondNumber );
18
19       int sum = number1 + number2; // adiciona números
20
21       // exibe o resultado em um diálogo de mensagem JOptionPane
22       JOptionPane.showMessageDialog( null, "The sum is " + sum,
23          "Sum of Two Integers", JOptionPane.PLAIN_MESSAGE );
24    } // fim do método  main
25 } // fim da classe Addition
```

Mostra o diálogo de entrada para receber o primeiro inteiro

Mostra o diálogo de entrada para receber o segundo inteiro

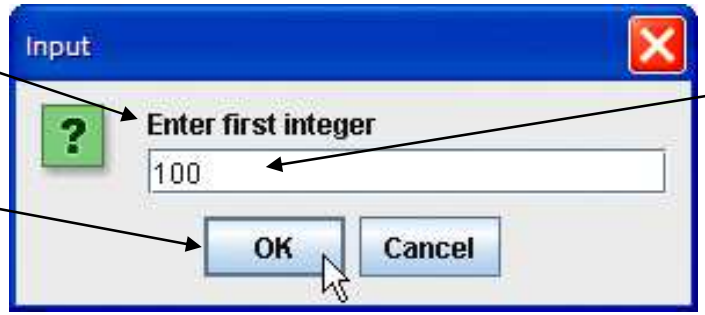
Mostra o diálogo de mensagem para gerar a saída da soma para o usuário

Exemplo

Diálogo de entrada exibido nas linhas 10–11

Prompt para o usuário

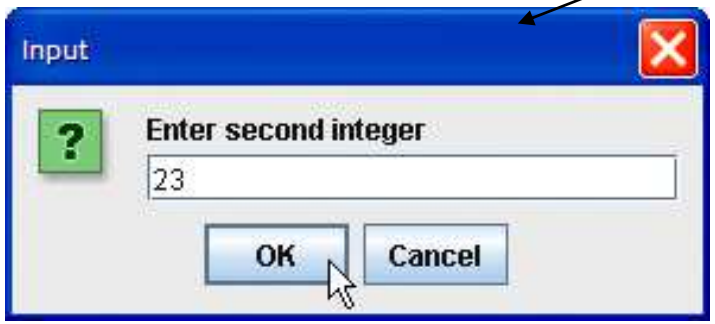
Quando o usuário clica em **OK**, **showInputDialog** retorna ao programa o **100** digitado pelo usuário como uma **String**. O programa deve converter a **String** em um **int**



Campo de texto em que o usuário digita um valor

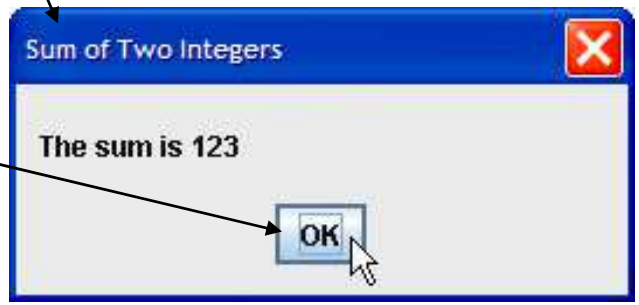
barra de título

Diálogo de entrada exibido nas linhas 12–13



Diálogo de entrada exibido nas linhas 22–23

Quando o usuário clique em **OK**, o diálogo de mensagem é fechado (é removido da tela)



Constantes **JOptionPane static** para diálogos de mensagem

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..

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.

Alguns componentes GUI básicos

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.

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

- 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.

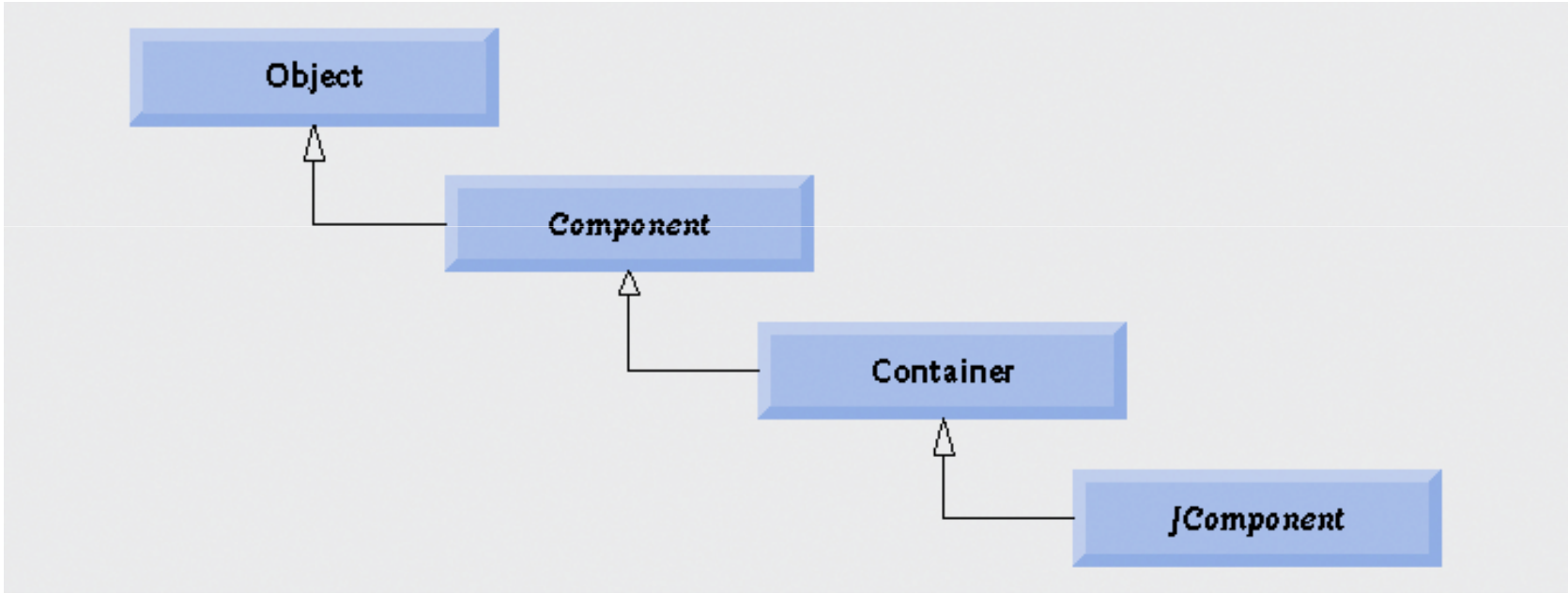
Componentes GUI leves *versus* pesados

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

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 Components.
- Classe JComponent (pacote javax.swing):
  - Subclasse de Container.
  - Superclasse de todos os componentes Swing leves.

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.
  - 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.

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

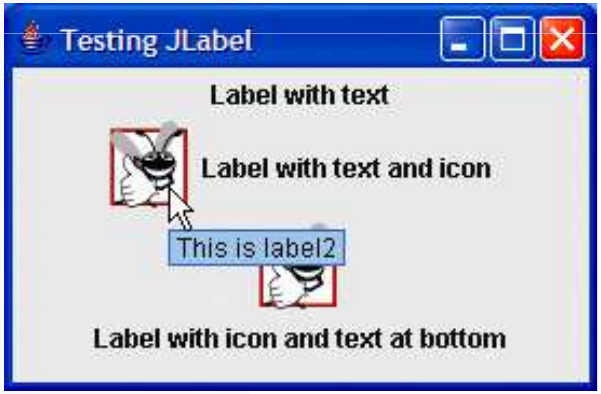
- Rótulo:
  - Instruções de texto ou informações que declaram o propósito de cada componente.
  - Criadas com a classe **JLabel**.

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.

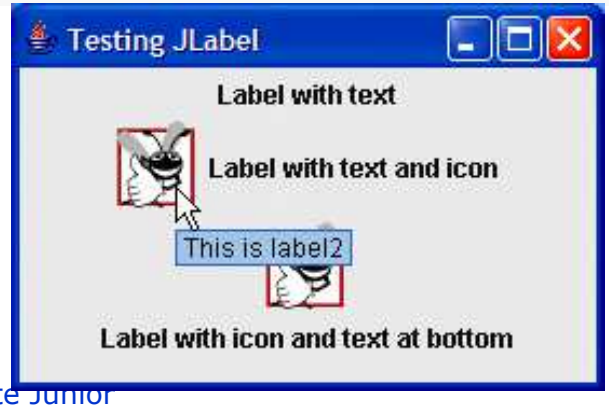
Resumo

```
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
```



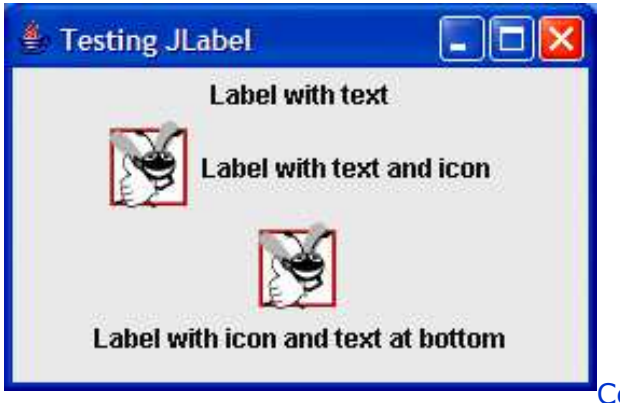
Resumo

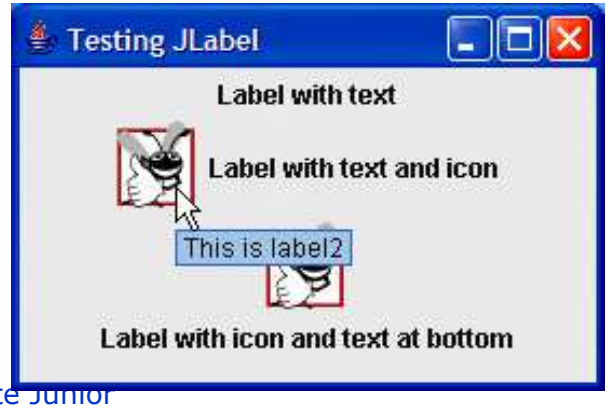
```
27       // construtor JLabel com string, Icon e argumentos de alinhamento
28       Icon bug = new ImageIcon( getClass().getResource(  "bug1.gif" ) );
29       label2 = new JLabel( "Label with text and icon", bug,
30          SwingConstants.LEFT );
31       label2.setToolTipText( "This is label2" );
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

```
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
```





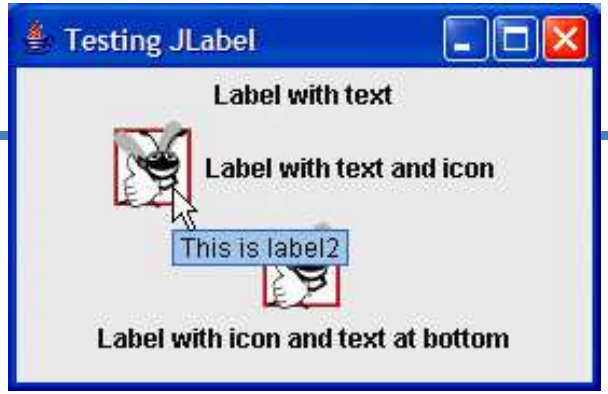
Criando e anexando label1

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

```
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
```

Cuidado!!!

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



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**.

```
label1.setHorizontalTextPosition(SwingConstants.CENTER);
Icon brinquedo = new ImageIcon("imagens/brinquedo.gif");
JLabel label2 = new JLabel("Centralizado - Imagem", brinquedo,
                                SwingConstants.CENTER);
```

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.

Alguns componentes GUI básicos

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.

Criando e exibindo uma janela LabelFrame

- Outros métodos JFrame:
  - setDefaultCloseOperation
    - Determina como a aplicação reage quando o usuário clica no botão de fechar.

```
//configurar fechamento automático
frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
```

  - setSize
    - Especifica a largura e altura da janela.
  - setVisible
    - Determina se a janela é exibida (true) ou não (false).

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.

Texto e uma introdução ao tratamento de eventos com classes aninhadas

- 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.

Exemplo

```
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
```

Cria um novo `JTextField`

Tipos de eventos

Componentes afetados	"Listener"	addxxxListener	Métodos do Listener
Button TextField MenuItem	ActionListener	`.addActionListener()`	`actionPerformed (ActionEvent e)`
List ComboBox	ItemListener	`.addItemListener()`	`stateChanged (ChangeEvent e)`
key on component	KeyListener	`.addKeyListener()`	`keyPressed(), keyReleased(), keyTyped()`
mouse on component	MouseListener	`.addMouseListener()`	`mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased()`
mouse on component	MouseMotionListener	`.addMouseMotionListener()`	`mouseMoved(), mouseDragged()`
Frame	WindowListener	`.addWindowListener()`	`windowClosing(WindowEvent e), ...`

Exemplo

```
28        // constrói campo de texto com texto padrão
29        textField2 = new JTextField( "Enter text here" );
30        add( textField2 ); // adiciona textField2 a JFrame
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 JFrame
36
37        // constrói passwordfield com o texto padrão
38        passwordField = new JPasswordField( "Hidden text" );
39        add( passwordField ); // adiciona passwordField a JFrame
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
```

Cria um novo `JTextField`

Exemplo

```
28        // constrói campo de texto com texto padrão
29        textField2 = new JTextField( "Enter text here" );
30        add( textField2 ); // adiciona textField2 a JFrame
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 JFrame
36
37        // constrói passwordfield com o texto padrão
38        passwordField = new JPasswordField( "Hidden text" );
39        add( passwordField ); // adiciona passwordField a JFrame
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
```

Cria um novo `JTextField`

Cria um novo `JTextField` não editável

Exemplo

```
28        // constrói campo de texto com texto padrão
29        textField2 = new JTextField( "Enter text here" );
30        add( textField2 ); // adiciona textField2 a JFrame
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 JFrame
36
37        // constrói passwordfield com o texto padrão
38        passwordField = new JPasswordField( "Hidden text" );
39        add( passwordField ); // adiciona passwordField a JFrame
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
```

Cria um novo `JTextField`

Cria um novo `JTextField` não editável

Cria um novo `JPasswordField`

Exemplo

```
28        // constrói campo de texto com texto padrão
29        textField2 = new JTextField( "Enter text here" );
30        add( textField2 ); // adiciona textField2 a JFrame
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 JFrame
36
37        // constrói passwordfield com o texto padrão
38        passwordField = new JPasswordField( "Hidden text" );
39        add( passwordField ); // adiciona passwordField a JFrame
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
```

Cria um novo `JTextField`

Cria um novo `JTextField` não editável

Cria um novo `JPasswordField`

Criar um handler de evento

Exemplo

```
28        // constrói campo de texto com texto padrão
29        textField2 = new JTextField( "Enter text here" );
30        add( textField2 ); // adiciona textField2 a JFrame
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 JFrame
36
37        // constrói passwordfield com o texto padrão
38        passwordField = new JPasswordField( "Hidden text" );
39        add( passwordField ); // adiciona passwordField a JFrame
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
```

Cria um novo `JTextField`

Cria um novo `JTextField` não editável

Cria um novo `JPasswordField`

Criar um handler de evento

Registra um handler de evento

Exemplo

```
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 exibida
56
```

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

Declara o método `actionPerformed`

Exemplo

```
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 textField2
63          else if ( event.getSource() == textField2 )
64             string = String.format( "textField2: %s",
65                event.getActionCommand() );
66
67          // usuário pressionou Enter no JTextField textField3
68          else if ( event.getSource() == textField3 )
69             string = String.format( "textField3: %s",
70                event.getActionCommand() );
71
72          // usuário pressionou Enter no JTextField passwordField
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

Exemplo

```
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 textFi
63          else if ( event.getSource() == textField2 )
64             string = String.format( "textField2: %s",
65                event.getActionCommand() );
66
67          // usuário pressionou Enter no JTextField textField3
68          else if ( event.getSource() == textField3 )
69             string = String.format( "textField3: %s",
70                event.getActionCommand() );
71
72          // usuário pressionou Enter no JTextField passwordField
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

Exemplo

```
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 textFi
63          else if ( event.getSource() == textField2 )
64             string = String.format( "textField2: %s",
65                event.getActionCommand() );
66
67          // usuário pressionou Enter no JTextField textField3
68          else if ( event.getSource() == textField3 )
69             string = String.format( "textField3: %s",
70                event.getActionCommand() );
71
72          // usuário pressionou Enter no JTextField passwordField
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

Exemplo

```
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 textFi
63          else if ( event.getSource() == textField2 )
64             string = String.format( "textField2: %s",
65                event.getActionCommand() );
66
67          // usuário pressionou Enter no JTextField textFi
68          else if ( event.getSource() == textField3 )
69             string = String.format( "textField3: %s",
70                event.getActionCommand() );
71
72          // usuário pressionou Enter no JTextField passwordField
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

Obtém texto a partir do campo de texto

Exemplo

```
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 textFi
63          else if ( event.getSource() == textField2 )
64             string = String.format( "textField2: %s",
65                event.getActionCommand() );
66
67          // usuário pressionou Enter no JTextField textFi
68          else if ( event.getSource() == textField3 )
69             string = String.format( "textField3: %s",
70                event.getActionCommand() );
71
72          // usuário pressionou Enter no JTextField passwordField
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

Obtém texto a partir do campo de texto

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

Exemplo

```
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 textFi
63          else if ( event.getSource() == textField2 )
64             string = String.format( "textField2: %s",
65                event.getActionCommand() );
66
67          // usuário pressionou Enter no JTextField textFi
68          else if ( event.getSource() == textField3 )
69             string = String.format( "textField3: %s",
70                event.getActionCommand() );
71
72          // usuário pressionou Enter no JTextField passwo
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

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

Exemplo

```
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 textFi
63          else if ( event.getSource() == textField2 )
64             string = String.format( "textField2: %s",
65                event.getActionCommand() );
66
67          // usuário pressionou Enter no JTextField textFi
68          else if ( event.getSource() == textField3 )
69             string = String.format( "textField3: %s",
70                event.getActionCommand() );
71
72          // usuário pressionou Enter no JTextField passwo
73          else if ( event.getSource() == passwordField )
74             string = String.format( "passwordField:
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

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

Exemplo

```
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 textFi
63          else if ( event.getSource() == textField2 )
64             string = String.format( "textField2: %s",
65                event.getActionCommand() );
66
67          // usuário pressionou Enter no JTextField textFi
68          else if ( event.getSource() == textField3 )
69             string = String.format( "textField3: %s",
70                event.getActionCommand() );
71
72          // usuário pressionou Enter no JTextField passwo
73          else if ( event.getSource() == passwordField )
74             string = String.format( "passwordField:
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

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

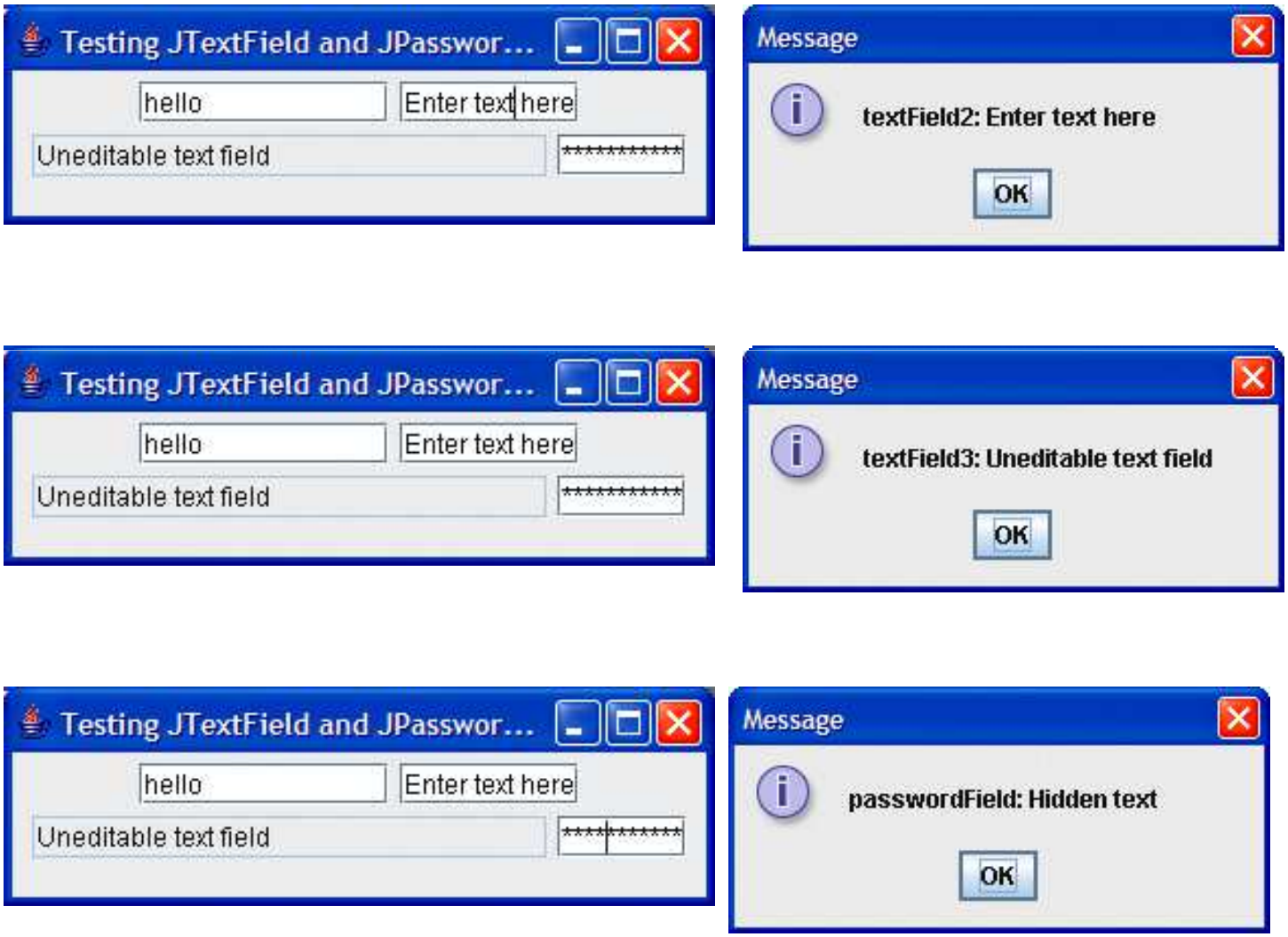
Testa se a origem do evento é o campo de senha

Obtém senha a partir do campo de senha

Exemplo

```
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
```

Exemplo


Testing JTextField and JPasswor...
hello
Enter text here
Uneditable text field
****
Message
textField2: Enter text here
OK
Testing JTextField and JPasswor...
hello
Enter text here
Uneditable text field
****
Message
textField3: Uneditable text field
OK
Testing JTextField and JPasswor...
hello
Enter text here
Uneditable text field
****
Message
passwordField: Hidden text
OK

Configurando o tratamento de evento de um componente GUI

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

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.
  - Frequentemente utilizadas para tratamento de eventos.

Classe aninhada para implementar um handler de evento

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

Tipos de eventos

Componentes afetados	"Listener"	addxxxListener	Métodos do Listener
Button TextField MenuItem	ActionListener	`.addActionListener()`	`actionPerformed (ActionEvent e)`
List ComboBox	ItemListener	`.addItemListener()`	`stateChanged (ChangeEvent e)`
key on component	KeyListener	`.addKeyListener()`	`keyPressed(), keyReleased(), keyTyped()`
mouse on component	MouseListener	`.addMouseListener()`	`mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased()`
mouse on component	MouseMotionListener	`.addMouseMotionListener()`	`mouseMoved(), mouseDragged()`
Frame	WindowListener	`.addWindowListener()`	`windowClosing(WindowEvent e), ...`

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.

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.

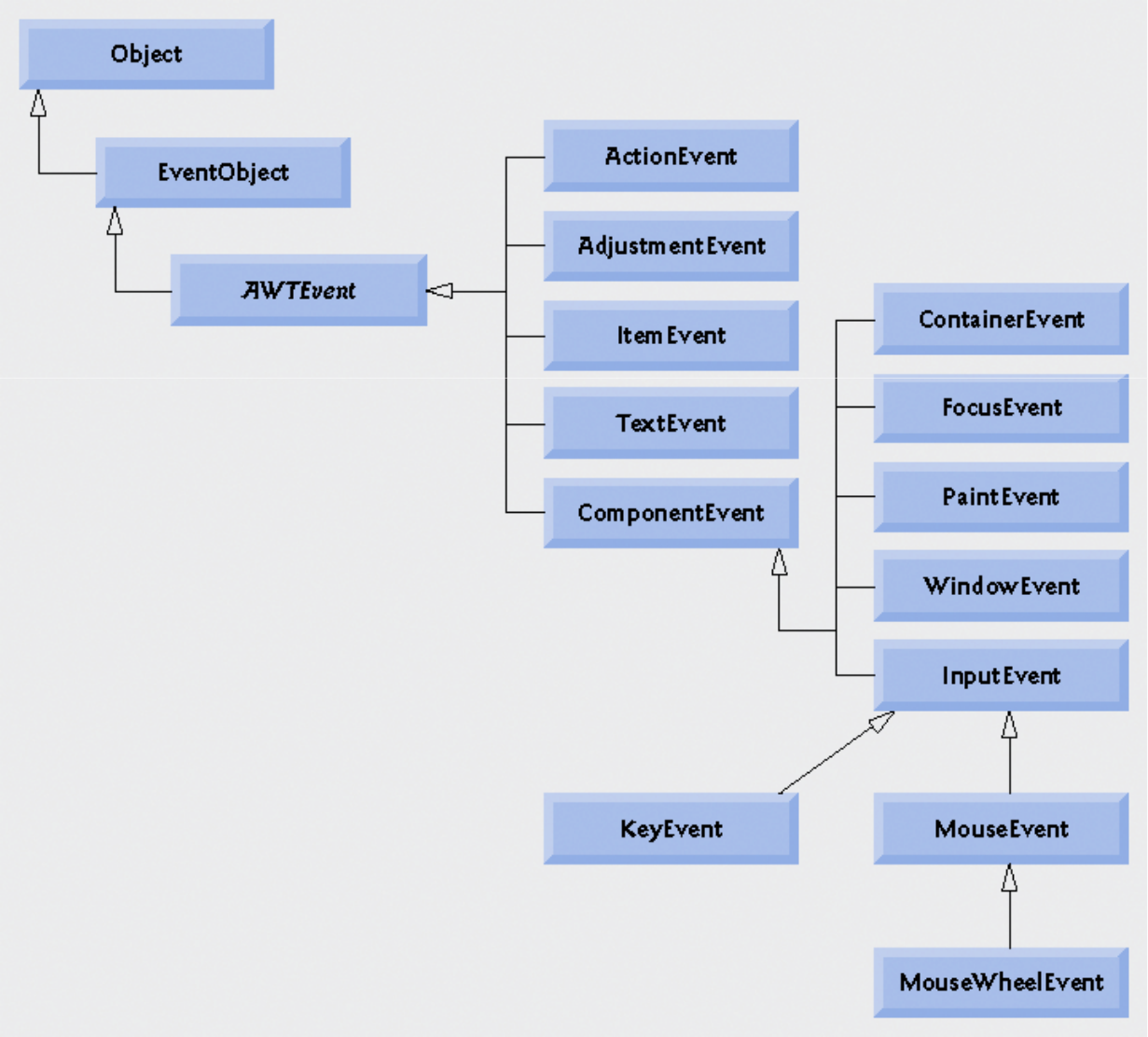
Tipos comuns de eventos GUI e interfaces ouvintes

- 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.

Tipos comuns de eventos GUI e interfaces ouvintes

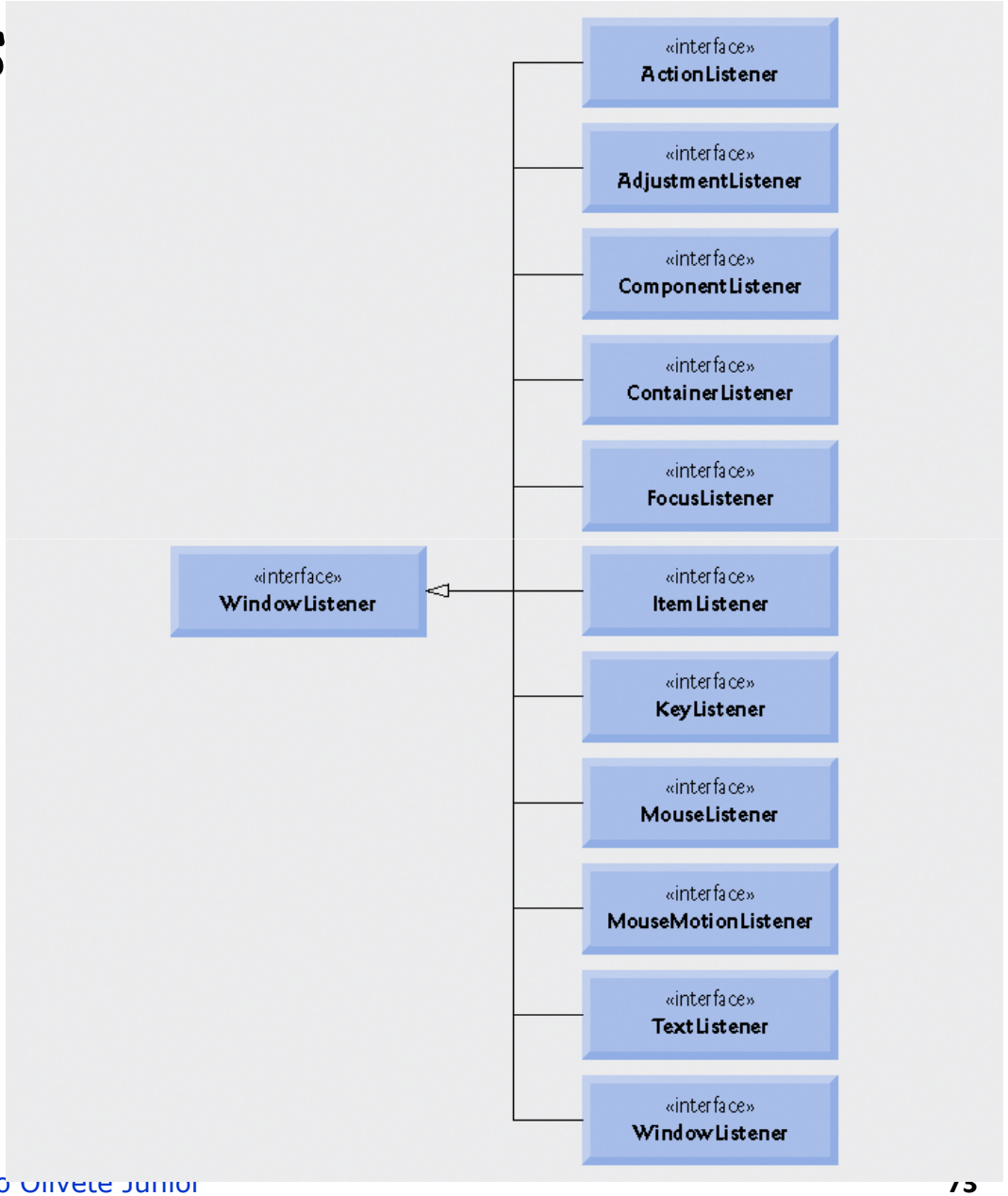
- Modelo de evento de delegação:
  - A origem do evento é o componente com o qual o usuário interage.
  - O objeto do evento é criado e contém as informações sobre o evento que aconteceu.
  - O ouvinte de evento é notificado quando um evento acontece.

Algumas classes de evento do pacote java.awt.event.



Interfaces ouvintes

Algumas interfaces ouvintes de eventos comuns do pacote java.awt.event.



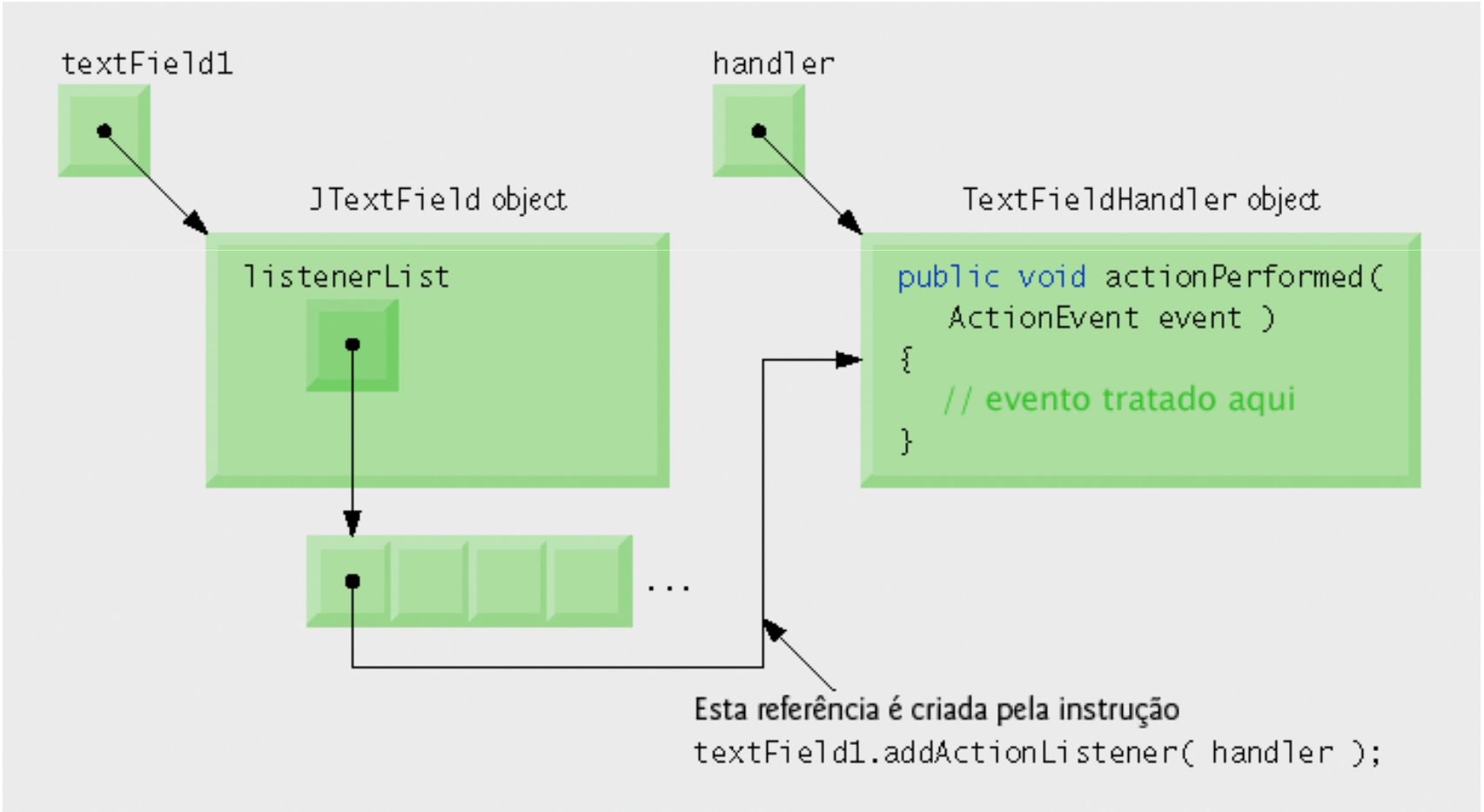
O tratamento de evento

- 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.

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.

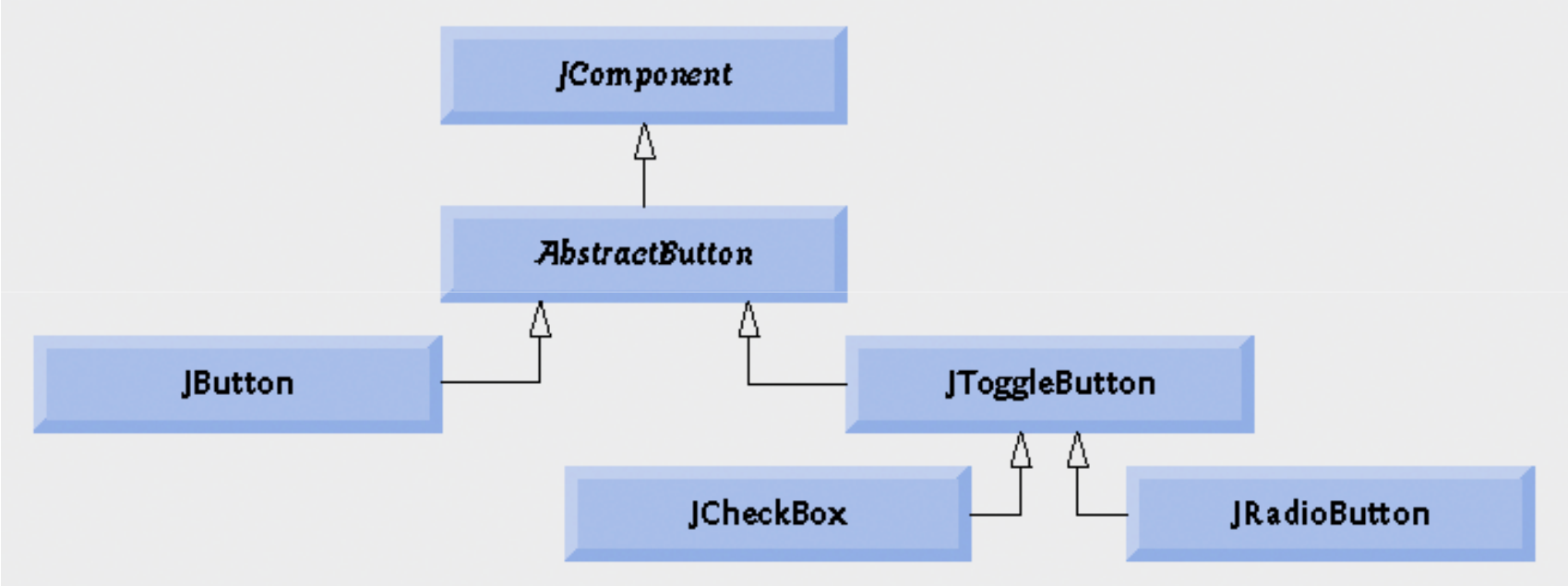
JButton

- 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.

JButton

- 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.

Hierarquia do botão Swing



```java
// Fig. 11.15: ButtonFrame.java
// Criando JButtons.
import java.awt.FlowLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;

public class ButtonFrame extends JFrame
{
   private JButton plainJButton; // botão apenas com texto
   private JButton fancyJButton; // botão com ícones

   // ButtonFrame adiciona JButtons ao JFrame
   public ButtonFrame()
   {
      super( "Testing Buttons" );
      setLayout( new FlowLayout() ); // configura o layout do frame

      plainJButton = new JButton( "Plain Button" ); // botão com texto
      add( plainJButton ); // adiciona plainJButton ao JFrame
```

```
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
```

Declara duas variáveis de instância `JButton`

```
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
```

Declara duas variáveis de instância `JButton`

Cria um novo `JButton`

Cria dois ImageIcon

Cria um novo JButton

```
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 ); // configura imagem
29        fancyJButton.setRolloverIcon( bug2 ); // configura imagem de rollover
30        add( fancyJButton ); // adiciona fancyJButton ao JFrame
```

Configura o ícone de rollover para JButton

Aparece quando o mouse é posicionado sobre um botão.

```
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 botão
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
```

Cria um handler para botões

```
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 d
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
```

Cria um handler para botões

Registra um handler de evento

```
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 d
39    private class ButtonHandler implements ActionListener
40    {
41       // trata evento de botão
42       public void actionPerformed( ActionEvent e
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
```

Cria um handler para botões

Registra um handler de evento

A classe interna implementa ActionListener

```
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 d
39    private class ButtonHandler implements ActionListener
40    {
41       // trata evento de botão
42       public void actionPerformed( ActionEvent e
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 priv
48 } // fim da classe ButtonFrame
```

Cria um handler para botões

Registra um handler de evento

A classe interna implementa `ActionListener`

Acessa a instância da classe externa utilizando essa referência

```
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 d
39    private class ButtonHandler implements ActionListener
40    {
41       // trata evento de botão
42       public void actionPerformed( ActionEvent e
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 priv
48 } // fim da classe ButtonFrame
```

Cria um handler para botões

Registra um handler de evento

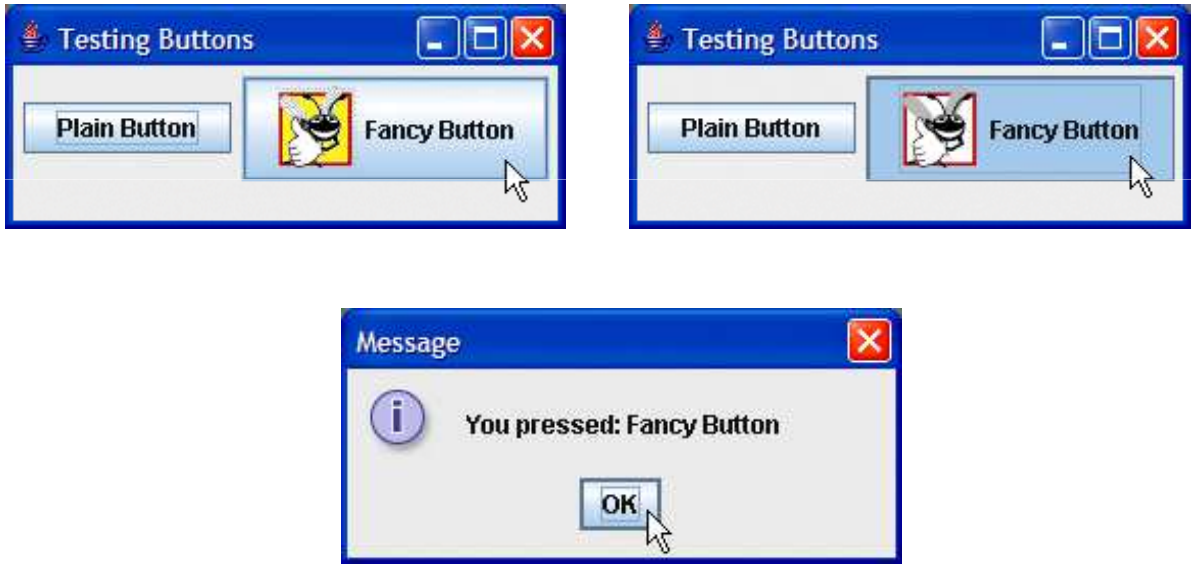
A classe interna implementa `ActionListener`

Acessa a instância da classe externa utilizando essa referência

Obtém o texto do `JButton` pressionado

```
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


Testing Buttons
Plain Button
Fancy Button
Testing Buttons
Plain Button
Fancy Button
Message
You pressed: Fancy Button
OK

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.

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.

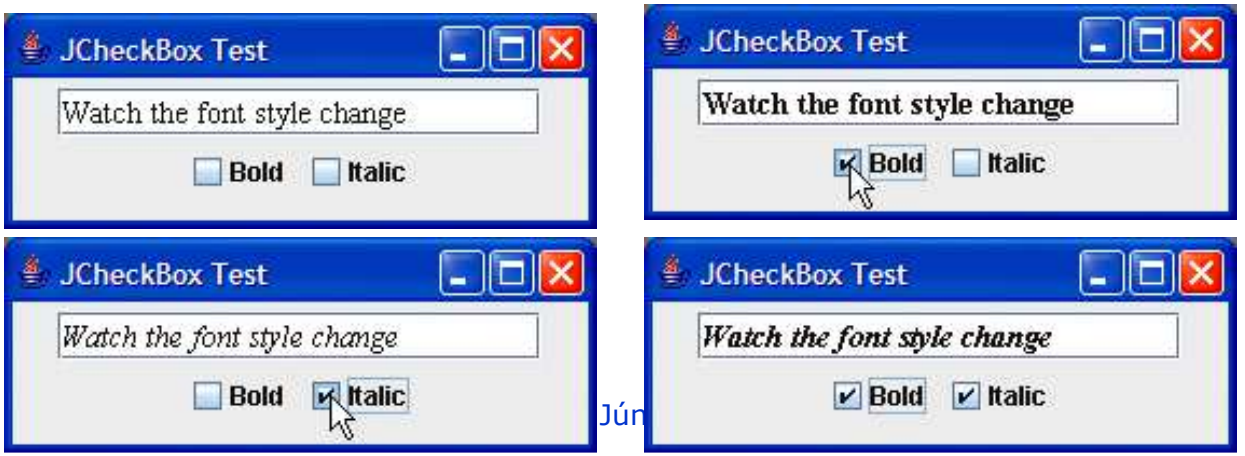
JCheckBox

- JCheckBox:
  - Contém um rótulo de caixa de seleção que aparece à direita da caixa de seleção por padrão.
  - Gera um ItemEvent quando é clicado.
    - ItemEvents 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).

Exemplo

```
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
```

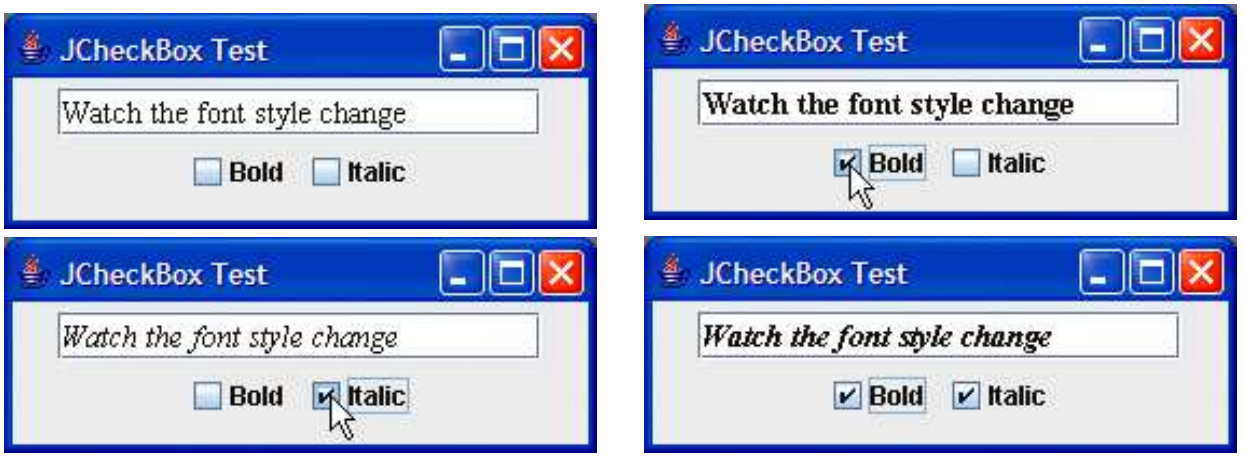
Declara duas variáveis de instância JCheckBox



Exemplo

```
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
```

Configura a origem do campo de texto



Exemplo

```
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 ne
31       add( italicJCheckBox ); // adiciona caixa de seleção para
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 ItemListener
40    private class CheckBoxHandler implements ItemListener
41    {
42       private int valBold = Font.PLAIN; // controla o estilo de fonte negrito
43       private int valItalic = Font.PLAIN; // controla o estilo de fonte itálico
44
45       // responde aos eventos de caixa de seleção
46       public void itemStateChanged( ItemEvent event )
47       {
48          // processa aos eventos de caixa de seleção de negrito
49          if ( event.getSource() == boldJCheckBox )
50             valBold =
51                boldJCheckBox.isSelected() ? Font.BOLD : Font.PLAIN;
52
```

Cria dois JCheckBoxes

Exemplo

```
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 ne
31       add( italicJCheckBox ); // adiciona caixa de seleção para
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 ItemListener
40    private class CheckBoxHandler implements ItemListener
41    {
42       private int valBold = Font.PLAIN; // controla o estilo de fonte negrito
43       private int valItalic = Font.PLAIN; // controla o estilo de fonte itálico
44
45       // responde aos eventos de caixa de seleção
46       public void itemStateChanged( ItemEvent event )
47       {
48          // processa aos eventos de caixa de seleção de negrito
49          if ( event.getSource() == boldJCheckBox )
50             valBold =
51                boldJCheckBox.isSelected() ? Font.BOLD : Font.PLAIN;
52
```

Cria dois JCheckBoxes

Cria um handler de evento

Exemplo

```
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 ne
31       add( italicJCheckBox ); // adiciona caixa de seleção para
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 Iten
40    private class CheckBoxHandler implements ItemListener
41    {
42       private int valBold = Font.PLAIN; // controla o estilo de fonte negrito
43       private int valItalic = Font.PLAIN; // controla o estilo de fonte itálico
44
45       // responde aos eventos de caixa de seleção
46       public void itemStateChanged( ItemEvent event )
47       {
48          // processa aos eventos de caixa de seleção de negrito
49          if ( event.getSource() == boldJCheckBox )
50             valBold =
51                boldJCheckBox.isSelected() ? Font.BOLD : Font.PLAIN;
52
```

Cria dois JCheckBoxes

Cria um handler de evento

Registra um handler de evento com JCheckBoxes

Exemplo

```
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 ne
31       add( italicJCheckBox ); // adiciona caixa de seleção para
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 Iten
40    private class CheckBoxHandler implements ItemListener
41    {
42       private int valBold = Font.PLAIN; // controla o estilo
43       private int valItalic = Font.PLAIN; // controla o est
44
45       // responde aos eventos de caixa de seleção
46       public void itemStateChanged( ItemEvent event )
47       {
48          // processa aos eventos de caixa de seleção de negrito
49          if ( event.getSource() == boldJCheckBox )
50             valBold =
51                boldJCheckBox.isSelected() ? Font.BOLD : Font.PLAIN;
52
```

Cria dois JCheckBoxes

Cria um handler de evento

Registra um handler de evento com JCheckBoxes

A classe interna implementa ItemListener

Exemplo

```
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 ne
31       add( italicJCheckBox ); // adiciona caixa de seleção para
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 Iten
40    private class CheckBoxHandler implements ItemListener
41    {
42       private int valBold = Font.PLAIN; // controla o estilo
43       private int valItalic = Font.PLAIN; // controla o est
44
45       // responde aos eventos de caixa de seleção
46       public void itemStateChanged( ItemEvent event )
47       {
48          // processa aos eventos de caixa de seleção de negr
49          if ( event.getSource() == boldJCheckBox )
50             valBold =
51                boldJCheckBox.isSelected() ? Font.BOLD : Font.PLAIN;
52
```

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

Exemplo

```
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 ne
31       add( italicJCheckBox ); // adiciona caixa de seleção para
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 Iten
40    private class CheckBoxHandler implements ItemListener
41    {
42       private int valBold = Font.PLAIN; // controla o estilo
43       private int valItalic = Font.PLAIN; // controla o est
44
45       // responde aos eventos de caixa de seleção
46       public void itemStateChanged( ItemEvent event )
47       {
48          // processa aos eventos de caixa de seleção de negr
49          if ( event.getSource() == boldJCheckBox )
50             valBold =
51                boldJCheckBox.isSelected() ? Font.BOLD : Font.PLAIN;
52
```

Cria dois `JCheckBox`es

Cria um handler de evento

Registra um handler de evento com `JCheckBox`es

A classe interna implementa `ItemListener`

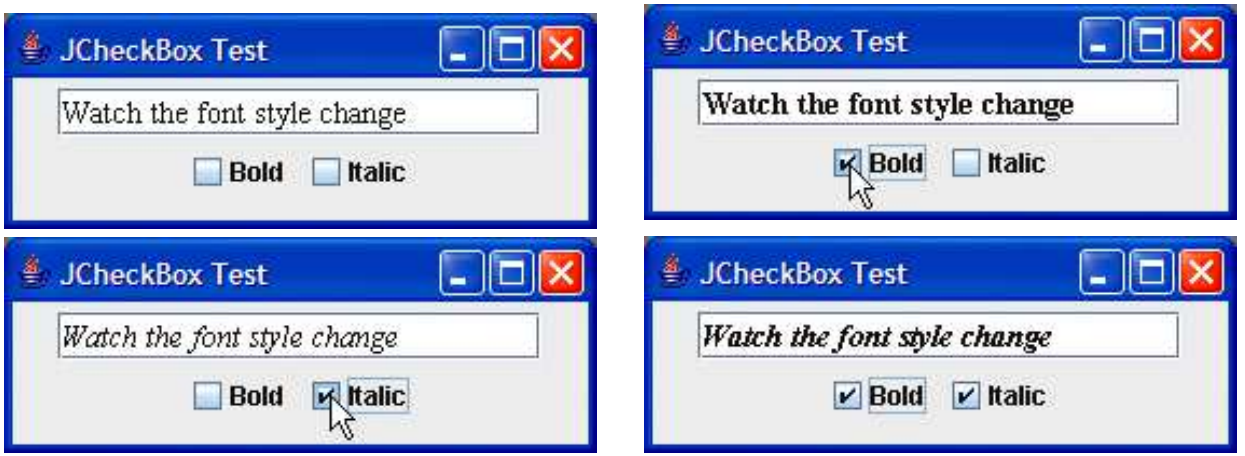
O método `itemStateChanged` é chamado quando uma `JCheckBox` é clicada

Testa se `JCheckBox` está selecionada

Exemplo

Testa a origem do evento

```
53          // processa eventos de caixa de seleção de italico
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
```

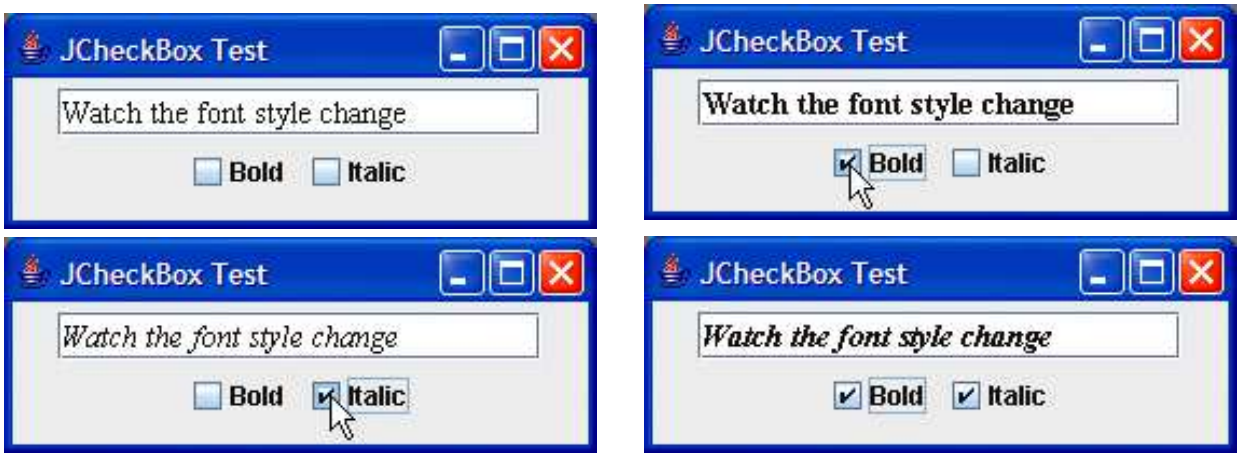


Exemplo

Testa a origem do evento

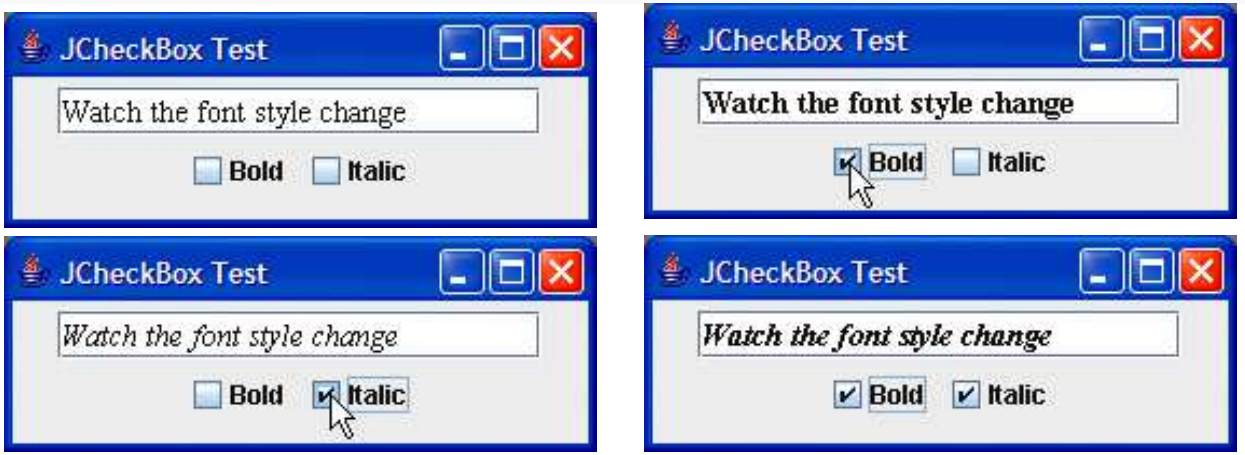
```
53          // processa eventos de caixa de seleção de itálico
54          if ( event.getSource() == italicJCheckBox )
55             valItalic =
56                italicJCheckBox.isSelected() ? Font.ITALIC : Font.PLAIN;
57
58          // configura fonte do campo de texto
59          textField.setFont(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

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



Exemplo

```
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
```



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.

Exemplo

```
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 alterações de fonte
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
```

Exemplo


```
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 a
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
```

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

Exemplo


```
25    // construtor RadioButtonFrame adiciona JRadioButtons ao
26    public RadioButtonFrame()
27    {
28       super( "RadioButton Test" );
29       setLayout( new FlowLayout() ); // configura layout do frame
30
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"
38       boldItalicJRadioButton = new JRadioButton( "Bol
39       add( plainJRadioButton ); // adiciona botão simples ao JFrame
40       add( boldJRadioButton ); // adiciona botão de negrito ao JFrame
41       add( italicJRadioButton ); // adiciona botão de itálico ao JFrame
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 simples ao grupo
47       radioGroup.add( boldJRadioButton ); // adiciona negrito ao grupo
48       radioGroup.add( italicJRadioButton ); // adiciona itálico ao grupo
49       radioGroup.add( boldItalicJRadioButton ); // adiciona negrito e itálico
50
```

Cria os quatro `JRadioButtons`

Exemplo


```
25    // construtor RadioButtonFrame adiciona JRadioButtons a
26    public RadioButtonFrame()
27    {
28       super( "RadioButton Test" );
29       setLayout( new FlowLayout() ); // configura layout do frame
30
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"
38       boldItalicJRadioButton = new JRadioButton( "Bol
39       add( plainJRadioButton ); // adiciona botão simples ao JFrame
40       add( boldJRadioButton ); // adiciona botão de negrito ao JFrame
41       add( italicJRadioButton ); // adiciona botão de itálico ao JFrame
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 ); // adicion
47       radioGroup.add( boldJRadioButton ); // adiciona
48       radioGroup.add( italicJRadioButton ); // adiciona itálico ao grupo
49       radioGroup.add( boldItalicJRadioButton ); // adiciona negrito e itálico
50
```

Cria os quatro `JRadioButtons`

Cria o `ButtonGroup`

Exemplo


```
25    // construtor RadioButtonFrame adiciona JRadioButtons a
26    public RadioButtonFrame()
27    {
28       super( "RadioButton Test" );
29       setLayout( new FlowLayout() ); // configura layout do frame
30
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"
38       boldItalicJRadioButton = new JRadioButton( "Bol
39       add( plainJRadioButton ); // adiciona botão simples ao JFrame
40       add( boldJRadioButton ); // adiciona botão de negrito ao JFrame
41       add( italicJRadioButton ); // adiciona botão de itálico ao JFrame
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 ); // adicion
47       radioGroup.add( boldJRadioButton ); // adiciona
48       radioGroup.add( italicJRadioButton ); // adicio
49       radioGroup.add( boldItalicJRadioButton ); // ad
50
```

Cria os quatro `JRadioButtons`

Cria o `ButtonGroup`

Adiciona cada `JRadioButton` ao `ButtonGroup`

Exemplo


```
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
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
```

Registra um handler de evento com cada `JRadioButton`

Exemplo


```
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
73
74       public RadioButtonHandler( Font f )
75       {
76          font = f; // configura a fonte desse listener
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
```

A classe interna do handler de evento implementa `ItemListener`

Exemplo

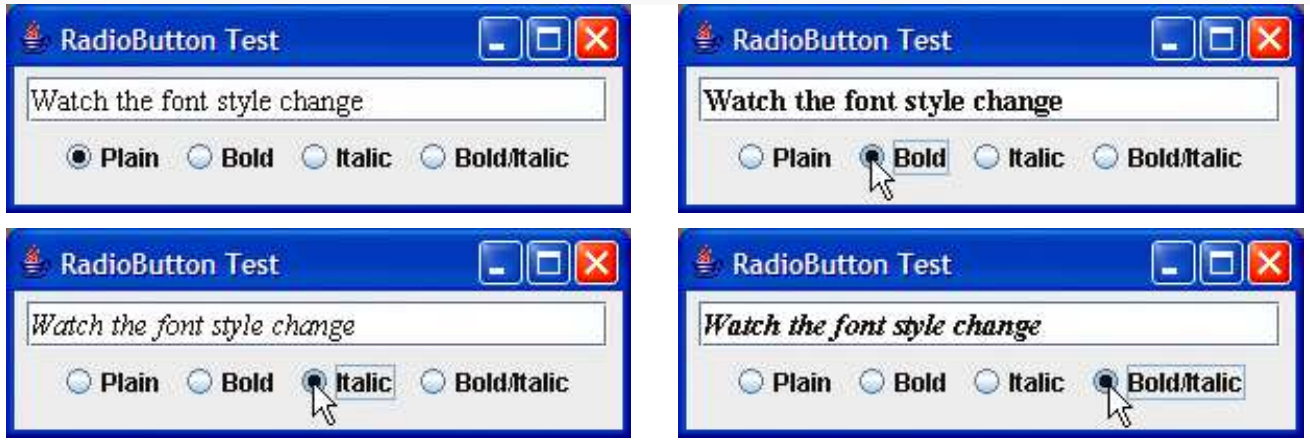

```
69    // classe interna private para tratar eventos de botão de opção
70    private class RadioButtonHandler implements ItemListener
71    {
72       private Font font; // fonte associada com
73
74       public RadioButtonHandler( Font f )
75       {
76          font = f; // configura a fonte desse li
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
```

A classe interna do handler de evento implementa `ItemListener`

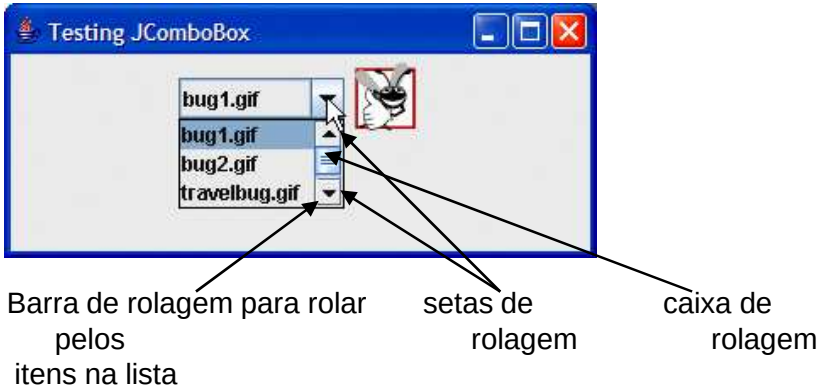
Quando o botão de opção é selecionado, a origem do campo de texto é configurada com o valor passado para o construtor

Exemplo

```
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
```



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.

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.

Exemplo

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

Exemplo

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

Declara a variável de instância JComboBox

Exemplo

Cria JComboBox e configura a contagem máxima de linhas

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

Exemplo


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

Cria `JComboBox` e configura a contagem máxima de linhas

Cria a classe interna anônima como o handler de evento

Exemplo


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

Exemplo


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

Exemplo


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

Cria `JComboBox` e configura a contagem máxima de linhas

Cria a classe interna anônima como o handler de evento

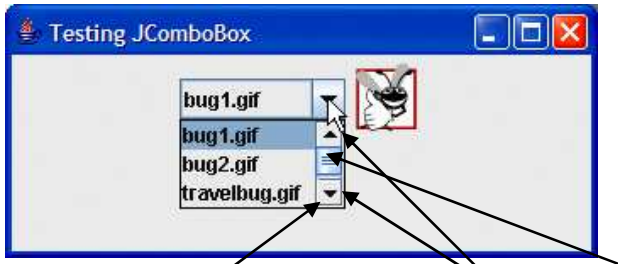
Declara o método `itemStateChanged`

Testa a alteração de estado da `JComboBox`

O método `getSelectedIndex` localiza o item selecionado

Exemplo

```
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
```



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.

Exemplo


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

Exemplo


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

Declara a variável de instância `JList`

Exemplo


```
25       colorJList = new JList( colorNames ); // cria com colorNames
26       colorJList.setVisibleRowCount( 5 ); // exibe cinco linhas de uma vez
27
28       // não permite múltiplas seleções
29       colorJList.setSelectionMode( ListSelectionModel.SINGLE_SELECTION );
30
31       // adiciona um JScrollPane que contém JList ao frame
32       add( new JScrollPane( colorJList ) );
33
34       colorJList.addListSelectionListener(
35          new ListSelectionListener() // classe interna anônima
36          {
37             // trata eventos de seleção de lista
38             public void valueChanged( ListSelectionEvent event )
39             {
40                getContentPane().setBackground(
41                   colors[ colorJList.getSelectedIndex() ] );
42             } // fim do método valueChanged
43          } // fim da classe interna anônima
44       ); // fim da chamada para addListSelectionListener
45    } // fim do construtor ListFrame
46 } // fim da classe ListFrame
```

Exemplo


```
25       colorJList = new JList( colorNames ); // cria com colorNames
26       colorJList.setVisibleRowCount( 5 ); // exibe cinco                vez
27
28       // não permite múltiplas seleções
29       colorJList.setSelectionMode( ListSelectionModel.SINGLE_SELECTION );
30
31       // adiciona um JScrollPane que contém JList ao frame
32       add( new JScrollPane( colorJList ) );
33
34       colorJList.addListSelectionListener(
35          new ListSelectionListener() // classe interna anônima
36          {
37             // trata eventos de seleção de lista
38             public void valueChanged( ListSelectionEvent event )
39             {
40                getContentPane().setBackground(
41                   colors[ colorJList.getSelectedIndex() ] );
42             } // fim do método valueChanged
43          } // fim da classe interna anônima
44       ); // fim da chamada para addListSelectionListener
45    } // fim do construtor ListFrame
46 } // fim da classe ListFrame
```

Cria `JList`

Exemplo


```
25       colorJList = new JList( colorNames ); // cria com colorNames
26       colorJList.setVisibleRowCount( 5 ); // exibe cinco                vez
27
28       // não permite múltiplas seleções
29       colorJList.setSelectionMode( ListSelectionModel.SINGLE_SELECTION );
30
31       // adiciona um JScrollPane que contém JList ao f
32       add( new JScrollPane( colorJList ) );
33
34       colorJList.addListSelectionListener(
35          new ListSelectionListener() // classe interna anônima
36          {
37             // trata eventos de seleção de lista
38             public void valueChanged( ListSelectionEvent event )
39             {
40                getContentPane().setBackground(
41                   colors[ colorJList.getSelectedIndex() ] );
42             } // fim do método valueChanged
43          } // fim da classe interna anônima
44       ); // fim da chamada para addListSelectionListener
45    } // fim do construtor ListFrame
46 } // fim da classe ListFrame
```

Cria `JList`

Configura o modo de seleção da `JList`

Exemplo


```
25       colorJList = new JList( colorNames ); // cria com colorNames
26       colorJList.setVisibleRowCount( 5 ); // exibe cinco [illegible] vez
27
28       // não permite múltiplas seleções
29       colorJList.setSelectionMode( ListSelectionModel.SINGLE_SELECTION );
30
31       // adiciona um JScrollPane que contém JList ao f
32       add( new JScrollPane( colorJList ) );
33
34       colorJList.addListSelectionListener(
35          new ListSelectionListener() // classe interna
36          {
37             // trata eventos de seleção de lista
38             public void valueChanged( ListSelectionEvent event )
39             {
40                getContentPane().setBackground(
41                   colors[ colorJList.getSelectedIndex() ] );
42             } // fim do método valueChanged
43          } // fim da classe interna anônima
44       ); // fim da chamada para addListSelectionListener
45    } // fim do construtor ListFrame
46 } // fim da classe ListFrame
```

Cria `JList`

Configura o modo de seleção da `JList`

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

Exemplo


```
25       colorJList = new JList( colorNames ); // cria com colorNames
26       colorJList.setVisibleRowCount( 5 ); // exibe cinco
27
28       // não permite múltiplas seleções
29       colorJList.setSelectionMode( ListSelectionModel.SINGLE_SELECTION );
30
31       // adiciona um JScrollPane que contém JList ao f
32       add( new JScrollPane( colorJList ) );
33
34       colorJList.addListSelectionListener(
35          new ListSelectionListener() // classe interna
36          {
37             // trata eventos de seleção de lista
38             public void valueChanged( ListSelectionEvent event )
39             {
40                getContentPane().setBackground(
41                   colors[ colorJList.getSelectedIndex() ] );
42             } // fim do método valueChanged
43          } // fim da classe interna anônima
44       ); // fim da chamada para addListSelectionListener
45    } // fim do construtor ListFrame
46 } // fim da classe ListFrame
```

Cria `JList`

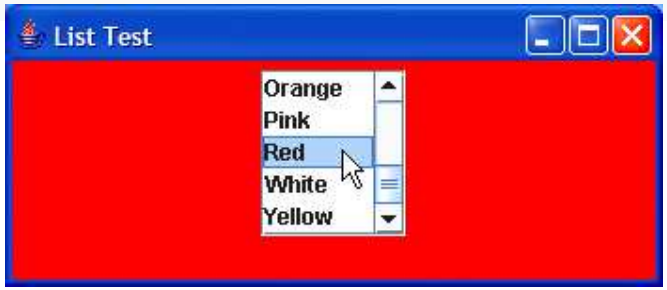
Configura o modo de seleção da `JList`

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

Obtém o índice do item selecionado

Exemplo

```
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
```



Listas 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.

Exemplo


```
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
```

Exemplo


```
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 ) ); // adiciona lista com scrollpane
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
```

Exemplo


```
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

Exemplo


```
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

Exemplo


```
47        add( copyJButton ); // adiciona botão de cópia ao JFrame
48
49        copyJList = new JList(); // cria lista p/ armazenar nomes de cor copiados
50        copyJList.setVisibleRowCount( 5 ); // mostra 5 linhas
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
```

Exemplo


```
47        add( copyJButton ); // adiciona botão de cópia ao JFrame
48
49        copyJList = new JList(); // cria lista p/ armaze
50        copyJList.setVisibleRowCount( 5 ); // mostra 5 l
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

Exemplo


```
47        add( copyJButton ); // adiciona botão de cópia ao JFrame
48
49        copyJList = new JList(); // cria lista p/ armaze
50        copyJList.setVisibleRowCount( 5 ); // mostra 5 l
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

Exemplo


```
47        add( copyJButton ); // adiciona botão de cópia ao JFrame
48
49        copyJList = new JList(); // cria lista p/ armaze
50        copyJList.setVisibleRowCount( 5 ); // mostra 5 l
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

Exemplo

```
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
```

Tratamento de evento de mouse

Eventos de mouse:

Cria um objeto MouseEvent.

Tratado por MouseListeners e MouseMotionListeners.

MouseInputListener combina as duas interfaces.

Métodos de interface MouseListener e MouseMotionListener

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.

Métodos de interface MouseListener e MouseMotionListener

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.

Resumo


```
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;
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
17    // construtor MouseTrackerFrame configura GUI e
18    // registra handlers de evento de mouse
19    public MouseTrackerFrame()
20    {
21       super( "Demonstrating Mouse Events" );
23       mousePanel = new JPanel(); // cria painel
24       mousePanel.setBackground( Color.WHITE ); // configura cor de fundo
25       add( mousePanel, BorderLayout.CENTER ); // adiciona painel ao JFrame
26
27       statusBar = new JLabel( "Mouse outside JPanel" );
28       add( statusBar, BorderLayout.SOUTH ); // adiciona rótulo ao JFrame
```

Resumo


```
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;
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 inf
17    // construtor MouseTrackerFrame configura GUI e
18    // registra handlers de evento de mouse
19    public MouseTrackerFrame()
20    {
21       super( "Demonstrating Mouse Events" );
23       mousePanel = new JPanel(); // cria painel
24       mousePanel.setBackground( Color.WHITE ); // configura cor de fundo
25       add( mousePanel, BorderLayout.CENTER ); // adiciona painel ao JFrame
26
27       statusBar = new JLabel( "Mouse outside JPanel" );
28       add( statusBar, BorderLayout.SOUTH ); // adiciona rótulo ao JFrame
```

Cria `JPanel` para capturar eventos de mouse

Resumo


```
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;
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 inf
17    // construtor MouseTrackerFrame configura GUI e
18    // registra handlers de evento de mouse
19    public MouseTrackerFrame()
20    {
21       super( "Demonstrating Mouse Events" );
23       mousePanel = new JPanel(); // cria painel
24       mousePanel.setBackground( Color.WHITE ); // configura cor de fundo
25       add( mousePanel, BorderLayout.CENTER ); // adiciona painel ao JFrame
26
27       statusBar = new JLabel( "Mouse outside JPanel" );
28       add( statusBar, BorderLayout.SOUTH ); // adiciona rótulo ao JFrame
```

Cria `JPanel` para capturar eventos de mouse

Configura o fundo como branco

Resumo


```
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;
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 inf
17    // construtor MouseTrackerFrame configura GUI e
18    // registra handlers de evento de mouse
19    public MouseTrackerFrame()
20    {
21       super( "Demonstrating Mouse Events" );
23       mousePanel = new JPanel(); // cria painel
24       mousePanel.setBackground( Color.WHITE ); // confi
25       add( mousePanel, BorderLayout.CENTER ); // adicic
26
27       statusBar = new JLabel( "Mouse outside JPanel" );
28       add( statusBar, BorderLayout.SOUTH ); // adiciona rótulo ao JFrame
```

Cria `JPanel` para capturar eventos de mouse

Configura o fundo como branco

Cria `JLabel` e o adiciona à aplicação

Resumo

```
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
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
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
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
```

Resumo


```
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
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
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
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

Resumo


```
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
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
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
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

Resumo

```
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
36    private class MouseHandler implements MouseListener,
37       MouseMotionListener
38    {
39       // handlers de evento MouseListener
40       // trata evento quando o mouse é liberado logo depoi
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
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
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

Resumo

```
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
36    private class MouseHandler implements MouseListener,
37       MouseMotionListener
38    {
39       // handlers de evento MouseListener
40       // trata evento quando o mouse é liberado logo depoi
41       public void mouseClicked( MouseEvent event )
42       {
43          statusBar.setText( String.format( "Clicked at [%d
44             event.getX(), event.getY() ) );
45       } // fim do método mouseClicked
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
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

Declara o método `mouseClicked`

Resumo

```
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
36    private class MouseHandler implements MouseListener,
37       MouseMotionListener
38    {
39       // handlers de evento MouseListener
40       // trata evento quando o mouse é liberado logo depoi
41       public void mouseClicked( MouseEvent event )
42       {
43          statusBar.setText( String.format( "Clicked at [%d
44             event.getX(), event.getY() ) );
45       } // fim do método mouseClicked
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
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

Declara o método `mouseClicked`

Determina a localização do clique de mouse

Resumo

```
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
36    private class MouseHandler implements MouseListener,
37       MouseMotionListener
38    {
39       // handlers de evento MouseListener
40       // trata evento quando o mouse é liberado logo depoi
41       public void mouseClicked( MouseEvent event )
42       {
43          statusBar.setText( String.format( "Clicked at [%d
44             event.getX(), event.getY() ) );
45       } // fim do método mouseClicked
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
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

Declara o método `mouseClicked`

Determina a localização do clique de mouse

Declara o método `mousePressed`

Resumo

```
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
36    private class MouseHandler implements MouseListener,
37       MouseMotionListener
38    {
39       // handlers de evento MouseListener
40       // trata evento quando o mouse é liberado logo depoi
41       public void mouseClicked( MouseEvent event )
42       {
43          statusBar.setText( String.format( "Clicked at [%d
44             event.getX(), event.getY() ) );
45       } // fim do método mouseClicked
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
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

Declara o método `mouseClicked`

Determina a localização do clique de mouse

Declara o método `mousePressed`

Declara o método `mouseReleased`

Resumo

```
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
```

Resumo

```
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

Resumo

```
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`

Resumo

```
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`

Resumo

```
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

```
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
```

Resumo

```
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`

Resumo

```
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
```

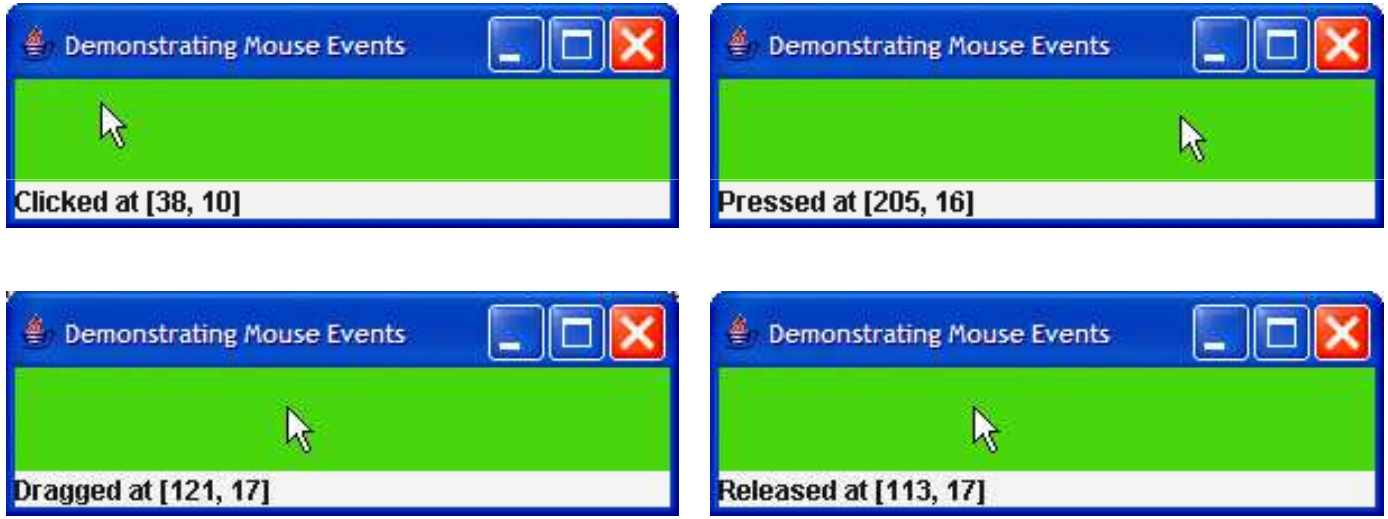
Declara o método mouseDragged

Declara o método mouseMoved

Resumo

```
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


Demonstrating Mouse Events
Clicked at [38, 10]
Demonstrating Mouse Events
Pressed at [205, 16]
Demonstrating Mouse Events
Dragged at [121, 17]
Demonstrating Mouse Events
Released at [113, 17]

Classes adaptadoras

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

Herdando MouseAdapter

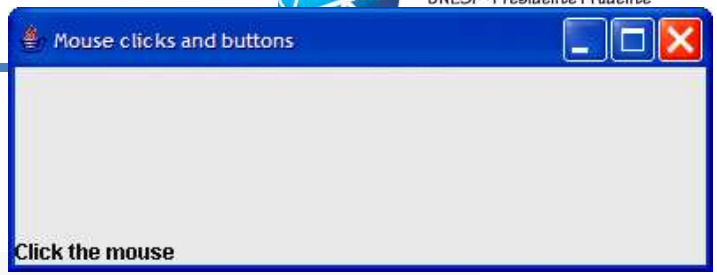
- MouseAdapter:
  - Classe adaptadora para as interfaces MouseListener e MouseMotionListener.
  - Estender a classe permite sobrescrever somente os métodos que você deseja utilizar.

Classes Adaptadoras

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

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

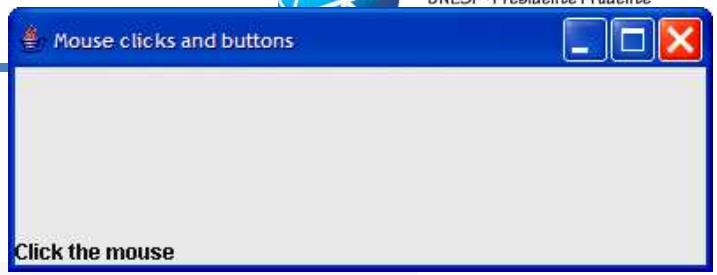
Exemplo




```
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
```

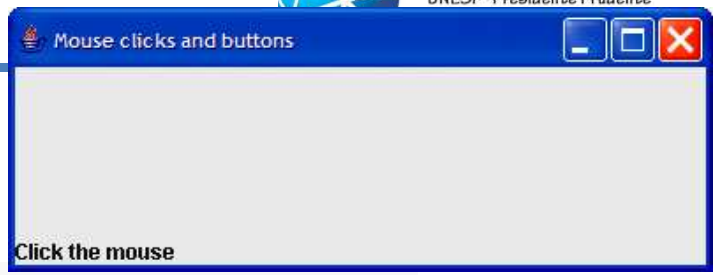
Registra um handler de evento

Exemplo




```
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
```

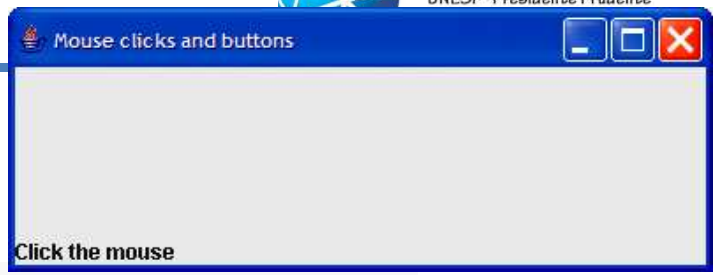
Exemplo




```
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 dire
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

Exemplo




```
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 dire
38             details += " with right mouse button"
39          else if ( event.isAltDown() ) // botão c
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

Exemplo

```
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 dire
38             details += " with right mouse button"
39          else if ( event.isAltDown() ) // botão c
40             details += " with center mouse buttor
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
```

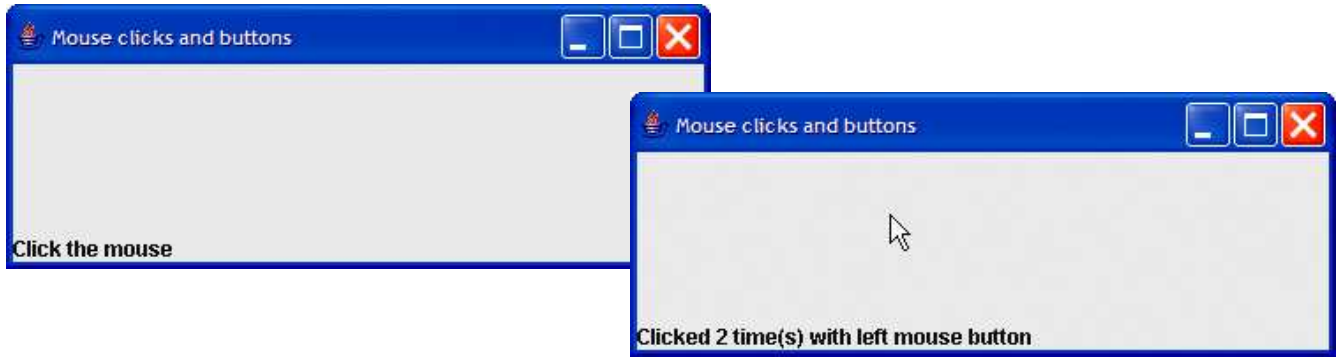
Obtém o número de vezes que o botão do mouse foi clicado

Testa se o botão direito do mouse foi clicado

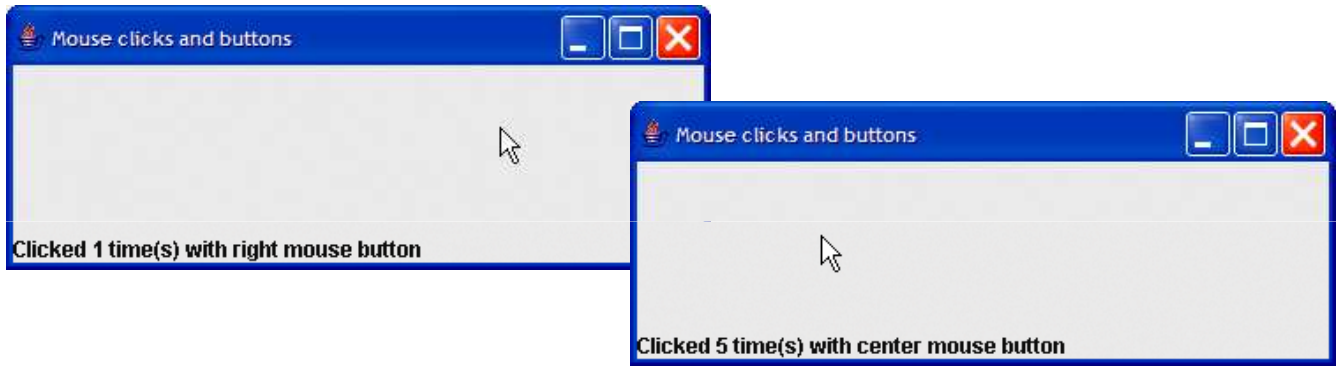
Testa se o botão do meio do mouse foi clicado

Exemplo

```
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
```



Exemplo


Mouse clicks and buttons
Clicked 1 time(s) with right mouse button
Mouse clicks and buttons
Clicked 5 time(s) with center mouse button

Subclasse JPanel para desenhar com o mouse

- Sobrescrevendo a classe JPanel:
  - Fornece uma área dedicada de desenho.

Eventos do Mouse

Os métodos InputEvent que ajudam a distinguir entre os cliques do botão esquerdo, do centro e direito do mouse.

Método InputEvent	Descrição
`isMetaDown()`	Retorna `true` quando o usuário clica no botão direito do mouse em um mouse com dois ou três botões. Para simular um clique de botão direito com um mouse de um botão, o usuário pode manter pressionada a tecla *Meta* no teclado e clicar no botão do mouse.
`isAltDown()`	Retorna `true` quando o usuário clica no botão do mouse do meio em um mouse com três botões. Para simular um clique com o botão do meio do mouse em um mouse com um ou dois botões, o usuário pode pressionar a tecla *Alt* no teclado e clicar no único botão ou no botão esquerdo do mouse.

Método paintComponent

- Desenha em um componente Swing.
- A sobrescrição de método permite criar desenhos personalizados.
- Deve primeiro chamar o método de superclasse quando sobrescrito.

Observação sobre aparência e comportamento

- A maioria dos componentes Swing GUI pode ser transparente ou opaca. Se um componente Swing GUI for opaco, seu fundo será limpo quando seu método paintComponent for chamado. Somente componentes opacos podem exibir uma cor de segundo plano personalizada. Os objetos JPanel são opacos por padrão.

Dica

- No método paintComponent de uma subclasse JComponent, a primeira instrução deve ser sempre uma chamada para o método da superclasse paintComponent a fim de assegurar que um objeto da subclasse seja exibido corretamente.

Definindo a área personalizada de desenho

- Subclasse personalizada de JPanel:
  - Oferece uma área de desenho personalizada.
  - A classe Graphics é utilizada para desenhar nos componente Swing.
  - A classe Point representa uma coordenada x-y.

Exemplo

```
1  // Fig. 11.34: PaintPanel.java
2  // Utilizando class MouseMotionAdapter.
3  import java.awt.Point;
4  import java.awt.Graphics;
5  import java.awt.event.MouseEvent;
6  import java.awt.event.MouseMotionAdapter;
7  import javax.swing.JPanel;
8
9  public class PaintPanel extends JPanel
10 {
11    private int pointCount = 0; // número de contagem de pontos
12
13    // array de 10000 referências java.awt.Point
14    private Point points[] = new Point[ 10000 ];
15
16    // configura a GUI e registra handler de evento de mouse
17    public PaintPanel()
18    {
19       // trata evento de movimento de mouse do frame
20       addMouseMotionListener(
21
```

Exemplo

```
1  // Fig. 11.34: PaintPanel.java
2  // Utilizando class MouseMotionAdapter.
3  import java.awt.Point;
4  import java.awt.Graphics;
5  import java.awt.event.MouseEvent;
6  import java.awt.event.MouseMotionAdapter;
7  import javax.swing.JPanel;
8
9  public class PaintPanel extends JPanel
10 {
11    private int pointCount = 0; // número de contagem de pontos
12
13    // array de 10000 referências java.awt.Point
14    private Point points[] = new Point[ 10000 ];
15
16    // configura a GUI e registra handler de evento de mouse
17    public PaintPanel()
18    {
19       // trata evento de movimento de mouse do frame
20       addMouseMotionListener(
21
```

Cria array de `Points`

Exemplo

```
22          new MouseMotionAdapter() // classe interna anônima
23          {
24             // armazena coordenadas de arrastar e repinta
25             public void mouseDragged( MouseEvent event )
26             {
27                if ( pointCount < points.length )
28                {
29                   points[ pointCount ] = event.getPoint(); // localiza ponto
30                   pointCount++; // número de increment de pontos no array
31                   repaint(); // repinta JFrame
32                } // fim de if
33             } // fim do método mouseDragged
34          } // fim da classe interna anônima
35       ); // fim da chamada para addMouseMotionListener
36    } // fim do construtor PaintPanel
37
38    // desenha oval em um quadro delimitador de 4x4 no local especificado na janela
39    public void paintComponent( Graphics g )
40    {
41       super.paintComponent( g ); // limpa a área de desenho
42
43       // desenha todos os pontos no array
44       for ( int i = 0; i < pointCount; i++ )
45          g.fillOval( points[ i ].x, points[ i ].y, 4, 4 );
46    } // fim do método paintComponent
47 } // fim da classe PaintPanel
```

Exemplo

```
22          new MouseMotionAdapter() // classe interna
23          {
24             // armazena coordenadas de arrastar e r
25             public void mouseDragged( MouseEvent event )
26             {
27                if ( pointCount < points.length )
28                {
29                   points[ pointCount ] = event.getPoint(); // localiza ponto
30                   pointCount++; // número de increment de pontos no array
31                   repaint(); // repinta JFrame
32                } // fim de if
33             } // fim do método mouseDragged
34          } // fim da classe interna anônima
35       ); // fim da chamada para addMouseMotionListener
36    } // fim do construtor PaintPanel
37
38    // desenha oval em um quadro delimitador de 4x4 no local especificado na janela
39    public void paintComponent( Graphics g )
40    {
41       super.paintComponent( g ); // limpa a área de desenho
42
43       // desenha todos os pontos no array
44       for ( int i = 0; i < pointCount; i++ )
45          g.fillOval( points[ i ].x, points[ i ].y, 4, 4 );
46    } // fim do método paintComponent
47 } // fim da classe PaintPanel
```

Classe interna anônima para tratamento de evento

Exemplo

```
22          new MouseMotionAdapter() // classe interna
23          {
24             // armazena coordenadas de arrastar e r
25             public void mouseDragged( MouseEvent event )
26             {
27                if ( pointCount < points.length )
28                {
29                   points[ pointCount ] = event.getPoint(); // localiza ponto
30                   pointCount++; // número de increment de pontos no array
31                   repaint(); // repinta JFrame
32                } // fim de if
33             } // fim do método mouseDragged
34          } // fim da classe interna anônima
35       ); // fim da chamada para addMouseMotionListener
36    } // fim do construtor PaintPanel
37
38    // desenha oval em um quadro delimitador de 4x4 no local especificado na janela
39    public void paintComponent( Graphics g )
40    {
41       super.paintComponent( g ); // limpa a área de desenho
42
43       // desenha todos os pontos no array
44       for ( int i = 0; i < pointCount; i++ )
45          g.fillOval( points[ i ].x, points[ i ].y, 4, 4 );
46    } // fim do método paintComponent
47 } // fim da classe PaintPanel
```

Classe interna anônima para tratamento de evento

Sobrescreve o método `mouseDragged`

Exemplo

```
22          new MouseMotionAdapter() // classe interna
23          {
24             // armazena coordenadas de arrastar e r
25             public void mouseDragged( MouseEvent event )
26             {
27                if ( pointCount < points.length )
28                {
29                   points[ pointCount ] = event.getPoint(); // localiza ponto
30                   pointCount++; // número de increm
31                   repaint(); // repinta JFrame
32                } // fim de if
33             } // fim do método mouseDragged
34          } // fim da classe interna anônima
35       ); // fim da chamada para addMouseMotionListener
36    } // fim do construtor PaintPanel
37
38    // desenha oval em um quadro delimitador de 4x4 no local especificado na janela
39    public void paintComponent( Graphics g )
40    {
41       super.paintComponent( g ); // limpa a área de desenho
42
43       // desenha todos os pontos no array
44       for ( int i = 0; i < pointCount; i++ )
45          g.fillOval( points[ i ].x, points[ i ].y, 4, 4 );
46    } // fim do método paintComponent
47 } // fim da classe PaintPanel
```

Classe interna anônima para tratamento de evento

Sobrescreve o método `mouseDragged`

Obtém a localização do cursor do mouse

Exemplo

```
22          new MouseMotionAdapter() // classe interna
23          {
24             // armazena coordenadas de arrastar e r
25             public void mouseDragged( MouseEvent event )
26             {
27                if ( pointCount < points.length )
28                {
29                   points[ pointCount ] = event.getPoint(); // localiza ponto
30                   pointCount++; // número de increm
31                   repaint(); // repinta JFrame
32                } // fim de if
33             } // fim do método mouseDragged
34          } // fim da classe interna anônima
35       ); // fim da chamada para addMouseMotionListener
36    } // fim do construtor PaintPanel
37
38    // desenha oval em um quadro delimitador de 4x4 no local especificado na janela
39    public void paintComponent( Graphics g )
40    {
41       super.paintComponent( g ); // limpa a área de desenho
42
43       // desenha todos os pontos no array
44       for ( int i = 0; i < pointCount; i++ )
45          g.fillOval( points[ i ].x, points[ i ].y, 4, 4 );
46    } // fim do método paintComponent
47 } // fim da classe PaintPanel
```

Classe interna anônima para tratamento de evento

Sobrescreve o método `mouseDragged`

Obtém a localização do cursor do mouse

Repinta o `JFrame`

Exemplo

```
22          new MouseMotionAdapter() // classe interna
23          {
24             // armazena coordenadas de arrastar e r
25             public void mouseDragged( MouseEvent event )
26             {
27                if ( pointCount < points.length )
28                {
29                   points[ pointCount ] = event.getPoint(); // localiza ponto
30                   pointCount++; // número de increm
31                   repaint(); // repinta JFrame
32                } // fim de if
33             } // fim do método mouseDragged
34          } // fim da classe interna anônima
35       ); // fim da chamada para addMouseMotionListener
36    } // fim do construtor PaintPanel
37
38    // desenha oval em um quadro delimitador de 4x4 no local especificado na janela
39    public void paintComponent( Graphics g )
40    {
41       super.paintComponent( g ); // limpa a área de desenho
42
43       // desenha todos os pontos no array
44       for ( int i = 0; i < pointCount; i++ )
45          g.fillOval( points[ i ].x, points[ i ].y, 4, 4 );
46    } // fim do método paintComponent
47 } // fim da classe PaintPanel
```

Classe interna anônima para tratamento de evento

Sobrescreve o método `mouseDragged`

Obtém a localização do cursor do mouse

Repinta o `JFrame`

Obtém as coordenadas *x* e *y* de `Point`

Observação sobre aparência e comportamento

- Chamar repaint para um componente Swing GUI indica que o componente deve ser atualizado na tela o mais rápido possível. O fundo do componente GUI é limpo somente se o componente for opaco. Para o método JComponent setOpaque pode ser passado um argumento boolean indicando se o componente é opaco (true) ou transparente (false).

Exemplo

```
1  // Fig. 11.35: Painter.java
2  // Testando PaintPanel.
3  import java.awt.BorderLayout;
4  import javax.swing.JFrame;
5  import javax.swing.JLabel;
6
7  public class Painter
8  {
9     public static void main( String args[] )
10    {
11       // cria JFrame
12       JFrame application = new JFrame( "A simple paint program" );
13
14       PaintPanel paintPanel = new PaintPanel(); // cria o painel de pintura
15       application.add( paintPanel, BorderLayout.CENTER ); // no centro
16
17       // cria um rótulo e o coloca em SOUTH de BorderLayout
18       application.add( new JLabel( "Drag the mouse to draw" ),
19          BorderLayout.SOUTH );
20
21       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
22       application.setSize( 400, 200 ); // configura o tamanho do frame
23       application.setVisible( true ); // exibe o frame
24    } // fim de main
25 } // fim da classe Painter
```

Exemplo

```
1  // Fig. 11.35: Painter.java
2  // Testando PaintPanel.
3  import java.awt.BorderLayout;
4  import javax.swing.JFrame;
5  import javax.swing.JLabel;
6
7  public class Painter
8  {
9     public static void main( String args[] )
10    {
11       // cria JFrame
12       JFrame application = new JFrame( "A simple paint program" );
13
14       PaintPanel paintPanel = new PaintPanel(); // cria o painel de pintura
15       application.add( paintPanel, BorderLayout.CENTER ); // no centro
16
17       // cria um rótulo e o coloca em SOUTH de BorderLayout
18       application.add( new JLabel( "Drag the mouse to draw" ),
19          BorderLayout.SOUTH );
20
21       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
22       application.setSize( 400, 200 ); // configura o tamanho do frame
23       application.setVisible( true ); // exibe o frame
24    } // fim de main
25 } // fim da classe Painter
```

Cria uma instância do painel de desenho personalizado

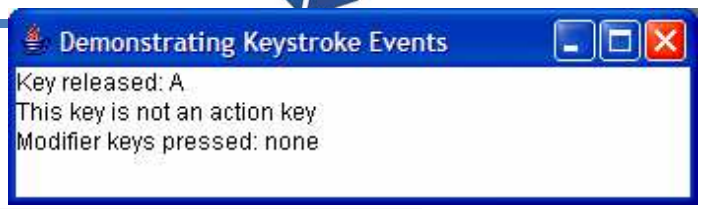
Exemplo


A simple paint program
JAVA
How To
Program
Drag the mouse to draw

Tratamento de eventos de teclado

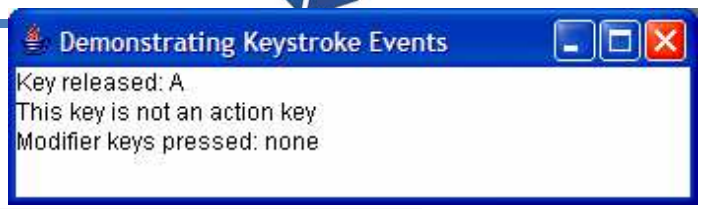
- Interface KeyListener:
  - Para tratar eventos de teclado — KeyEvents.
  - Declara os métodos keyPressed, keyReleased e keyTyped, sendo que cada um recebe um KeyEvent como seu argumento.

Exemplo




```
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;
9  public class KeyDemoFrame extends JFrame implements KeyListener
10 {
11    private String line1 = ""; // primeira linha de textarea
12    private String line2 = ""; // segunda linha de textarea
13    private String line3 = ""; // terceira linha de textarea
14    private JTextArea textArea; // textarea a exibir saída
16    // construtor KeyDemoFrame
17    public KeyDemoFrame()
18    {
19       super( "Demonstrating Keystroke Events" );
21       textArea = new JTextArea( 10, 15 ); // configura JTextArea
22       textArea.setText( "Press any key on the keyboard..." );
23       textArea.setEnabled( false ); // disativa textarea
24       textArea.setDisabledTextColor( Color.BLACK ); // configura cor de texto
25       add( textArea ); // adiciona textarea ao JFrame
26
27       addKeyListener( this ); // permite que o frame processe eventos de teclado
28    } // fim do construtor KeyDemoFrame
```

Exemplo




```
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;
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
16    // construtor KeyDemoFrame
17    public KeyDemoFrame()
18    {
19       super( "Demonstrating Keystroke Events" );
21       textArea = new JTextArea( 10, 15 ); // configura JTextArea
22       textArea.setText( "Press any key on the keyboard..." );
23       textArea.setEnabled( false ); // disativa textarea
24       textArea.setDisabledTextColor( Color.BLACK ); // configura cor de texto
25       add( textArea ); // adiciona textarea ao JFrame
26
27       addKeyListener( this ); // permite que o frame processe eventos de teclado
28    } // fim do construtor KeyDemoFrame
```

Implementa a interface `KeyListener`

Exemplo


```
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;
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
16    // construtor KeyDemoFrame
17    public KeyDemoFrame()
18    {
19       super( "Demonstrating Keystroke Events" );
21       textArea = new JTextArea( 10, 15 ); // configura JTextArea
22       textArea.setText( "Press any key on the keyboard..." );
23       textArea.setEnabled( false ); // disativa textarea
24       textArea.setDisabledTextColor( Color.BLACK ); // configura cor de texto
25       add( textArea ); // adiciona textarea ao JFrame
26
27       addKeyListener( this ); // permite que o frame processe eventos de teclado
28    } // fim do construtor KeyDemoFrame
```

Implementa a interface `KeyListener`

Configura a cor de segundo plano

Exemplo

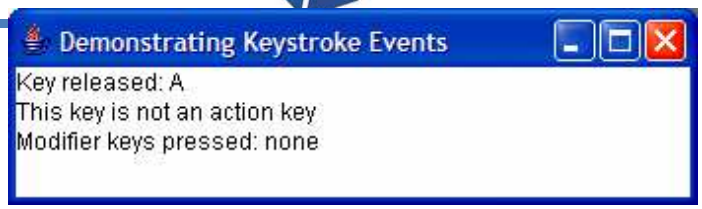

```
3  import java.awt.Color;
4  import java.awt.event.KeyListener;
5  import java.awt.event.KeyEvent;
6  import javax.swing.JFrame;
7  import javax.swing.JTextArea;
9  public class KeyDemoFrame extends JFrame implements KeyListener
10 {
11    private String line1 = ""; // primeira linha de textarea
12    private String line2 = ""; // segunda linha de tex
13    private String line3 = ""; // terceira linha de textarea
14    private JTextArea textArea; // textarea a exibir saída
16    // construtor KeyDemoFrame
17    public KeyDemoFrame()
18    {
19       super( "Demonstrating Keystroke Events" );
21       textArea = new JTextArea( 10, 15 ); // configura JTextArea
22       textArea.setText( "Press any key on the keyboar
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
```

Implementa a interface `KeyListener`

Configura a cor de segundo plano

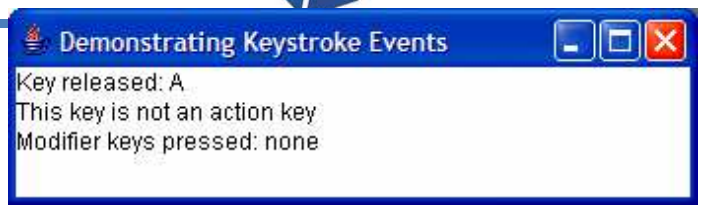
Registra a própria aplicação como um handler de evento

Exemplo




```
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 das linhas dois e três
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 das linhas dois e três
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
```

Exemplo




```
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 das linhas dois e três
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 das linhas dois e três
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

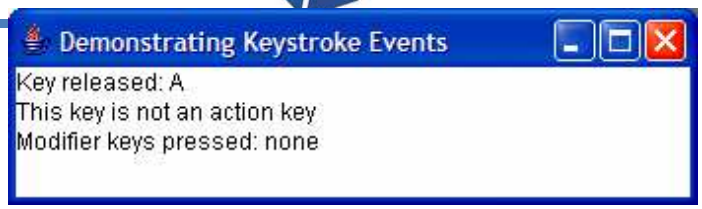
Exemplo


```
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
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 das linhas dois e três
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

Exemplo



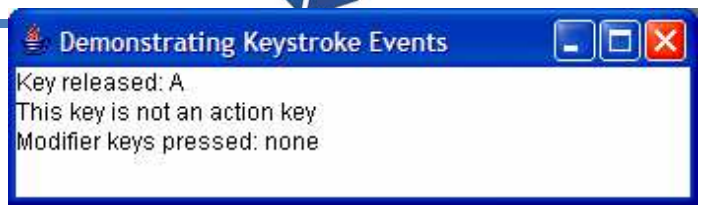

```
30    // trata pressionamento de qualquer tecla
31    public void keyPressed( KeyEvent event )
32    {
33       line1 = String.format( "Key pressed: %s",
34          event.getKeyText( event.getKeyCode() ) ); // gera saída de tecla pressionada
35       setLines2and3( event ); // configura a saída
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 das linhas dois e três
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

Declara o método `keyReleased`

Exemplo




```
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
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
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
```

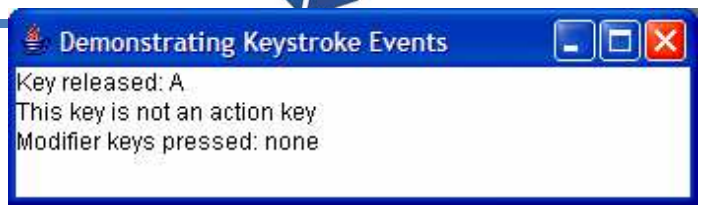
Declara o método `keyPressed`

Obtém o código da tecla pressionada

Declara o método `keyReleased`

Obtém o código da tecla liberada

Exemplo




```
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
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
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

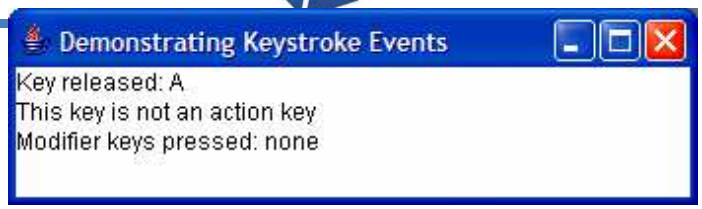
Obtém o código da tecla pressionada

Declara o método keyReleased

Obtém o código da tecla liberada

Declara o método keyTyped

Exemplo




```
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
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
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

Declara o método keyReleased

Obtém o código da tecla liberada

Declara o método keyTyped

Gera saída do caractere digitado

Exemplo


```
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: %s",
62          ( temp.equals( "" ) ? "none" : temp ) ); // envia modificadores para a saída
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
```

Exemplo


```
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: %s",
62          ( temp.equals( "" ) ? "none" : temp ) ); // envia modificadores para a saída
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

Exemplo

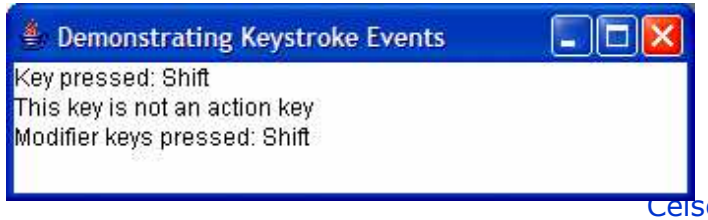

```
53    // configura segunda e terceira linhas de saída
54    private void setLines2and3( KeyEvent event )
55    {
56       line2 = String.format( "This key is %san action key",
57          ( event.isActionKey() ? "" : "not " ) );
58
59       String temp = event.getKeyModifiersText( event.getModifiers() );
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
```

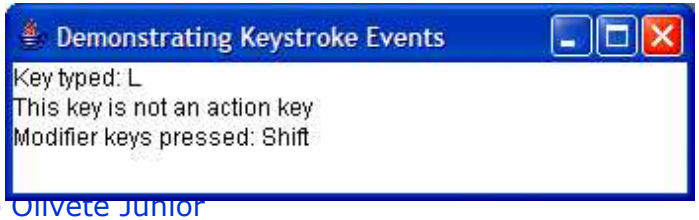
Testa se era uma tecla de ação

Determina quaisquer modificadores pressionados

Exemplo

```
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
```





Exemplo


Demonstrating Keystroke Events
Key released: L
This key is not an action key
Modifier keys pressed: Shift


Demonstrating Keystroke Events
Key pressed: F1
This key is an action key
Modifier keys pressed: none


Demonstrating Keystroke Events
Key released: F1
This key is an action key
Modifier keys pressed: none

Gerenciadores de layout

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

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.

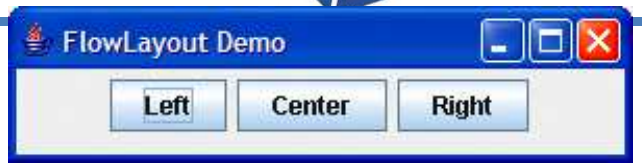
Gerenciadores de layout

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 `JFrames` (e outras janelas). Organiza os componentes em cinco áreas: `NORTH`, `SOUTH`, `EAST`, `WEST` e `CENTER`.
`GridLayout`	Organiza os componentes nas linhas e colunas.

Exemplo


```
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;
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
```

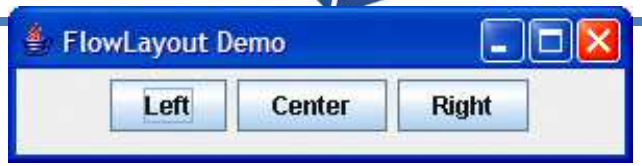
Exemplo




```
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;
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
```

Cria FlowLayout

Exemplo

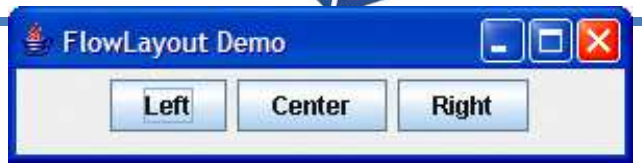



```
2  // Demonstrando os alinhamentos de FlowLayout.
3  import java.awt.FlowLayout;
4  import java.awt.Container;
5  import java.awt.event.ActionListener;
6  import java.awt.event.ActionEvent;
7  import javax.swing.JFrame;
8  import javax.swing.JButton;
10 public class FlowLayoutFrame extends JFrame
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
```

Cria FlowLayout

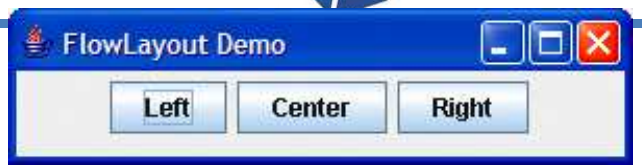
Configura o layout da aplicação

Exemplo




```
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(
32          new ActionListener() // classe interna anônima
33          {
34             // processa o evento leftJButton
35             public void actionPerformed( ActionEvent event )
36             {
37                layout.setAlignment( FlowLayout.LEFT );
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
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(
50          new ActionListener() // classe interna anônima
51          {
52             // processa evento centerJButton
53             public void actionPerformed( ActionEvent event )
54             {
55                layout.setAlignment( FlowLayout.CENTER );
```

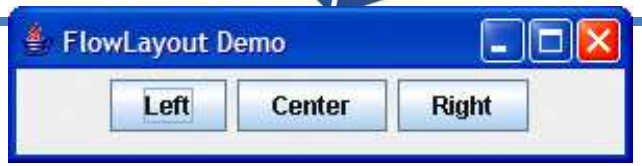
Exemplo




```
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(
32          new ActionListener() // classe interna anônim
33          {
34             // processa o evento leftJButton
35             public void actionPerformed( ActionEvent event )
36             {
37                layout.setAlignment( FlowLayout.LEFT );
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
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(
50          new ActionListener() // classe interna anônima
51          {
52             // processa evento centerJButton
53             public void actionPerformed( ActionEvent event )
54             {
55                layout.setAlignment( FlowLayout.CENTER );
```

Adiciona `JButton`; `FlowLayout` tratará o posicionamento

Exemplo

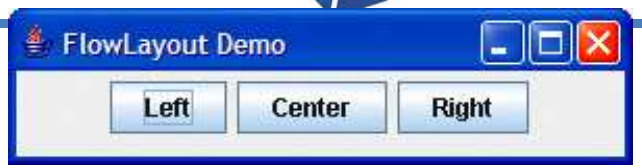



```
27       // configura leftJButton e registra listener
28       leftJButton = new JButton( "Left" ); // cria botão Left
29       add( leftJButton ); // adiciona o botão Left ao frame
30       leftJButton.addActionListener(
32          new ActionListener() // classe interna anônim
33          {
34             // processa o evento leftJButton
35             public void actionPerformed( ActionEvent event )
36             {
37                layout.setAlignment( FlowLayout.LEFT );
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
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(
50          new ActionListener() // classe interna anônima
51          {
52             // processa evento centerJButton
53             public void actionPerformed( ActionEvent event )
54             {
55                layout.setAlignment( FlowLayout.CENTER );
```

Adiciona `JButton`; `FlowLayout` tratará o posicionamento

Configura o alinhamento à esquerda

Exemplo



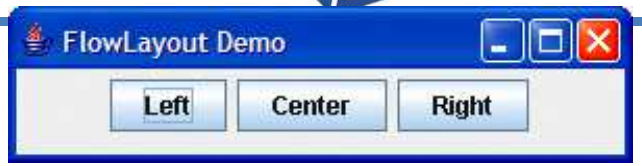

```
27       // configura leftJButton e registra listener
28       leftJButton = new JButton( "Left" ); // cria botão Left
29       add( leftJButton ); // adiciona o botão Left ao frame
30       leftJButton.addActionListener(
32          new ActionListener() // classe interna anônim
33          {
34             // processa o evento leftJButton
35             public void actionPerformed( ActionEvent event )
36             {
37                layout.setAlignment( FlowLayout.LEFT );
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
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(
50          new ActionListener() // classe interna anônima
51          {
52             // processa evento centerJButton
53             public void actionPerformed( ActionEvent event )
54             {
55                layout.setAlignment( FlowLayout.CENTER );
```

Adiciona `JButton`; `FlowLayout` tratará o posicionamento

Configura o alinhamento à esquerda

Ajusta o layout

Exemplo




```
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(
32           new ActionListener() // classe interna anônim
33           {
34              // processa o evento leftJButton
35              public void actionPerformed( ActionEvent event )
36              {
37                 layout.setAlignment( FlowLayout.LEFT );
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
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(
50           new ActionListener() // classe interna anônim
51           {
52              // processa evento centerJButton
53              public void actionPerformed( ActionEvent event )
54              {
55                 layout.setAlignment( FlowLayout.CENTER );
```

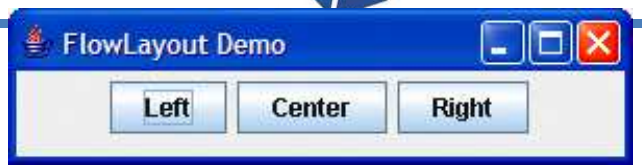
Adiciona `JButton`; `FlowLayout` tratará o posicionamento

Configura o alinhamento à esquerda

Ajusta o layout

Adiciona `JButton`; `FlowLayout` tratará o posicionamento

Exemplo




```
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(
32          new ActionListener() // classe interna anônim
33          {
34             // processa o evento leftJButton
35             public void actionPerformed( ActionEvent event )
36             {
37                layout.setAlignment( FlowLayout.LEFT );
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
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(
50          new ActionListener() // classe interna anônim
51          {
52             // processa evento centerJButton
53             public void actionPerformed( ActionEvent ev
54             {
55                layout.setAlignment( FlowLayout.CENTER );
```

Adiciona `JButton`; `FlowLayout` tratará o posicionamento

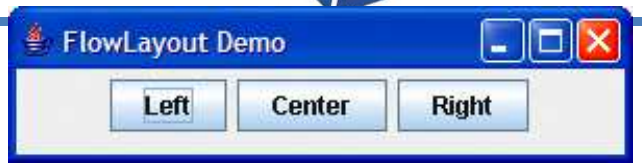
Configura o alinhamento à esquerda

Ajusta o layout

Adiciona `JButton`; `FlowLayout` tratará o posicionamento

Configura o alinhamento no centro

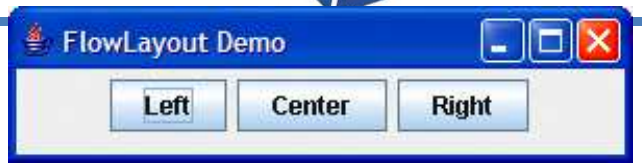
Exemplo




```
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 anônima
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

Exemplo

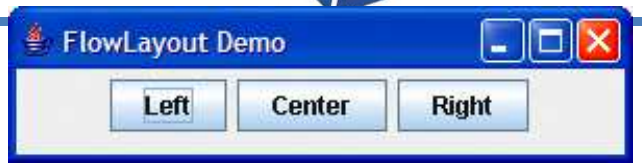



```
57                // realinha os componentes anexados
58                layout.layoutContainer( container );
59             } // fim do método actionPerformed
60          } // fim da classe interna anônima
61       ); // fim da chamada para addActionListener
62
63       // configura rightJButton 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

Exemplo



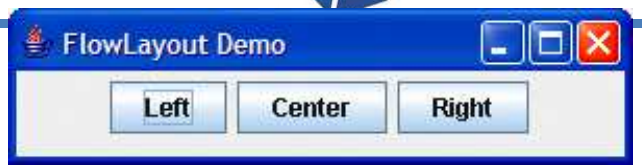

```
57                // realinha os componentes anexados
58                layout.layoutContainer( container );
59             } // fim do método actionPerformed
60          } // fim da classe interna anônima
61       ); // fim da chamada para addActionListener
62
63       // configura rightJButton 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

Exemplo




```
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

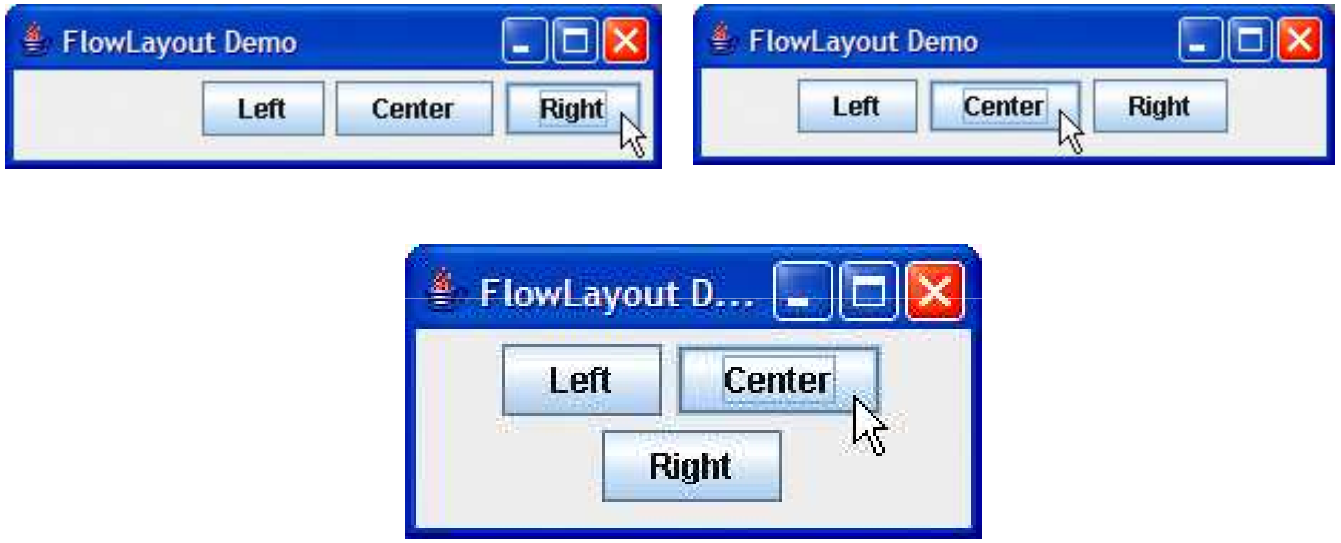
Configura o alinhamento à direita

Ajusta o layout

Exemplo

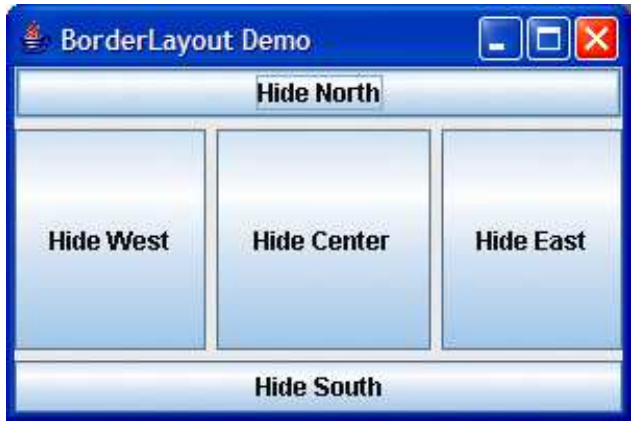
```
1  // Fig. 11.40: FlowLayoutDemo.java
2  // Testando FlowLayoutFrame.
3  import javax.swing.JFrame;
4
5  public class FlowLayoutDemo
6  {
7     public static void main( String args[] )
8     {
9        FlowLayoutFrame flowLayoutFrame = new FlowLayoutFrame();
10       flowLayoutFrame.setDefaultCloseOperation( 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
```

Exemplo


FlowLayout Demo
Left
Center
Right
FlowLayout Demo
Left
Center
Right
FlowLayout D...
Left
Center
Right

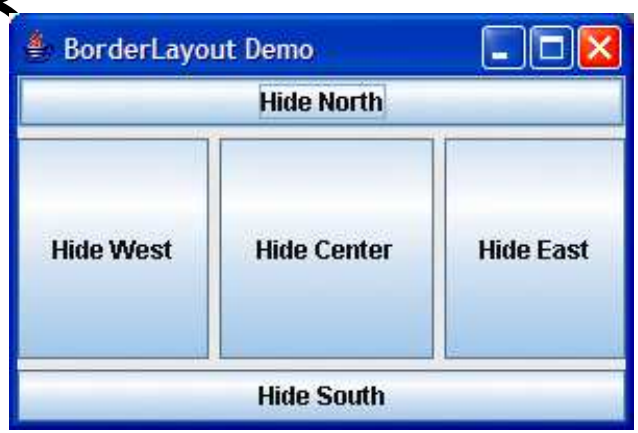
BorderLayout

- 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

- 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



Exemplo


```
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;
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
16    // configura GUI e tratamento de evento
17    public BorderLayoutFrame()
18    {
19       super( "BorderLayout Demo" );
20
21       layout = new BorderLayout( 5, 5 ); // 5 pixel gaps
22       setLayout( layout ); // configura o layout de frame
23       buttons = new JButton[ names.length ]; // configura o tamanho do array
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
```

Exemplo


```
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;
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
16    // configura GUI e tratamento de evento
17    public BorderLayoutFrame()
18    {
19       super( "BorderLayout Demo" );
20
21       layout = new BorderLayout( 5, 5 ); // 5 pixel gaps
22       setLayout( layout ); // configura o layout de frame
23       buttons = new JButton[ names.length ]; // configura o tamanho do array
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`

Exemplo


```
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;
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
16    // configura GUI e tratamento de evento
17    public BorderLayoutFrame()
18    {
19       super( "BorderLayout Demo" );
20
21       layout = new BorderLayout( 5, 5 ); // 5 pixel gaps
22       setLayout( layout ); // configura o layout de frame
23       buttons = new JButton[ names.length ]; // configura o tamanho do array
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`

Exemplo


```
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;
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
16    // configura GUI e tratamento de evento
17    public BorderLayoutFrame()
18    {
19       super( "BorderLayout Demo" );
20
21       layout = new BorderLayout( 5, 5 ); // 5
22       setLayout( layout ); // configura o layout de frame
23       buttons = new JButton[ names.length ]; // configura o tamanho do array
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

Exemplo


```
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;
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
16    // configura GUI e tratamento de evento
17    public BorderLayoutFrame()
18    {
19       super( "BorderLayout Demo" );
20
21       layout = new BorderLayout( 5, 5 ); // 5
22       setLayout( layout ); // configura o layout de frame
23       buttons = new JButton[ names.length ]; // configura o tamanho do array
25       // cria JButtons e registra listeners para eles
26       for ( int count = 0; count < names.
27       {
28          buttons[ count ] = new JButton( names[ count ] );
29          buttons[ count ].addActionListener( this );
30       } // fim de for
```

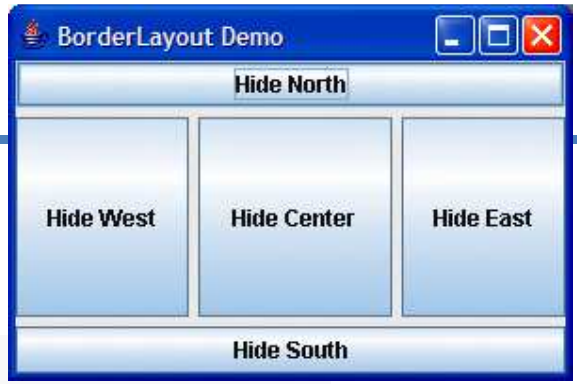
Declara a variável de instância `BorderLayout`

Cria `BorderLayout`

Configura o layout

Registra um handler de evento

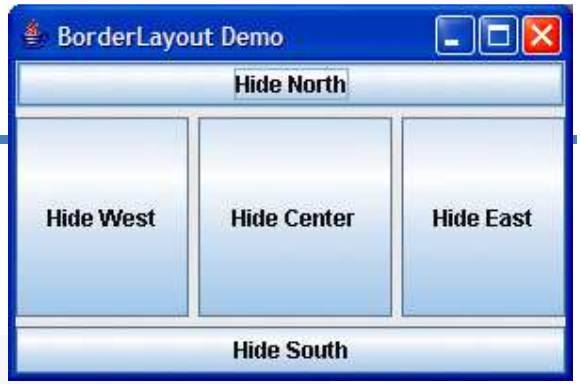
Exemplo




```
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 botão clicado
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

Exemplo

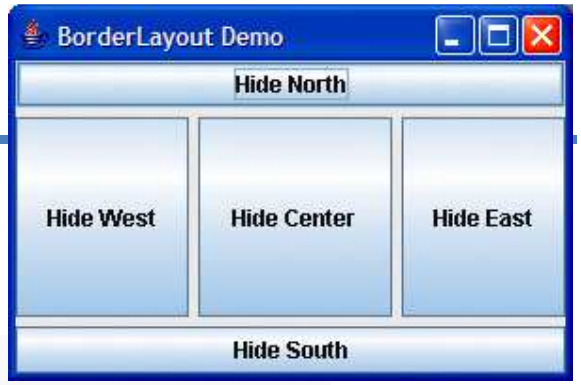



```
31
32       add( buttons[ 0 ], BorderLayout.NORTH ); // adiciona botão para o norte
33       add( buttons[ 1 ], BorderLayout.SOUTH ); // adiciona botão para o sul
34       add( buttons[ 2 ], BorderLayout.EAST ); // adiciona botão para o leste
35       add( buttons[ 3 ], BorderLayout.WEST ); // adiciona 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 botão clicado
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

Exemplo



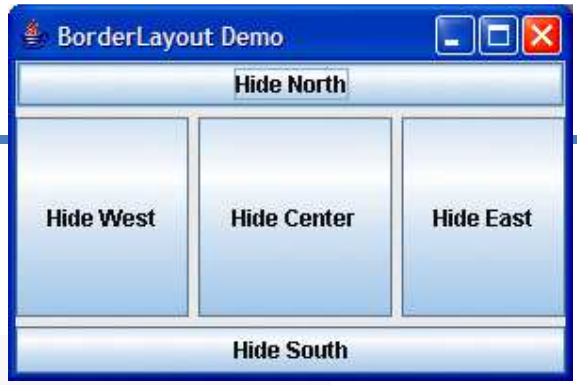

```
31
32       add( buttons[ 0 ], BorderLayout.NORTH ); // adiciona botão para o norte
33       add( buttons[ 1 ], BorderLayout.SOUTH ); // adiciona botão para o sul
34       add( buttons[ 2 ], BorderLayout.EAST ); // adiciona botão para o leste
35       add( buttons[ 3 ], BorderLayout.WEST ); // adiciona 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

Exemplo




```
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

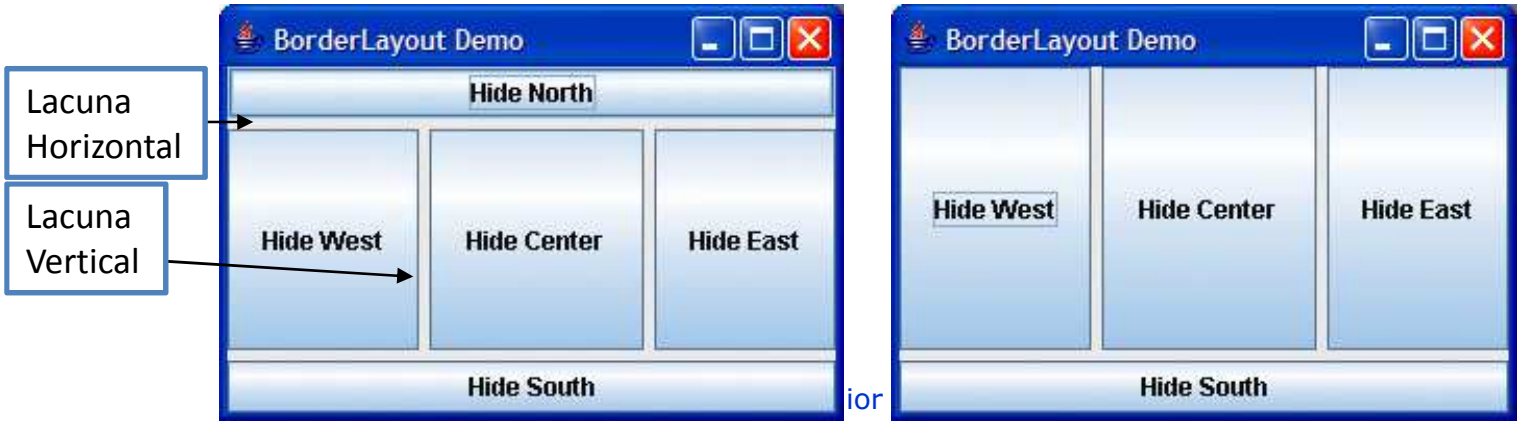
Torna o botão visível

Atualiza o layout

Exemplo

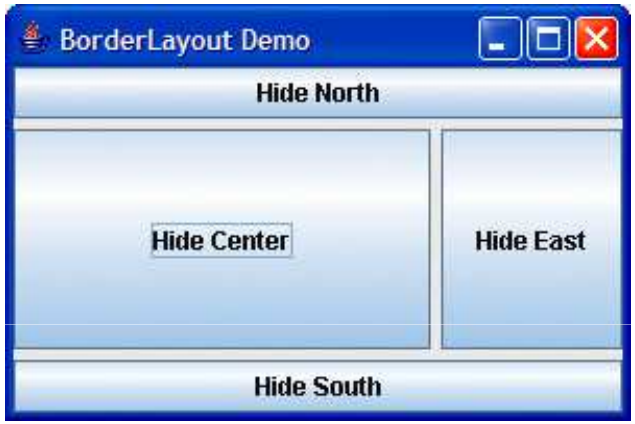
```
1  // Fig. 11.42: BorderLayoutDemo.java
2  // Testando BorderLayoutFrame.
3  import javax.swing.JFrame;
4
5  public class BorderLayoutDemo
6  {
7     public static void main( String args[] )
8     {
9        BorderLayoutFrame borderLayoutFrame = new BorderLayoutFrame();
10       borderLayoutFrame.setDefaultCloseOperation( 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
```

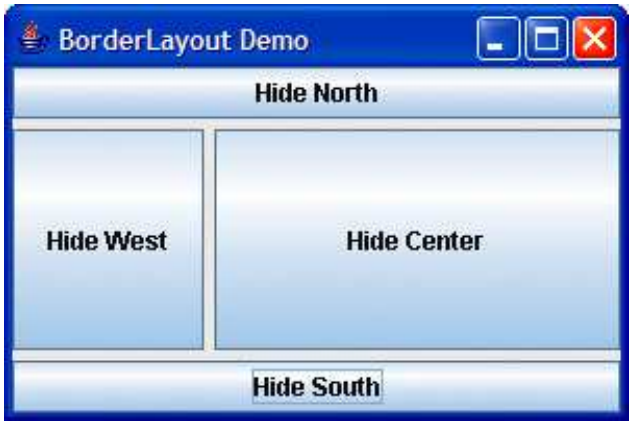
lacuna vertical

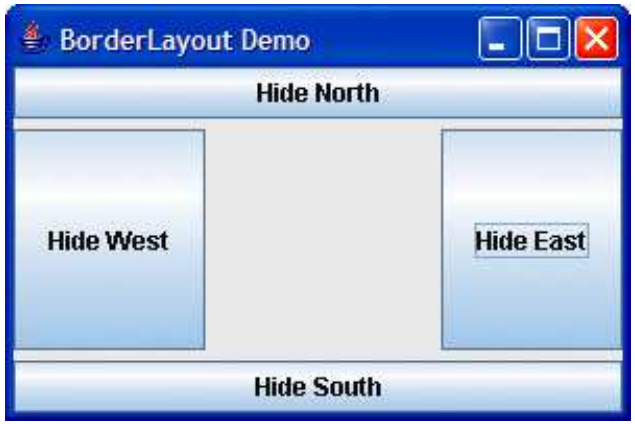


Exemplo


BorderLayout Demo
Hide North
Hide West
Hide Center
Hide East

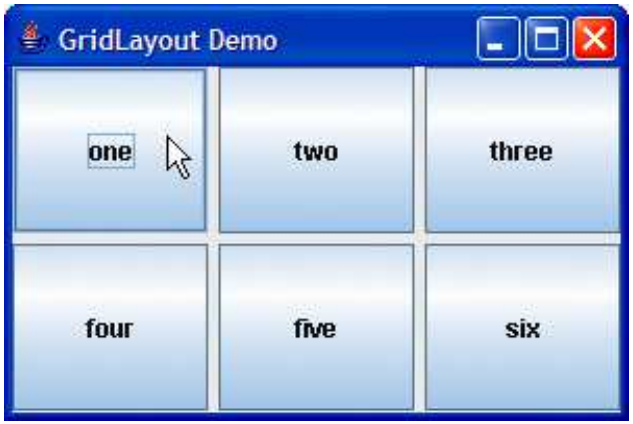

BorderLayout Demo
Hide North
Hide Center
Hide East
Hide South


BorderLayout Demo
Hide North
Hide West
Hide Center
Hide South

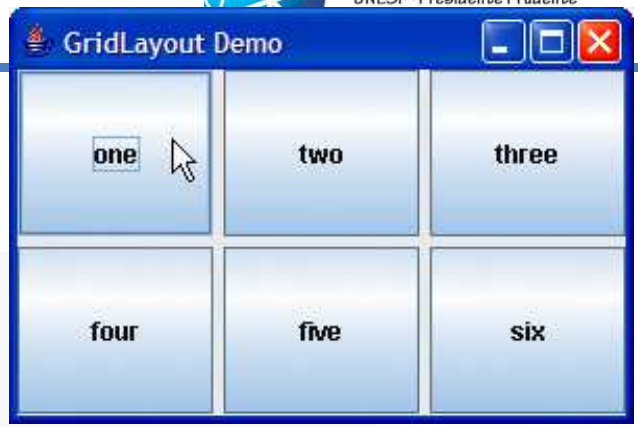

BorderLayout Demo
Hide North
Hide West
Hide East
Hide South

GridLayout

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

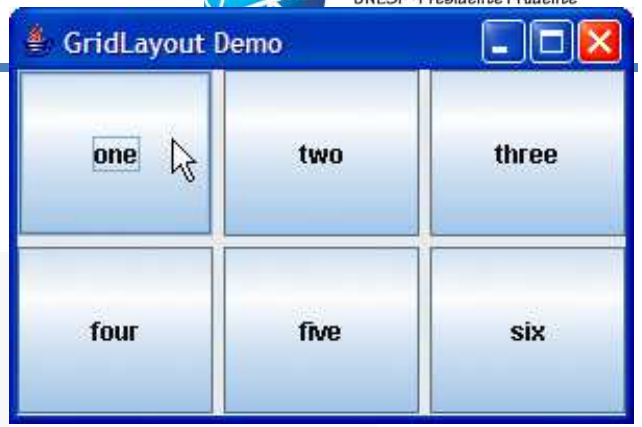


Exemplo




```
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;
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 dois layouts
16    private Container container; // contêiner do frame
17    private GridLayout gridLayout1; // primeiro gridlayout
18    private GridLayout gridLayout2; // segundo gridlayout
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 ]; // cria array de JButtons
```

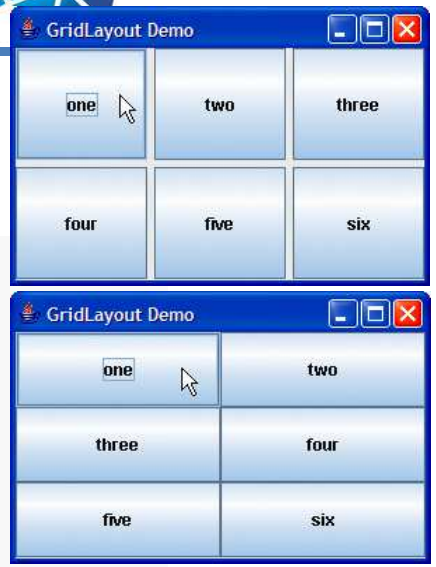
Exemplo




```
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;
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 d
16    private Container container; // contêiner do frame
17    private GridLayout gridLayout1; // primeiro gridlayout
18    private GridLayout gridLayout2; // segundo gridlayout
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 ]; // cria array de JButtons
```

Declara duas variáveis de instância GridLayout

Exemplo

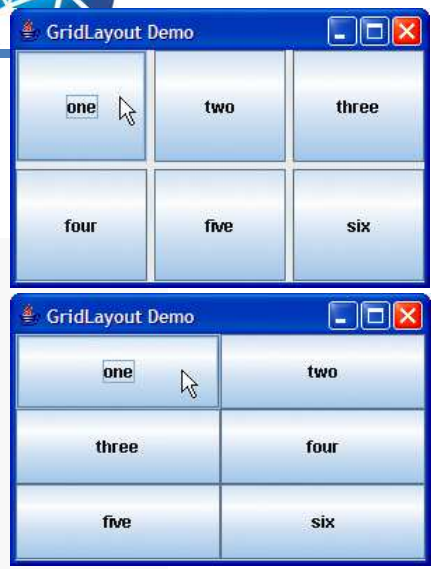



```
3  import java.awt.GridLayout;
4  import java.awt.Container;
5  import java.awt.event.ActionListener;
6  import java.awt.event.ActionEvent;
7  import javax.swing.JFrame;
8  import javax.swing.JButton;
10 public class GridLayoutFrame extends JFrame implements ActionListener
11 {
12    private JButton buttons[]; // array de botões
13    private final String names[] =
14       { "one", "two", "three", "four", "five", "six"
15    private boolean toggle = true; // alterna entre d
16    private Container container; // contêiner do frame
17    private GridLayout gridLayout1; // primeiro gridlayout
18    private GridLayout gridLayout2; // segundo gridlayout
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 ]; // cria array de JButtons
```

Declara duas variáveis de instância `GridLayout`

Cria `GridLayout`

Exemplo



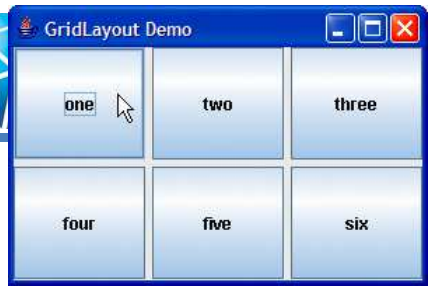

```
3  import java.awt.GridLayout;
4  import java.awt.Container;
5  import java.awt.event.ActionListener;
6  import java.awt.event.ActionEvent;
7  import javax.swing.JFrame;
8  import javax.swing.JButton;
10 public class GridLayoutFrame extends JFrame implements ActionListener
11 {
12    private JButton buttons[]; // array de botões
13    private final String names[] =
14       { "one", "two", "three", "four", "five", "six"
15    private boolean toggle = true; // alterna entre d
16    private Container container; // contêiner do frame
17    private GridLayout gridLayout1; // primeiro gridlayout
18    private GridLayout gridLayout2; // segundo gridlayout
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 ];
```

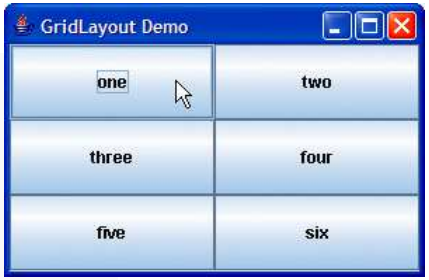
Declara duas variáveis de instância `GridLayout`

Cria `GridLayout`

Configura o layout

Exemplo






```
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 ); // configura layout como segundo
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`

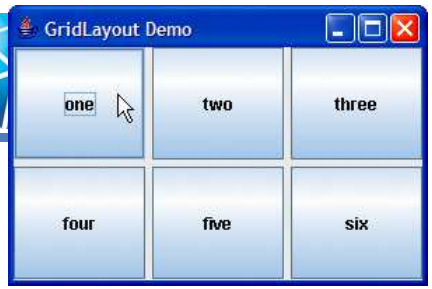
Exemplo

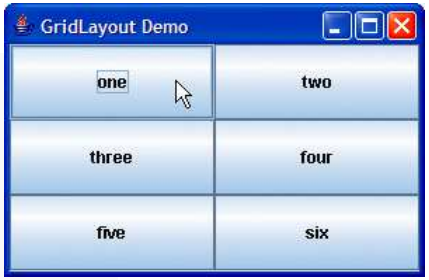

```
30        for ( int count = 0; count < names.length; count++ )
31        {
32           buttons[ count ] = new JButton( names[ count ] );
33           buttons[ count ].addActionListener( this ); // registra listener
34           add( buttons[ count ] ); // adiciona botão ao JFrame
35        } // 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 ); // configura layout como segundo
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

Exemplo






```
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

Exemplo


```
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

Exemplo

```
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
```

Utilizando painéis para gerenciar layouts mais complexos

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

Exemplo


```
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
```

Exemplo


```
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

Exemplo


```
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`

Exemplo


```
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

Exemplo


```
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
```

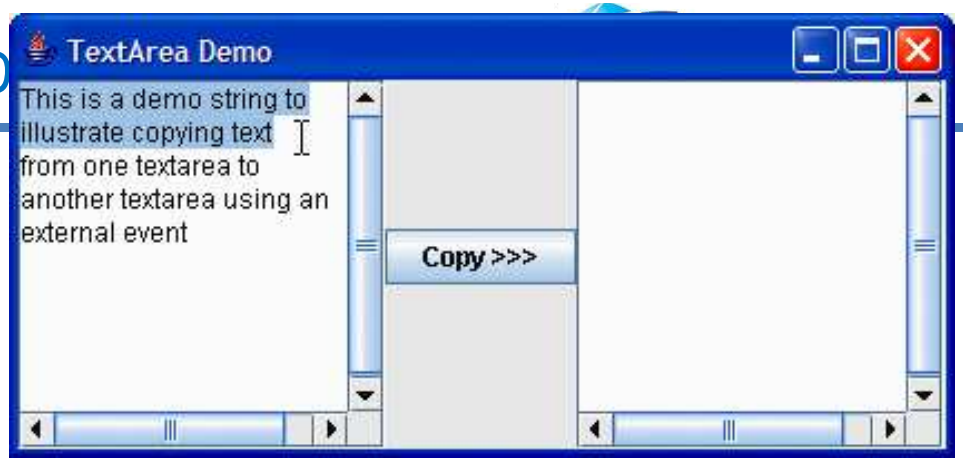
Adiciona um botão ao painel

Adiciona o painel à aplicação

Exemplo

```
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
```

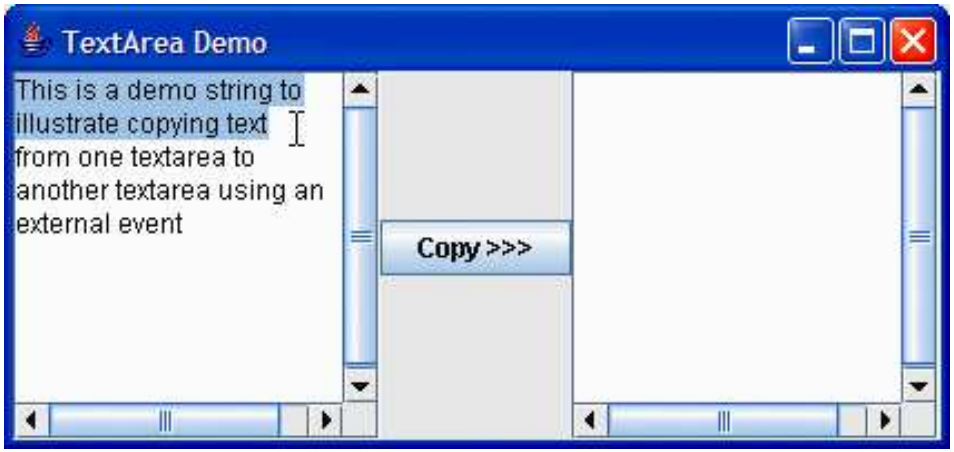
JTextArea



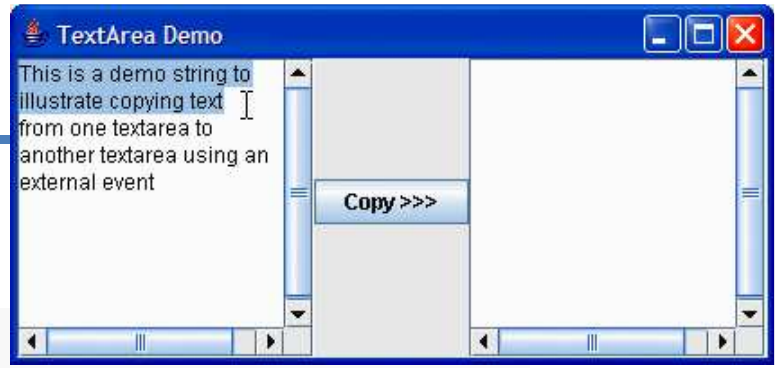
- JTextArea:
  - Fornece uma área para manipular múltiplas linhas de texto.
- Contêiner Box:
  - Subclasse de Container.
  - Utiliza um gerenciador de layout BoxLayout.

Observação

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

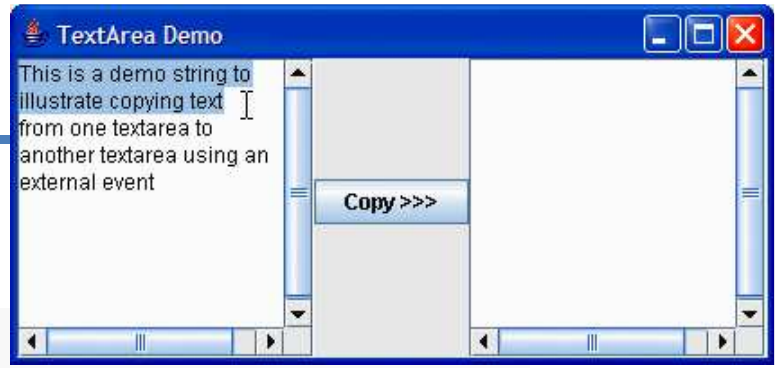


Exemplo




```
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
```

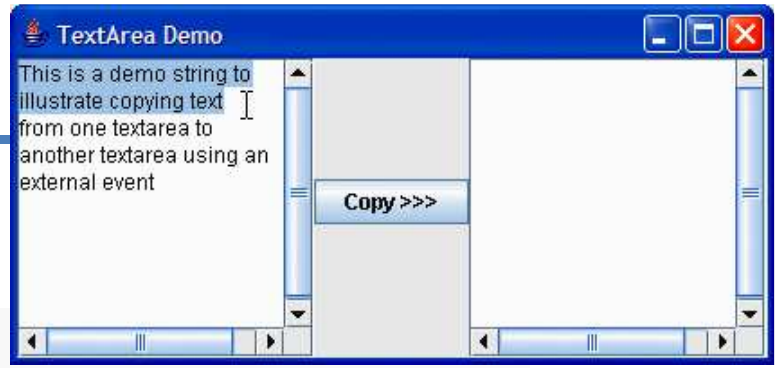
Exemplo




```
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`

Exemplo

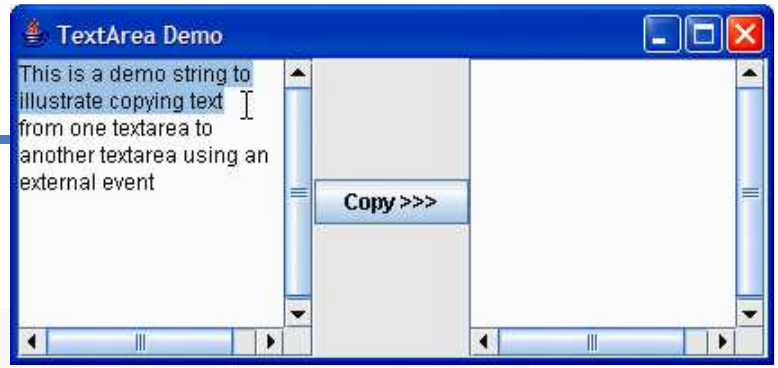



```
3  import java.awt.event.ActionListener;
4  import java.awt.event.ActionEvent;
5  import javax.swing.Box;
6  import javax.swing.JFrame;
7  import javax.swing.JTextArea;
8  import javax.swing.JButton;
9  import javax.swing.JScrollPane;
10
11 public class TextAreaFrame extends JFrame
12 {
13    private 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

Exemplo



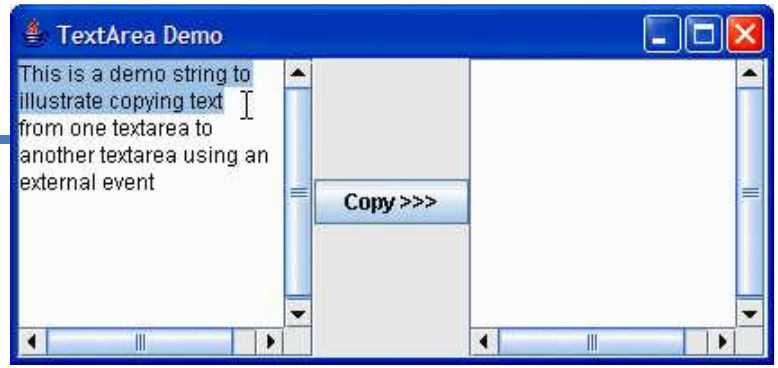

```
3  import java.awt.event.ActionListener;
4  import java.awt.event.ActionEvent;
5  import javax.swing.Box;
6  import javax.swing.JFrame;
7  import javax.swing.JTextArea;
8  import javax.swing.JButton;
9  import javax.swing.JScrollPane;
10
11 public class TextAreaFrame extends JFrame
12 {
13    private 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 \
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

Exemplo




```
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

Exemplo

```
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

Exemplo

```
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
```

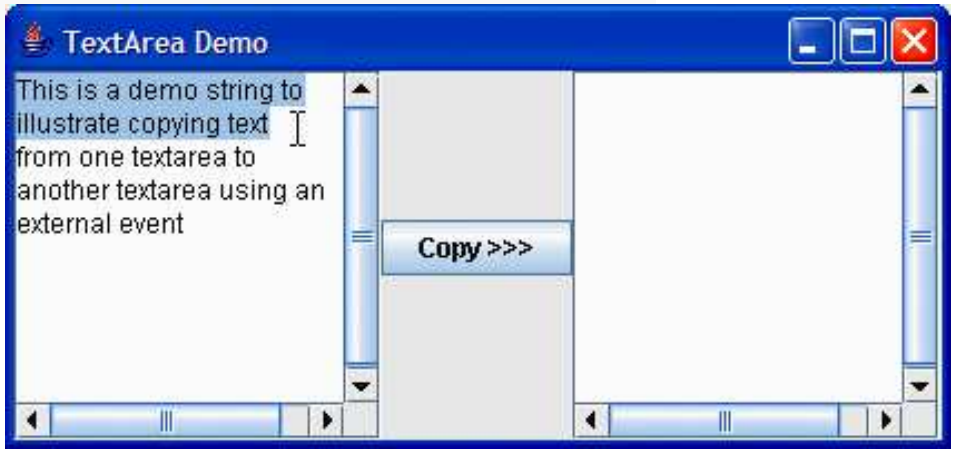
Adiciona o botão à caixa

Copia o texto selecionado de uma área de texto para outra

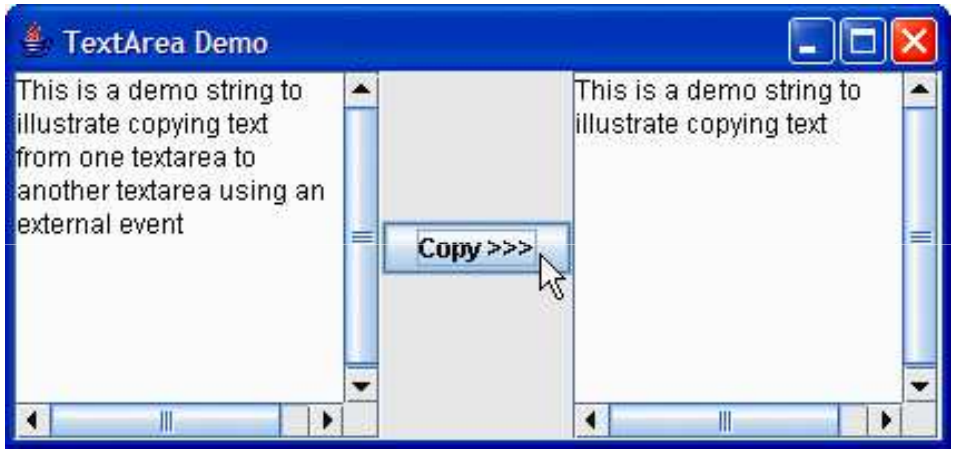
Cria uma segunda área de texto e a adiciona à caixa

Exemplo

```
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
```



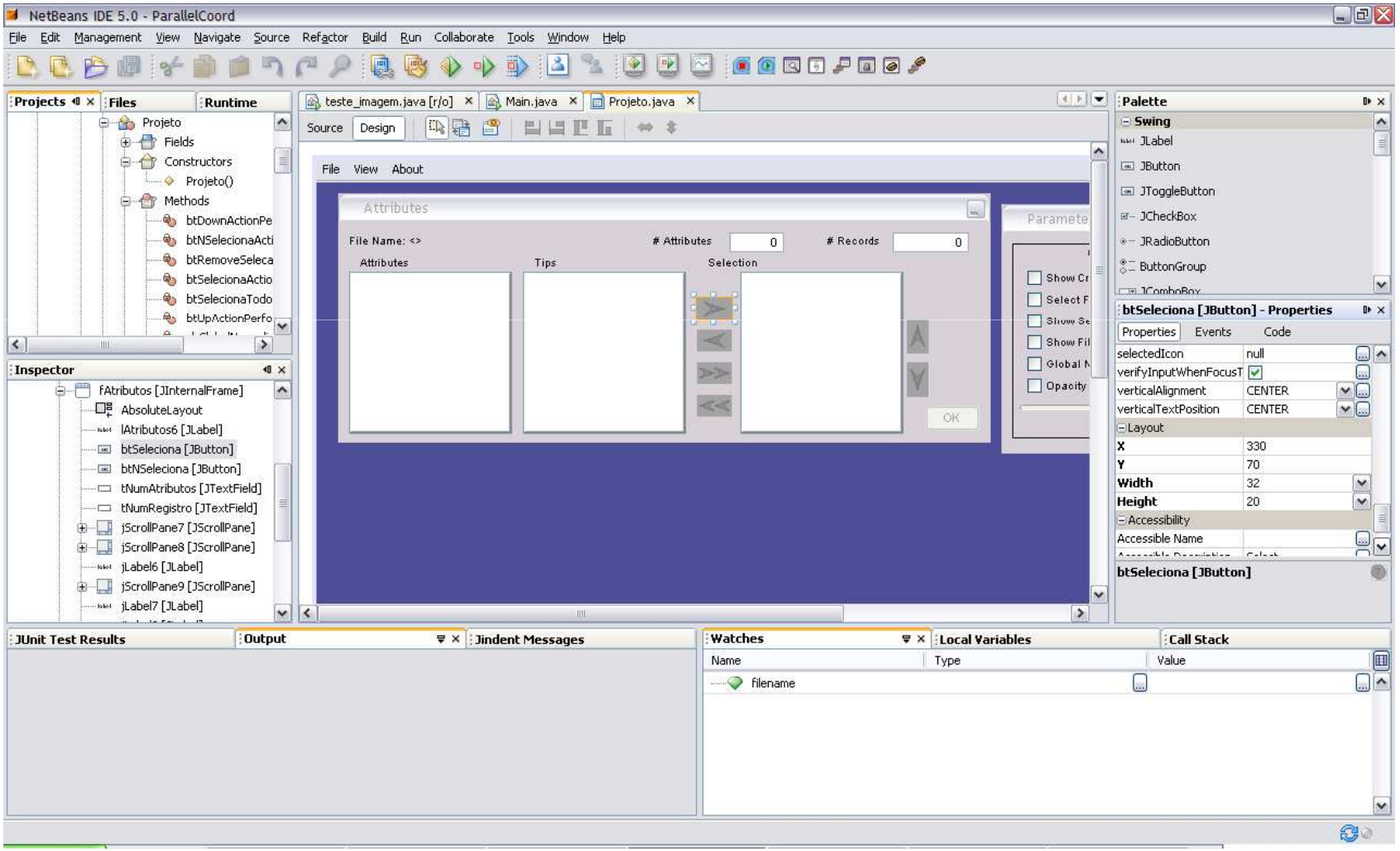
Exemplo


TextArea Demo
This is a demo string to illustrate copying text from one textarea to another textarea using an external event
Copy >>>
This is a demo string to illustrate copying text

Barra de Rolagem

- 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).

NetBeans IDE



Exercícios

- fazer todos os exemplos em um único projeto. A forma de acesso a cada um deles deverá ser realizada por um componente JMenu