

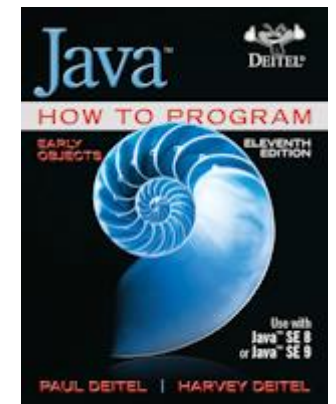
Interface Gráfica

Notas de Aula

Prof. André Bernardi

andrebernardi@unifei.edu.br

SWING - Uma interface gráfica consistente auxilia o usuário a aprender mais rápido a utilização de um programa





Componentes Básicos de uma Interface Gráfica

JLabel	Exibe um texto não editável e/ou ícones.
TextField	Caixa de Texto para usuário entrar com valores.
Button	Botão, usado para disparar um evento de ação quando for clicado pelo mouse.
CheckBox	Especifica uma opção que pode ser selecionada e desselecionada.



Componentes Básicos de uma Interface Gráfica

JComboBox	Caixa de combinação usada para selecionar elementos de uma lista drop-down ou digitar um texto na caixa de texto.
JList	Lista, usada para permitir o usuário selecionar um ou mais elementos através de cliques do mouse.
JPanel	Proporciona uma área onde podem ser acrescentados outros componentes ou usado como área de desenho.

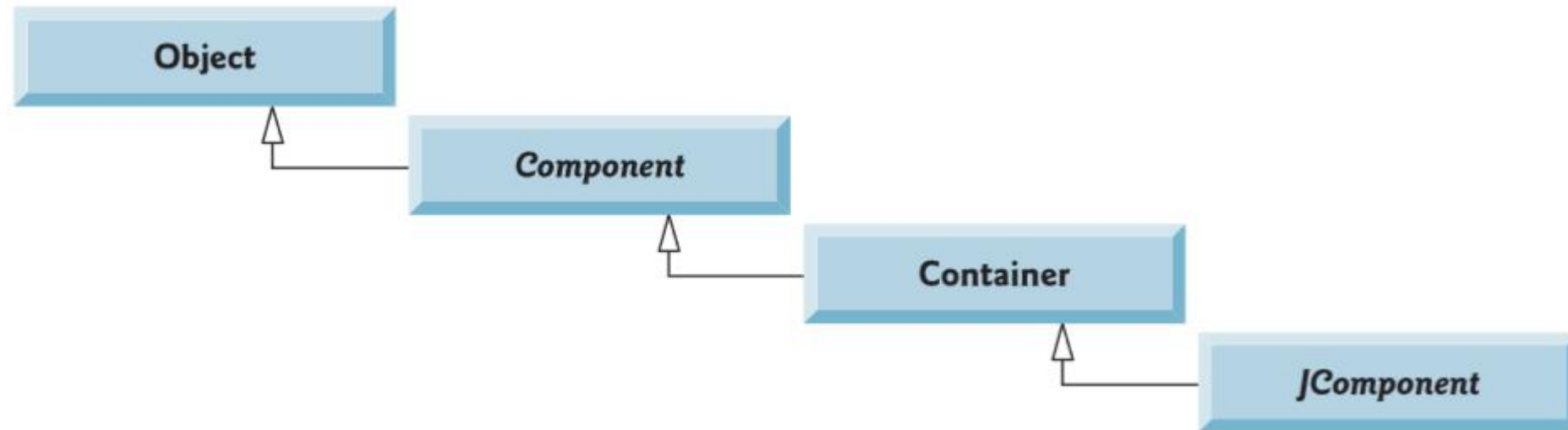


Componentes Básicos de uma Interface Gráfica

Componente	Descrição
JLabel	Exibe <i>texto</i> e/ou ícones <i>não editáveis</i> .
TextField	Normalmente <i>recebe entrada</i> do usuário.
Button	Dispara um evento quando o usuário clicar nele com o mouse.
CheckBox	Especifica uma opção que pode <i>ser</i> ou <i>não selecionada</i> .
ComboBox	Uma <i>lista drop-down dos itens</i> a partir dos quais o <i>usuário</i> pode fazer uma <i>seleção</i> .
List	Uma <i>lista dos itens</i> a partir dos quais o usuário pode fazer uma <i>seleção clicando</i> em <i>qualquer um</i> deles. <i>Múltiplos</i> elementos <i>podem</i> ser selecionados.
Panel	Uma área em que os <i>componentes</i> podem ser <i>colocados</i> e <i>organizados</i> .



Hierarquia dos componentes SWING





Exemplos de métodos da classe JComponent

- void **setToolTipText**(String text)
- Graphics **getGraphics**()
- int **getHeight**()
- String **getToolTipText**()
- int **getWidth**()

- void **paintComponent**(Graphics g)
- void **requestFocus**()
- void **setBackground**(Color bg)
- void **setFont**(Font font)
- void **setIcon**(Icon icone)
- void **setOpaque**(boolean isOpaque)



Classe **JLabel** - Construtores

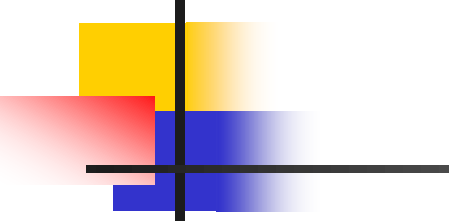
- `JLabel ()`
- `JLabel (Icon image)`
- `JLabel (Icon image, int horizontalAlignment)`
- `JLabel (String text)`
- `JLabel (String text, Icon icon, int horizontalAlignment)`
- `JLabel (String text, int horizontalAlignment)`



JLabel – Métodos úteis

- void setToolTipText(String str)
- int getHorizontalAlignment()
- int getHorizontalTextPosition()
- Icon getIcon()
- void setIcon(Icon icon)
- int getVerticalAlignment()
- int getVerticalTextPosition()
- void setHorizontalAlignment(int alignment)
- void setHorizontalTextPosition(int textPosition)
- void setText(String text)
- void setVerticalTextPosition(int textPosition)

Constantes para alinhamento



```
public abstract interface SwingConstants{  
    static int BOTTOM  
    static int CENTER  
    static int HORIZONTAL  
    static int LEFT  
    static int RIGHT  
    static int TOP  
    static int VERTICAL  
}
```

Constante	Descrição	Constante	Descrição
<i>Constantes de posição horizontal</i>		<i>Constantes de posição vertical</i>	
LEFT	Coloca o texto à esquerda	TOP	Coloca o texto na parte superior
CENTER	Coloca o texto no centro	CENTER	Coloca o texto no centro
RIGHT	Coloca o texto à direita	BOTTOM	Coloca o texto na parte inferior



Classe ImageIcon

Construtores:

- ImageIcon()
- ImageIcon(byte[] imageData)
- ImageIcon(byte[] imageData, String description)
- ImageIcon(Image image)
- ImageIcon(Image image, String description)
- ImageIcon(String filename)
- ImageIcon(String filename, String description)
- ImageIcon(URL location)
- ImageIcon(URL location, String description)



ImageIcon

Métodos:

- `int getIconHeight()`
- `int getIconWidth()`
- `Image getImage()`
- `void paintIcon(Component c, Graphics g, int x, int y)`
- `void setDescription(String description)`
- `void setImage(Image image)`

```

1 // Figura 12.6: LabelFrame.java
2 // JLabels com texto e ícones.
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 final JLabel label1; // JLabel apenas com texto
13     private final JLabel label2; // JLabel construído com texto e ícone
14     private final 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

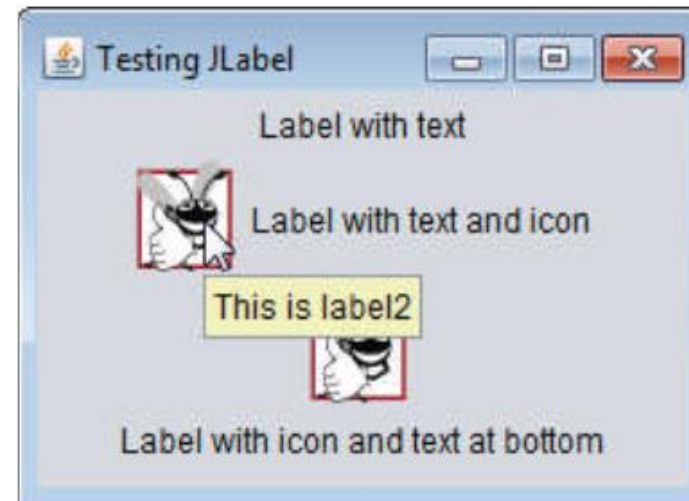
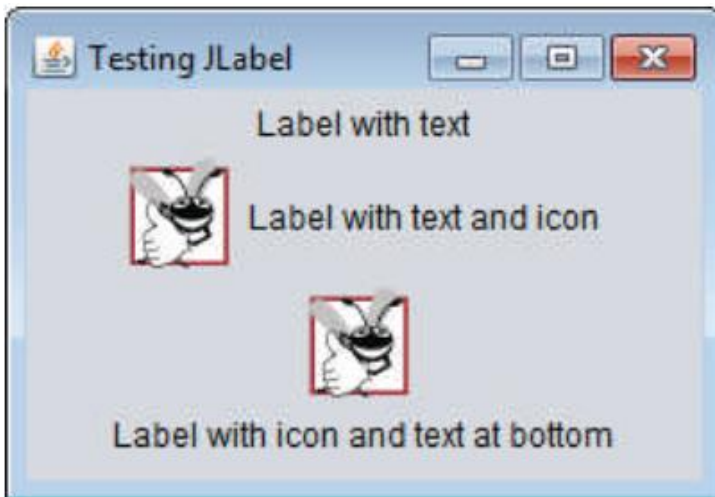
```

```

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 o label1 ao JFrame
26
27         // construtor JLabel com string, Icon e argumentos de alinhamento
28         Icon bug = new ImageIcon(getClass().getResource("bug1.png"));
29         label2 = new JLabel("Label with text and icon", bug,
30                             SwingConstants.LEFT);
31         label2.setToolTipText("This is label2");
32         add(label2);           // adiciona label2 ao 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 o ícone ao JLabel
37         label3.setHorizontalTextPosition(SwingConstants.CENTER);
38         label3.setVerticalTextPosition(SwingConstants.BOTTOM);
39         label3.setToolTipText("This is label3");
40         add(label3);           // adiciona label3 ao JFrame
41     }
42 } // fim da classe LabelFrame

```

```
1 // Figura 12.7: 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();
10        labelFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11        labelFrame.setSize(260, 180);
12        labelFrame.setVisible(true);
13    }
14 } // fim da classe LabelTest
```





Campo de Texto - Classe **JTextField**

Construtores:

- `JTextField()`
- `JTextField(int columns)`
- `JTextField(String text)`
- `JTextField(String text, int columns)`



Classe JTextField

Métodos úteis:

- `void addActionListener(ActionListener l)`
- `void removeActionListener(ActionListener l)`
- `int getColumns()`
- `int getHorizontalAlignment()`
- `void setColumns(int columns)`
- `void setEditable(boolean bEdit)`
- `void setFont(Font f)`
- `void setHorizontalAlignment(int alignment)`



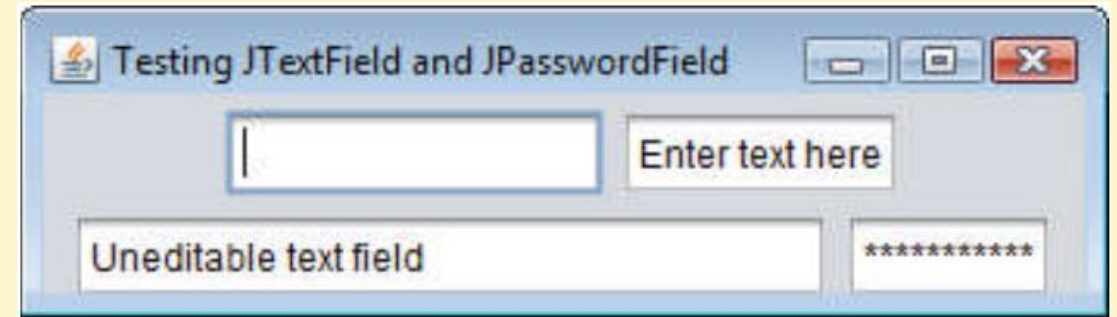
JPasswordField

- É derivada de JTextField
- Construtores:
 - JPasswordField()
 - JPasswordField(int columns)
 - JPasswordField(String text)
 - JPasswordField(String text, int columns)
- Métodos:
 - char getEchoChar()
 - char[] getPassword()
 - void setEchoChar(char c)

```

1 // Figura 12.9: TextFieldFrame.java
2 // JTextField e JPasswordField.
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 final JTextField textField1; // campo de texto com tamanho configurado
14     private final JTextField textField2; // campo de texto com texto
15     private final JTextField textField3; // campo de texto com texto e tamanho
16     private final 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());
23

```



```

24 // cria campo de texto com 10 colunas
25 textField1 = new JTextField(10);
26 add(textField1); // adiciona textField1 ao JFrame
27
28 // cria campo de texto com texto padrão
29 textField2 = new JTextField("Enter text here");
30 add(textField2); // adiciona textField2 ao JFrame
31
32 // cria campo de texto 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 // cria campo de senha com texto padrão
38 passwordField = new JPasswordField("Hidden text");
39 add(passwordField); // adiciona passwordField ao JFrame
40
41 // rotinas de tratamento de evento registradoras
42 TextFieldHandler handler = new TextFieldHandler();
43 textField1.addActionListener(handler);
44 textField2.addActionListener(handler);
45 textField3.addActionListener(handler);
46 passwordField.addActionListener(handler);
47 }

```

```
49 // classe interna private para tratamento de evento
50 private class TextFieldHandler implements ActionListener
51 {
52     // processa eventos de campo de texto
53     @Override
54     public void actionPerformed(ActionEvent event)
55     {
56         String string = "";
57
58         // usuário pressionou Enter no JTextField textField1
59         if (event.getSource() == textField1)
60             string = String.format("textField1: %s",
61                                     event.getActionCommand());
62
63         // usuário pressionou Enter no JTextField textField2
64         else if (event.getSource() == textField2)
65             string = String.format("textField2: %s",
66                                     event.getActionCommand());
67
68         // usuário pressionou Enter no JTextField textField3
69         else if (event.getSource() == textField3)
70             string = String.format("textField3: %s",
71                                     event.getActionCommand());
72     }
```

```

73         // usuário pressionou Enter no JTextField passwordField
74     else if (event.getSource() == passwordField)
75         string = String.format("passwordField: %s",
76                                 event.getActionCommand());
77
78         // exibe o conteúdo de JTextField
79         JOptionPane.showMessageDialog(null, string);
80     }
81 } // fim da classe TextFieldHandler interna private
82 } // fim da classe TextFieldFrame

```

1 // Figura 12.10: 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();

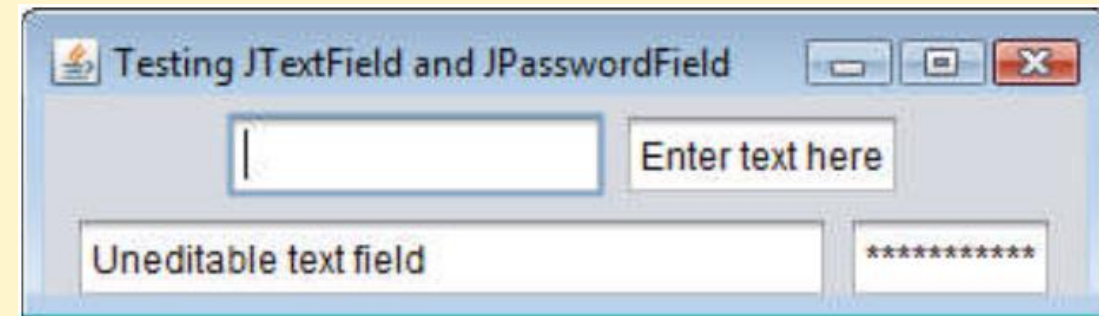
10 textFieldFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

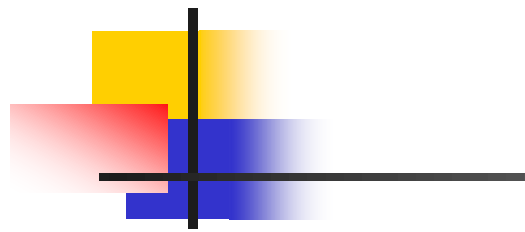
11 textFieldFrame.setSize(350, 100);

12 textFieldFrame.setVisible(true);

13 }

14 } // fim da classe TextFieldTest





Testing JTextField and JPasswordField

hello Enter text here

Uneditable text field *****

Message

textField1: hello

OK

Testing JTextField and JPasswordField

hello Enter text here

Uneditable text field *****

Message

textField2: Enter text here

OK

Testing JTextField and JPasswordField

hello Enter text here

Uneditable text field *****

Message

textField3: Uneditable text field

OK

Testing JTextField and JPasswordField

hello Enter text here

Uneditable text field *****

Message

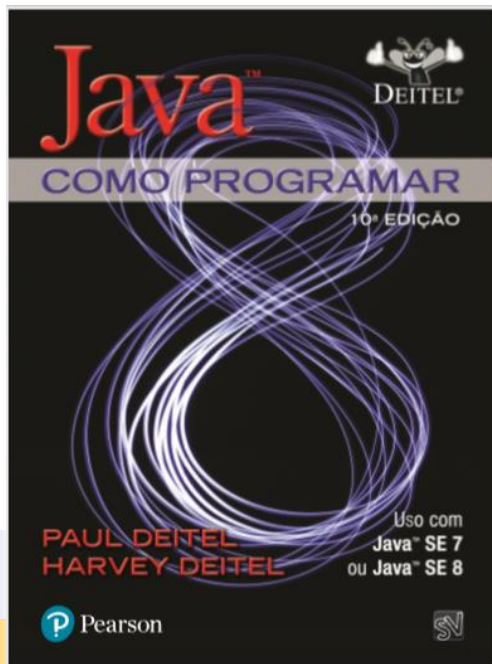
passwordField: Hidden text

OK



Exercício

- 1)** Escreva um programa utilizando janela em Java para criar um gráfico de pizza dados cinco valores obtidos em caixas de texto. Apresente um texto com a porcentagem de cada uma das cinco entradas disponíveis para o usuário sobre os arcos do gráfico de pizza.



Tipos comuns de eventos

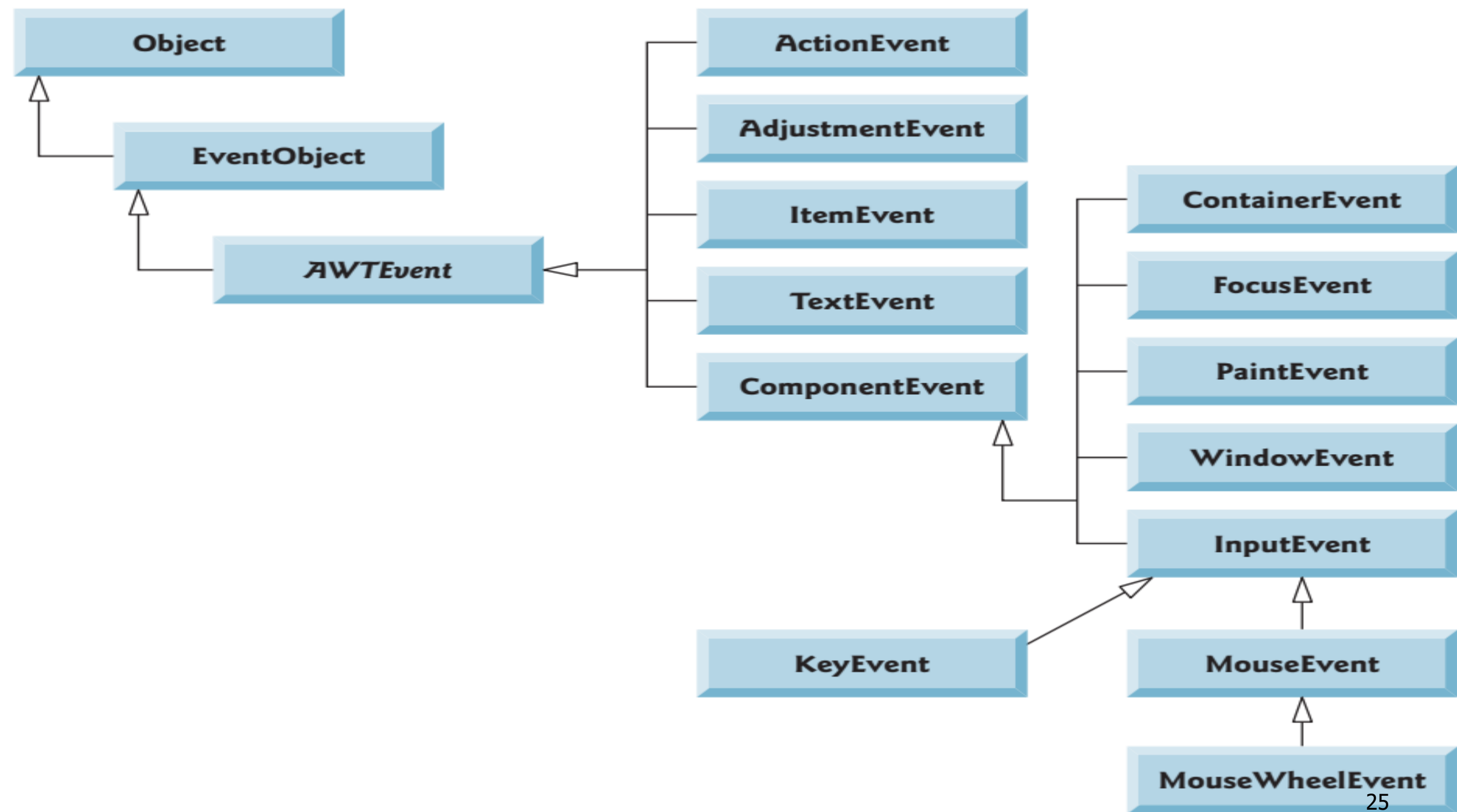
Interfaces Ouvintes

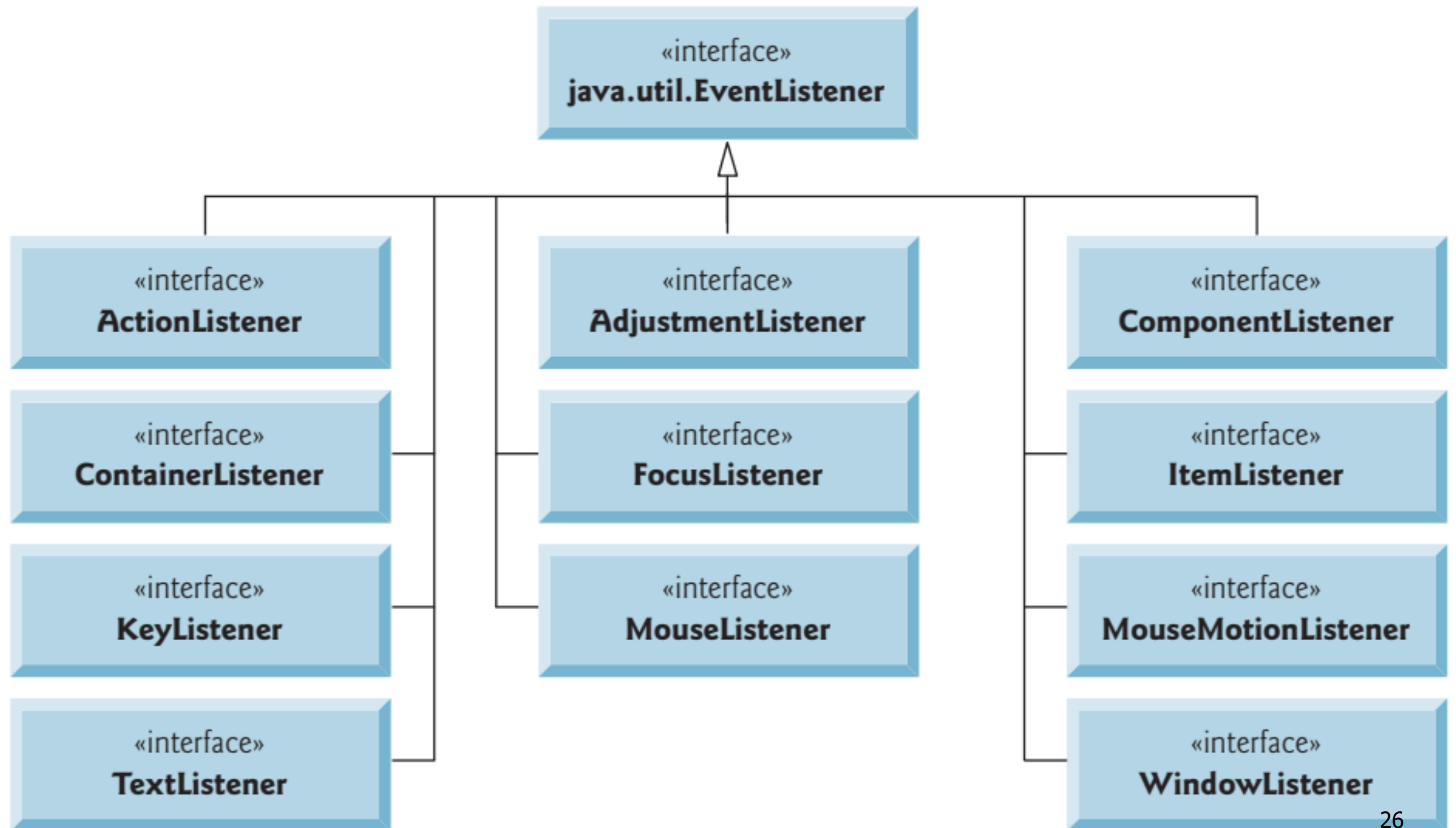
Notas de Aula

Prof. André Bernardi

andrebernardi@unifei.edu.br









Modelo de manipulação de eventos

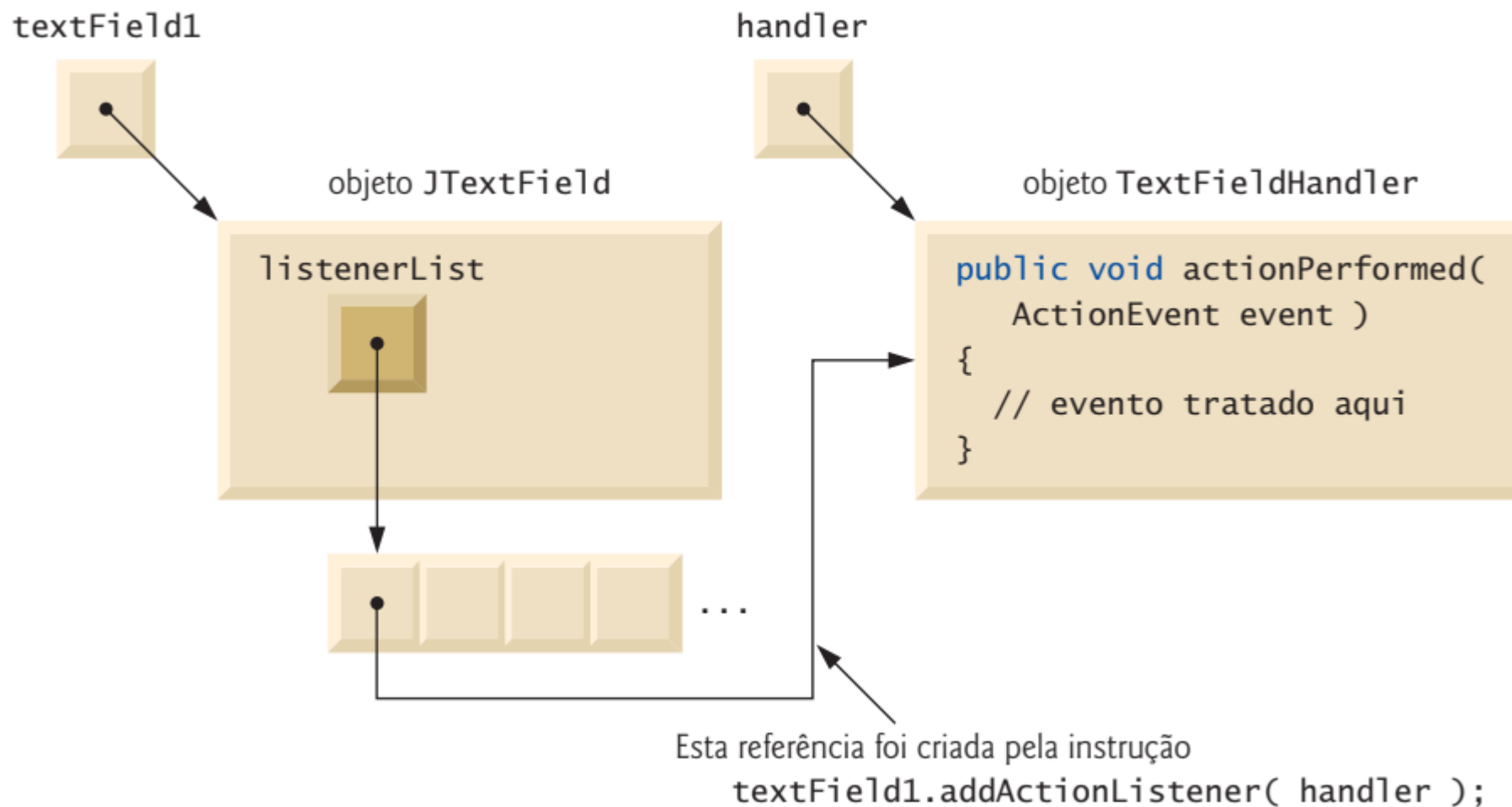
- A manipulação de eventos da GUI, engloba duas tarefas chaves: **registrar** um ***event-listener*** e **implementar** um ***event-handler***.
- **Event-Listener** – Objeto de uma classe que implementa uma ou mais interfaces espera por um evento específico.
- **Event-Handler** – Método que é chamado automaticamente quando um evento ocorre.



Eventos

- Cada interface *listener* especifica um ou mais métodos para manipulação de eventos. Sendo que eles devem ser definidos nas classes que implementam as interfaces.
- Não esquecer de registrar os eventos
- Usar sempre que possível, classes separadas para processar os eventos.

Como funcionam os eventos





Classes Internas

- A linguagem Java permite que sejam declaradas classes privadas **dentro** de classes públicas.
- Podem ser declaradas de duas maneiras:
 - **Anônimas** – geralmente usada como parâmetro de método
 - **Completa** – encapsulada na classe
- Todas classes internas podem **derivar** de outras classes e **implementar** interfaces



Sintaxe Classe Interna **Completa**

```
public class Externa{  
    ....// variáveis membros  
    ....// métodos  
    private class Interna extends Base  
        implements Interfaces  
    {  
        // variáveis e métodos da classe interna  
    }  
} // fim da classe Externa
```



Sintaxe Classe Interna Anônima

```
// deve ficar dentro de um método que pertence  
a uma classe
```

```
texto.addActionListener( new ActionListener () {  
    public void actionPerformed(ActionEvent e) {  
        // rotina para tratar o evento  
    } // fim da actionPerformed  
} // fim da Classe interna  
); // termino da lista de parâmetros da função
```




Timer

- Classe **Timer** cria um contador que uma vez iniciado gera um evento de ação em intervalos de tempo regulares.
- Exemplo

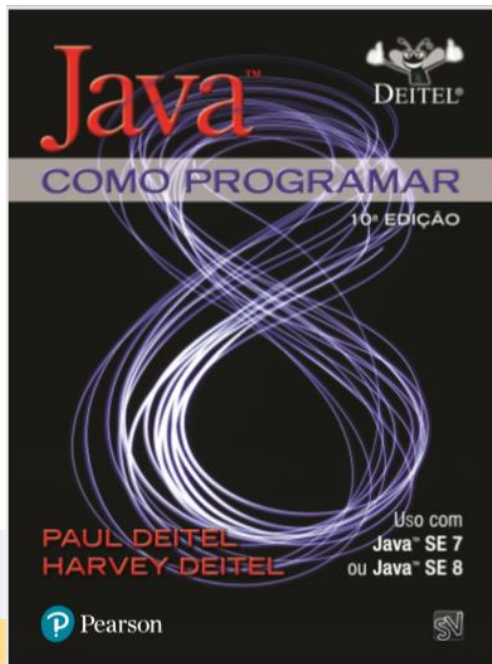
```
Timer t = new Timer (1000, new ActionListener()  
{  
    public void actionPerformed(ActionEvent e)  
    {    repaint(); }  
} );  
t.start();
```



Exercício

1)Escreva um programa utilizando janela em Java para criar um pseudo- protetor de tela, que desenha formas (GeneralPath) em posições aleatórias na tela e após um numero X de desenhos a tela deve ser apagada. Permita que cada vez que a tela for apagada ela seja desenhada de uma cor diferente.

Sugestão: Utilize um timer para gerenciar a impressão das formas.



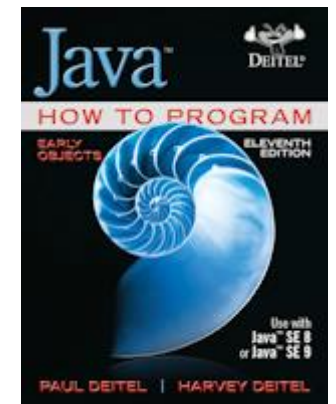
Botões

JButton, JCheckBox, JRadioButton

Notas de Aula

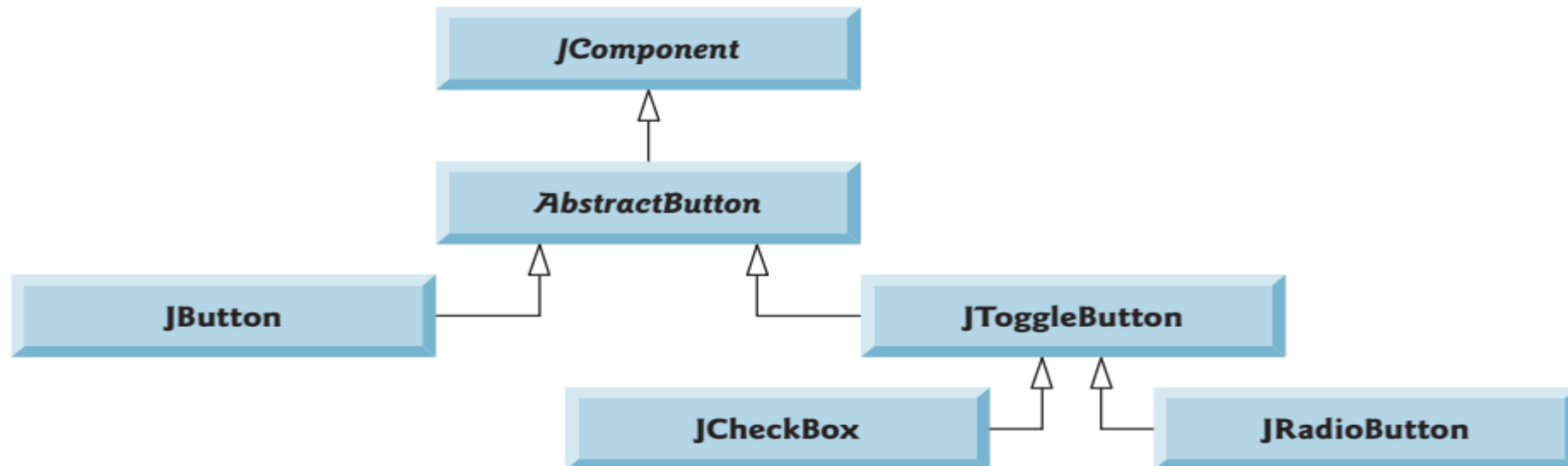
Prof. André Bernardi

andrebernardi@unifei.edu.br



Classe **JButton** - Construtores

- JButton()
- JButton(Icon icon)
- JButton(String text)
- JButton(String text, Icon icon)





JButton - Métodos

- `void setDisabledIcon(Icon disabledIcon)`
- `void setDisabledSelectedIcon(Icon disabledSelectedIcon)`
- `void setEnabled(boolean b)`
- `void setHorizontalAlignment(int alignment)`
- `void setHorizontalTextPosition(int textPosition)`
- `void setIcon(Icon defaultIcon)`
- `void setPressedIcon(Icon pressedIcon)`



JButton - Métodos

- void **setRolloverEnabled**(boolean b)
- void **setRolloverIcon**(Icon rolloverIcon)
- void **setRolloverSelectedIcon**(Icon rolloverSelectedIcon)
- void **setSelected**(boolean b)
- void **setSelectedIcon**(Icon selectedIcon)
- void **setText**(String text)
- void **setVerticalAlignment**(int alignment)
- void **setVerticalTextPosition**(int textPosition)

```

1 // Figura 12.15: ButtonFrame.java
2 // Botões de comando e eventos de ação.
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 final JButton plainJButton; // botão apenas com texto
15     private final JButton fancyJButton; // botão com ícones
16
17     // ButtonFrame adiciona JButtons ao JFrame
18     public ButtonFrame()
19     {
20         super("Testing Buttons");
21         setLayout(new FlowLayout());
22

```



```

23 plainJButton = new JButton("Plain Button"); // botão com texto
24 add(plainJButton); // adiciona plainJButton ao JFrame
25
26 Icon bug1 = new ImageIcon(getClass().getResource("bug1.gif"));
27 Icon bug2 = new ImageIcon(getClass().getResource("bug2.gif"));
28 fancyJButton = new JButton("Fancy Button", bug1); // configura imagem
29 fancyJButton.setRolloverIcon(bug2); // configura imagem de rollover
30 add(fancyJButton); // adiciona fancyJButton ao JFrame
31
32 // cria novo ButtonHandler de tratamento para tratamento de evento de botão
33 ButtonHandler handler = new ButtonHandler();
34 fancyJButton.addActionListener(handler);
35 plainJButton.addActionListener(handler);
36 }
37 // classe interna para tratamento de evento de botão
38 private class ButtonHandler implements ActionListener
39 {
40     public void actionPerformed(ActionEvent event) // trata evento de botão
41     {
42         JOptionPane.showMessageDialog(ButtonFrame.this, String.format(
43             "You pressed: %s", event.getActionCommand()));
44     } } // fim da classe ButtonFrame

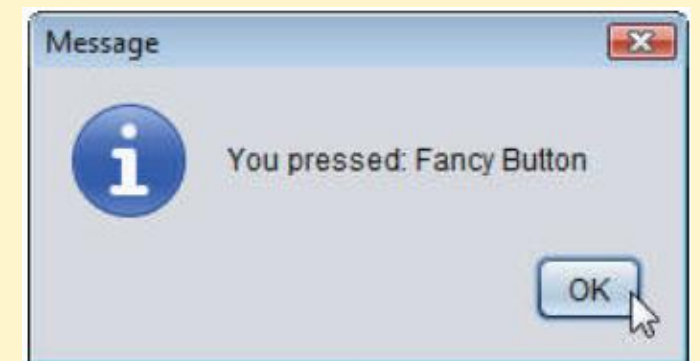
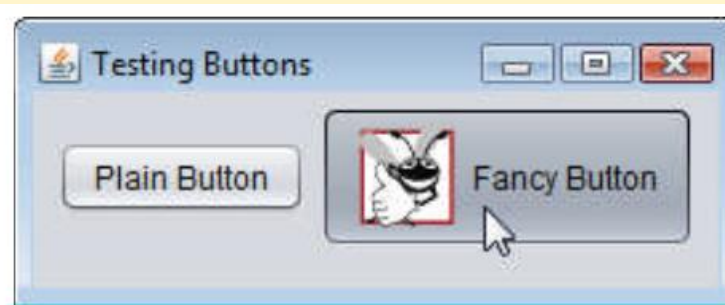
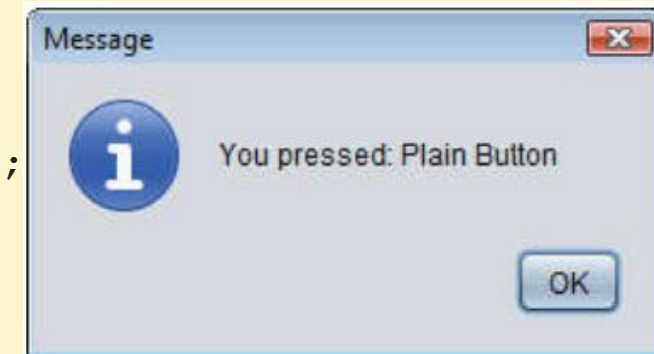
```



```

1 // Figura 12.16: 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();
10        buttonFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11        buttonFrame.setSize(275, 110);
12        buttonFrame.setVisible(true);
13    }
14 } // fim da classe ButtonTest

```





Classe **JCheckBox** - Construtores

- JCheckBox()
- JCheckBox(Icon icon)
- JCheckBox(Icon icon, boolean selected)
- JCheckBox(String text)
- JCheckBox(String text, boolean selected)
- JCheckBox(String text, Icon icon)
- JCheckBox(String text, Icon icon, boolean selected)

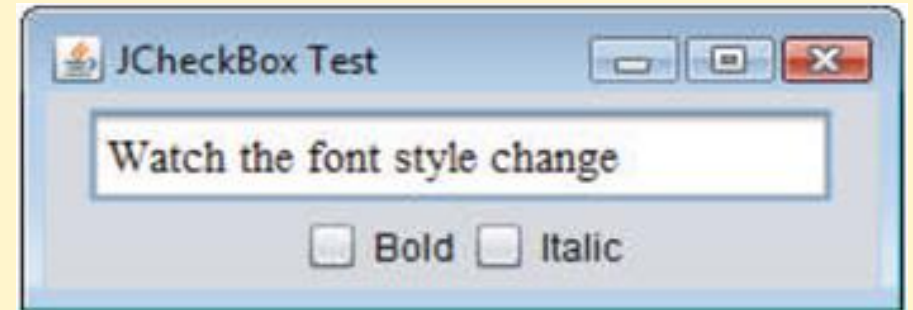


Classe **JCheckBox** - Método

Além do evento de **ação**, essa classe também suporta eventos de **item**.

- addItemListener(handle)
 - onde handle é um objeto que implementa a interface

```
1 // Figura 12.17: CheckBoxFrame.java
2 // JCheckBoxes e eventos de item.
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 final JTextField textField; // exibe o texto na alteração de fontes
14     private final JCheckBox boldJCheckBox; // para marcar/desmarcar estilo negrito
15     private final JCheckBox italicJCheckBox; // para marcar/desmarcar estilo itálico
16
17     // construtor CheckBoxFrame adiciona JCheckBoxes ao JFrame
18     public CheckBoxFrame()
19     {
20         super("JCheckBox Test");
21         setLayout(new FlowLayout());
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 ao JFrame
27
28 boldJCheckBox = new JCheckBox("Bold");
29 italicJCheckBox = new JCheckBox("Italic");
30 add(boldJCheckBox); // adiciona cxa de seleção de estilo negrito ao JFrame
31 add(italicJCheckBox); // adiciona caixa de seleção de itálico ao JFrame
32
33 // listeners registradores para JCheckBoxes
34 CheckBoxHandler handler = new CheckBoxHandler();
35 boldJCheckBox.addItemListener(handler);
36 italicJCheckBox.addItemListener(handler);
37 }
38
```

```

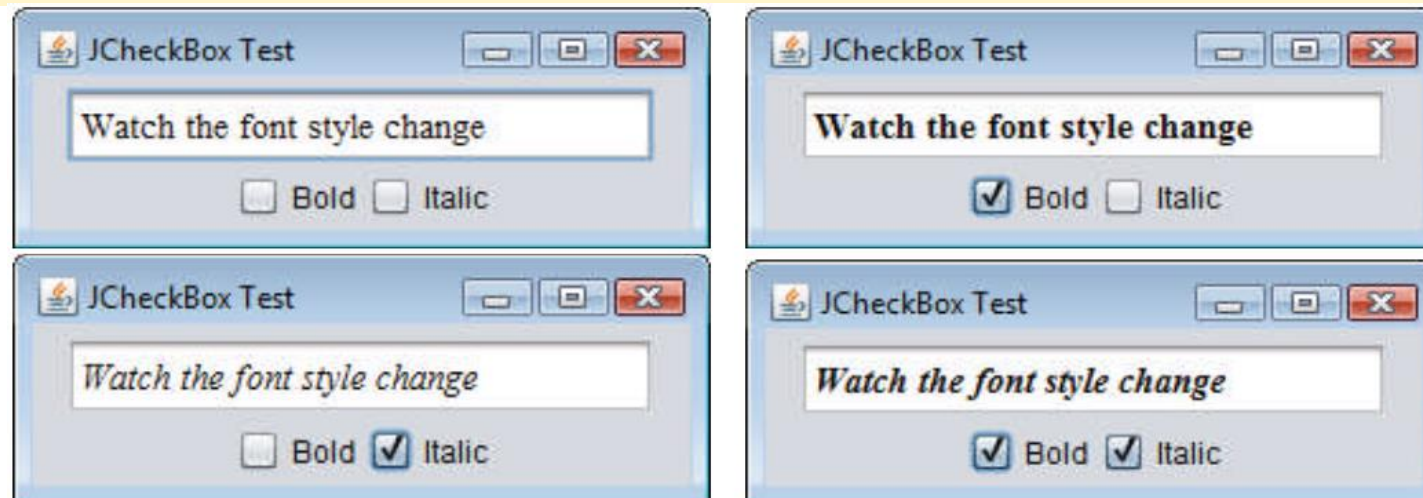
39 // classe interna private para tratamento de evento ItemListener
40 private class CheckBoxHandler implements ItemListener
41 {
42     // responde aos eventos de caixa de seleção
43     @Override
44     public void itemStateChanged(ItemEvent event)
45     {
46         Font font = null; // armazena a nova Font
47
48         // determina quais CheckBoxes estão marcadas e cria Font
49         if (boldJCheckBox.isSelected() && italicJCheckBox.isSelected())
50             font = new Font("Serif", Font.BOLD + Font.ITALIC, 14);
51         else if (boldJCheckBox.isSelected())
52             font = new Font("Serif", Font.BOLD, 14);
53         else if (italicJCheckBox.isSelected())
54             font = new Font("Serif", Font.ITALIC, 14);
55         else
56             font = new Font("Serif", Font.PLAIN, 14);
57
58         textField.setFont(font);
59     }
60 }

```

```

1 // Figura 12.18: 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);
12        checkBoxFrame.setVisible(true);
13    }
14 } // fim da classe CheckBoxTest

```





Classe **JRadioButton** - Construtores

- `JRadioButton()`
- `JRadioButton(Icon icon)`
- `JRadioButton(Icon icon, boolean selected)`
- `JRadioButton(String text)`
- `JRadioButton(String text, boolean selected)`
- `JRadioButton(String text, Icon icon)`
- `JRadioButton(String text, Icon icon, boolean selected)`



Classe **ButtonGroup**

- Construtor:
 - ButtonGroup()
- Métodos:
 - void add(AbstractButton b)
 - Enumeration getElements()
 - ButtonModel getSelection()
 - boolean isSelected(ButtonModel m)
 - void remove(AbstractButton b)
 - void setSelected(ButtonModel m, boolean b)



Interface **ItemListener**

Método:

- `void itemStateChanged(ItemEvent e)`
 - Chamada quando um item é selecionado ou desselecionado.



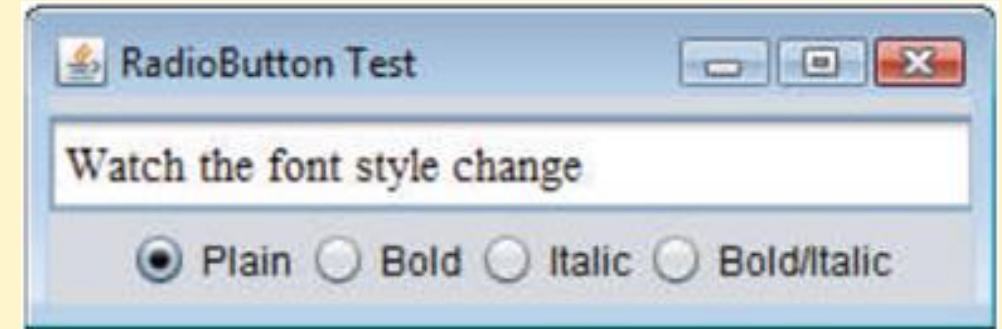
Classe **ItemEvent**

- static int DESELECTED
- static int SELECTED
- Object getItem()
 - Retorna o item afetado pelo evento.
- ItemSelectable getItemSelectable()
 - Retorna o item que originou o evento.
- int getStateChange()
 - Retorna o tipo de estado alterado (*selected* ou *deselected*).
- String paramString()
 - Retorna uma string identificando o evento de item.

```

1 // Figura 12.19: 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 final JTextField textField; // utilizado para exibir alterações de fonte
15     private final Font plainFont;      // fonte para texto simples
16     private final Font boldFont;       // fonte para texto em negrito
17     private final Font italicFont;     // fonte para texto em itálico
18     private final Font boldItalicFont; // fonte para texto em negrito e itálico
19     private final JRadioButton plainJRadioButton; // seleciona texto simples
20     private final JRadioButton boldJRadioButton; // seleciona texto em negrito
21     private final JRadioButton italicJRadioButton; // seleciona texto em itálico
22     private final JRadioButton boldItalicJRadioButton; // negrito e itálico
23     private final ButtonGroup radioGroup; // contém botões de opção
24

```



```

25 // construtor RadioButtonFrame adiciona JRadioButtons ao JFrame
26 public RadioButtonFrame()
27 {
28     super("RadioButton Test");
29     setLayout(new FlowLayout());
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", false);
38     boldItalicJRadioButton = new JRadioButton("Bold/Italic", false);
39     add(plainJRadioButton); // adiciona botão no estilo 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 negrito e itálico
43
44     // cria relacionamento lógico entre JRadioButtons
45     radioGroup = new ButtonGroup(); // cria ButtonGroup
46     radioGroup.add(plainJRadioButton); // adiciona texto 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

```

```
51 // cria fonte objetos
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);
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 }
68
69 // classe interna private para tratar eventos de botão de opção
70 private class RadioButtonHandler implements ItemListener
71 {
72     private Font font; // fonte associada com esse listener
73
```

```

74     public RadioButtonHandler(Font f)
75     {
76         font = f;
77     }
78     // trata eventos de botão de opção
79     @Override
80     public void itemStateChanged(ItemEvent event)
81     {
82         textField.setFont(font);
83     } // fim da classe RadioButtonHandler
84 } // fim da classe RadioButtonFrame

```

```

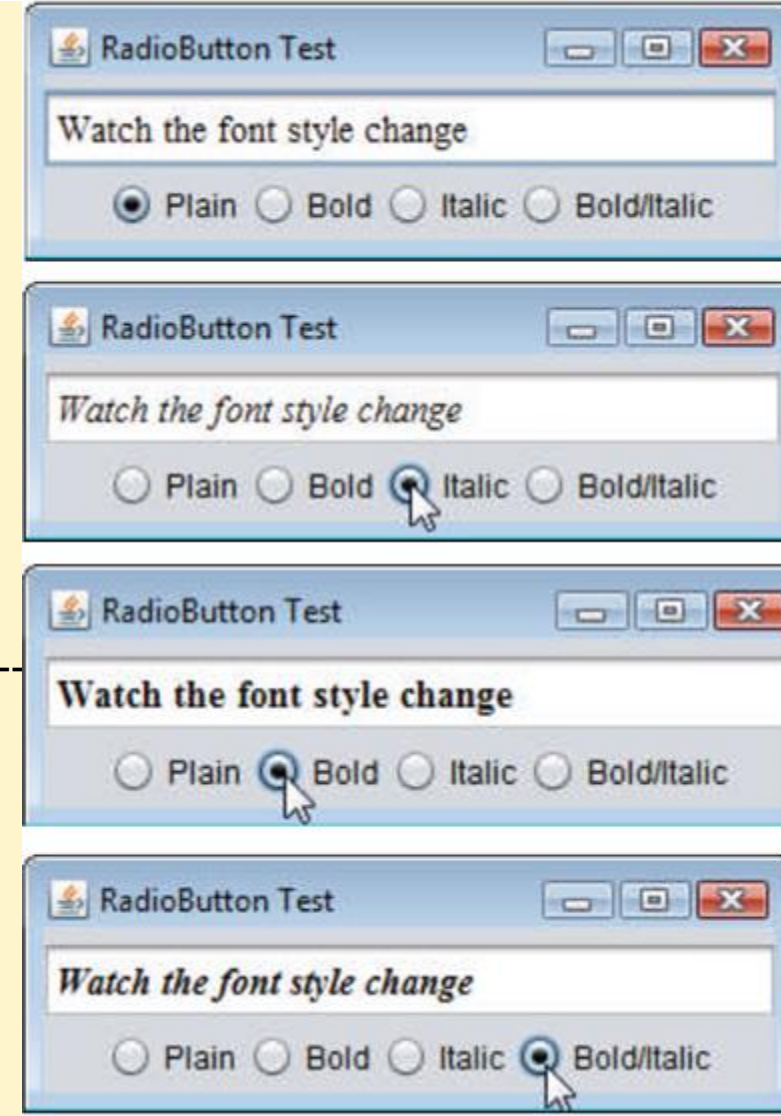
1 // Figura 12.20: RadioButtonTest.java
2 // Testando RadioButtonFrame.

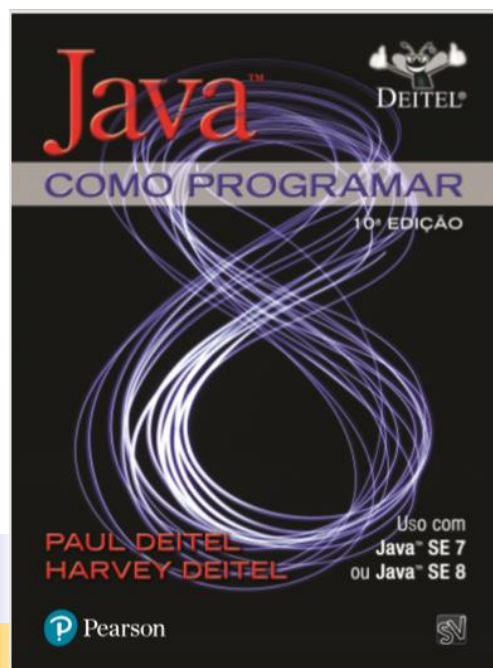
```

```

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);
12        radioButtonFrame.setVisible(true);
13    } // fim da classe RadioButtonTest

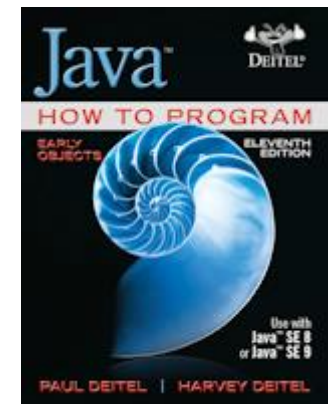
```





Eventos de Mouse e Teclado

Notas de Aula
Prof. André Bernardi
andrebernardi@unifei.edu.br





Manipulando Eventos do **Mouse**

- As interfaces
 - **MouseListener**
 - **MouseMotionListener**
- São utilizadas para tratar os eventos do mouse.
- Possuem a definição dos seguintes métodos:



MouseListener

- public void **mousePressed**(MouseEvent e)
 - Botão do mouse pressionado sobre um componente.
- public void **mouseClicked**(MouseEvent e)
 - Botão do mouse pressionado e liberado sobre um componente sem o movimento do mouse.
- public void **mouseReleased**(MouseEvent e)
 - Botão do mouse liberado sobre um componente



MouseListener

- `public void mouseEntered(MouseEvent e)`
 - Ponteiro do mouse entra dentro dos limites de um componente.
- `public void mouseExited(MouseEvent e)`
 - Ponteiro do mouse sai de dentro dos limites de um componente.
- Os 5 métodos descritos são chamados automaticamente quando ocorrer o evento correspondente no Componente registrado.
 - `addMouseListener();`



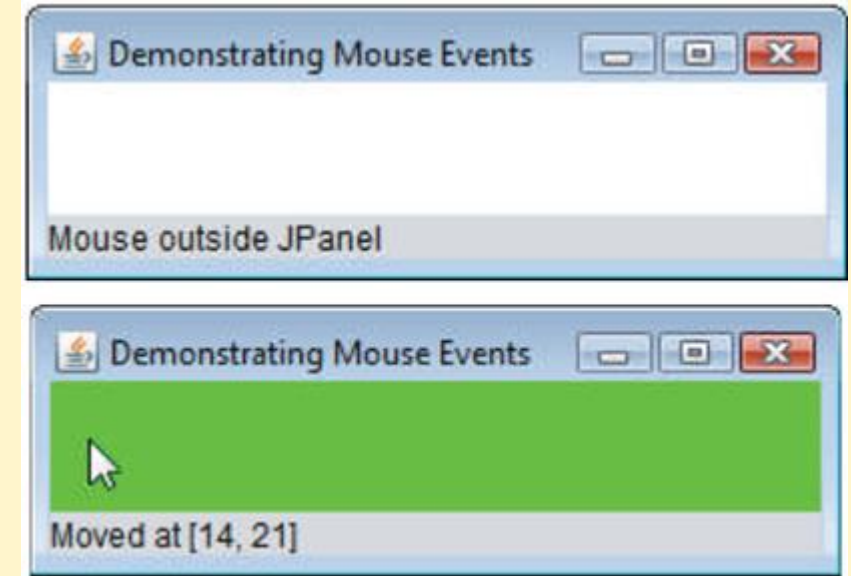
MouseMotionListener

- `public void mouseDragged(MouseEvent e)`
 - Botão do mouse é mantido pressionado e ponteiro movido de lugar.
 - `public void mouseMoved(MouseEvent e)`
 - Ponteiro do mouse é movido com o cursor sobre um componente.
-
- Os 2 métodos descritos são chamados automaticamente quando ocorrer o evento correspondente no Componente registrado.
 - `addMouseMotionListener();`

```

1 // Figura 12.28: MouseTrackerFrame.java
2 // Tratamento de evento de mouse.
3 import java.awt.Color;
4 import java.awt.BorderLayout;
5 import java.awt.event.MouseListener;
6 import java.awt.event.MouseMotionListener;
7 import java.awt.event.MouseEvent;
8 import javax.swing.JFrame;
9 import javax.swing.JLabel;
10 import javax.swing.JPanel;
11
12 public class MouseTrackerFrame extends JFrame
13 {
14     private final JPanel mousePanel; // painel em que os eventos de mouse ocorrem
15     private final JLabel statusBar; // exibe informações do evento
16
17     // construtor MouseTrackerFrame configura GUI e
18     // registra rotinas de tratamento de evento de mouse
19     public MouseTrackerFrame()
20     {
21         super("Demonstrating Mouse Events");
22
23         mousePanel = new JPanel();
24         mousePanel.setBackground(Color.WHITE);
25         add(mousePanel, BorderLayout.CENTER); // adiciona painel ao JFrame

```



```

27     statusBar = new JLabel("Mouse outside JPanel");
28     add(statusBar, BorderLayout.SOUTH); // adiciona rótulo ao JFrame
29
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 }
35
36 private class MouseHandler implements MouseListener,
37     MouseMotionListener
38 {
39     // rotinas de tratamento de evento MouseListener
40     // evento quando o mouse é liberado imediatamente depois de ter sido pressionado
41     @Override
42     public void mouseClicked(MouseEvent event) {
43         statusBar.setText(String.format("Clicked at [%d, %d]",
44             event.getX(), event.getY()));
45     }
46
47     // trata evento quando mouse é pressionado
48     @Override
49     public void mousePressed(MouseEvent event)
50     {

```

```
52         statusBar.setText(String.format("Pressed at [%d, %d]",
53                                         event.getX(), event.getY()));
54     }
55
56     // trata o evento quando o mouse é liberado
57     @Override
58     public void mouseReleased(MouseEvent event)
59     {
60         statusBar.setText(String.format("Released at [%d, %d]",
61                                         event.getX(), event.getY()));
62     }
63
64     // trata evento quando mouse entra na área
65     @Override
66     public void mouseEntered(MouseEvent event)
67     {
68         statusBar.setText(String.format("Mouse entered at [%d, %d]",
69                                         event.getX(), event.getY()));
70         mousePanel.setBackground(Color.GREEN);
71     }
72
73     // trata evento quando mouse sai da área
74     @Override
75     public void mouseExited(MouseEvent event)
76     {
```

```

77         statusBar.setText("Mouse outside JPanel");
78         mousePanel.setBackground(Color.WHITE);
79     }
80
81     // rotinas de tratamento de evento MouseMotionListener
82     // trata o evento quando o usuário arrasta o mouse com o botão pressionado
83     @Override
84     public void mouseDragged(MouseEvent event)
85     {
86         statusBar.setText(String.format("Dragged at [%d, %d]",
87                                         event.getX(), event.getY()));
88     }
89
90     // trata evento quando usuário move o mouse
91     @Override
92     public void mouseMoved(MouseEvent event)
93     {
94         statusBar.setText(String.format("Moved at [%d, %d]",
95                                         event.getX(), event.getY()));
96     }
97 } // fim da classe interna MouseHandler
98 } // fim da classe MouseTrackerFrame

```



```
1 // Figura 12.29: 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();
```

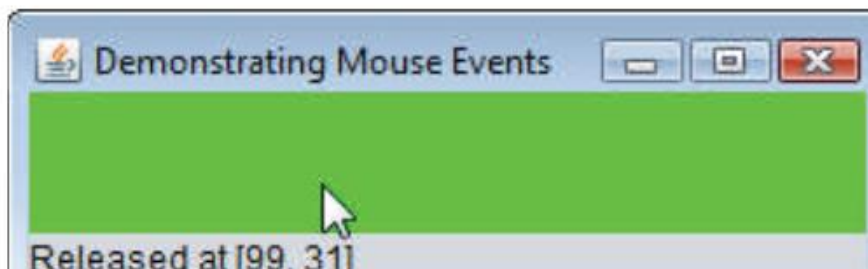
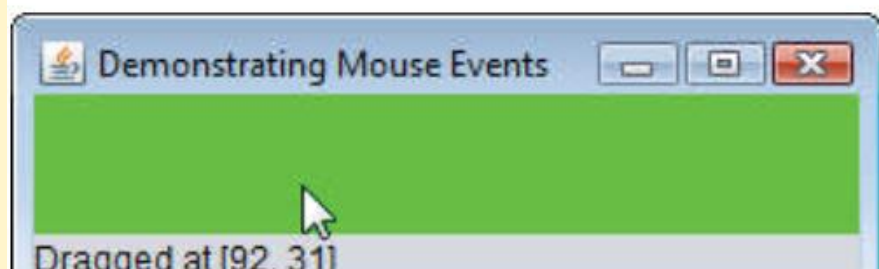
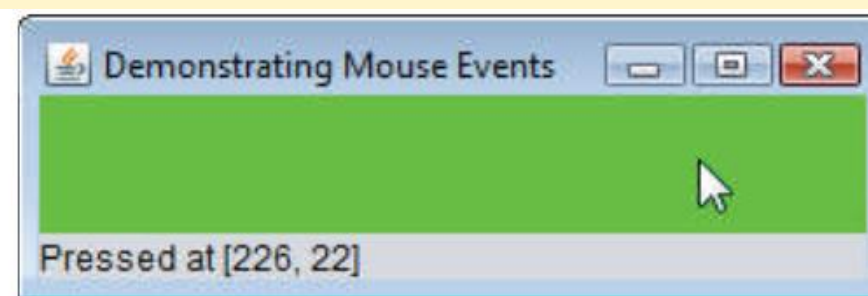
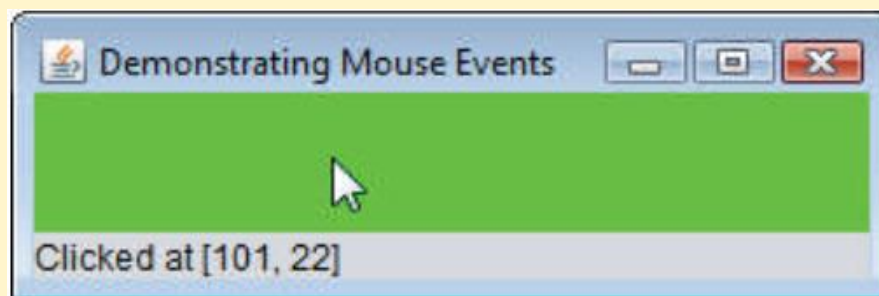
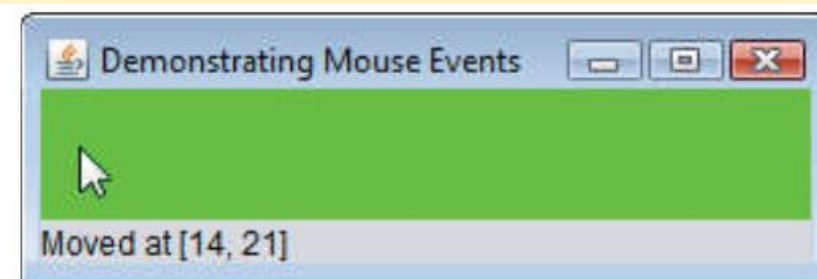
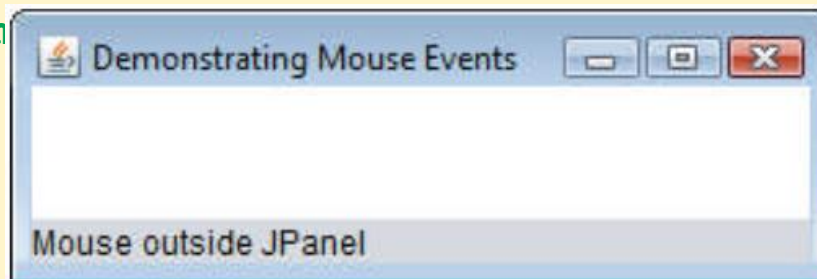
```
10        mouseTrackerFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
11        mouseTrackerFrame.setSize(300, 100);
```

```
12        mouseTrackerFrame.setVisible(true);
```

```
13    }
```

```
14 } // fim da classe MouseTracker
```





Classes de Adaptadores

Uma **classe de adaptadores** implementa uma interface e fornece uma implementação padrão (com o corpo de um método vazio) de cada método na interface.

Classe de adaptadores de evento em `java.awt.event`

Implementa a interface

ComponentAdapter

ComponentListener

ContainerAdapter

ContainerListener

FocusAdapter

FocusListener

KeyAdapter

KeyListener

MouseAdapter

MouseListener

MouseMotionAdapter

MouseMotionListener

WindowAdapter

WindowListener



Classes de Adaptadores

- Quando uma classe implementar uma interface, a classe tem um relacionamento **é um** com essa interface. Todas as subclasses diretas e indiretas dessa classe herdam essa interface. Portanto, um objeto de uma classe que **estende** uma classe de adaptadores de evento é um objeto do tipo *ouvinte de eventos* correspondente (por exemplo, um objeto de uma subclasse de ***MouseAdapter*** **é um** ***MouseListener***).
- Se você estender uma classe de adaptadores e **digitar incorretamente o nome ou assinatura** do método que está sendo sobrescrito, e se o método *@Override* não for declarado, seu método simplesmente se tornará outro método na classe. Esse é um **erro de lógica** difícil de ser detectado, visto que o programa chamará a versão vazia do método herdado da classe de adaptadores.

```

1 // Figura 12.31: MouseDetailsFrame.java
2 // Demonstrando cliques de mouse e distinguindo entre botões do mouse.
3 import java.awt.BorderLayout;
4 import java.awt.event.MouseAdapter;
5 import java.awt.event.MouseEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JLabel;
8
9 public class MouseDetailsFrame extends JFrame
10 {
11     private String details; // String exibida na statusBar
12     private final JLabel statusBar; // JLabel na parte inferior da janela
13
14     // construtor configura String de barra de título e registra o listener de mouse
15     public MouseDetailsFrame()
16     {
17         super("Mouse clicks and buttons");
18
19         statusBar = new JLabel("Click the mouse");
20         add(statusBar, BorderLayout.SOUTH);
21         addMouseListener(new MouseClickHandler()); // adiciona trat. de evento
22     }
23

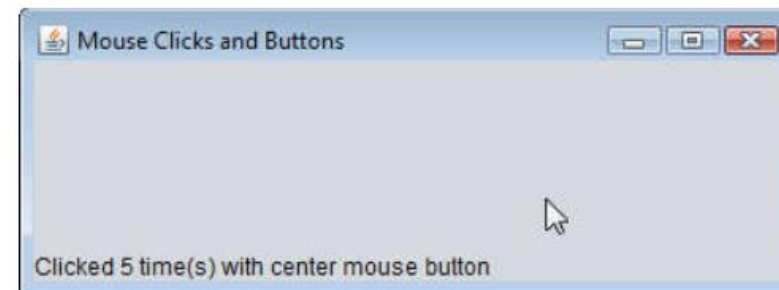
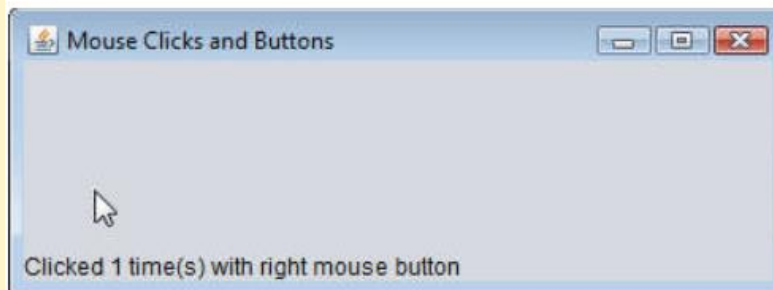
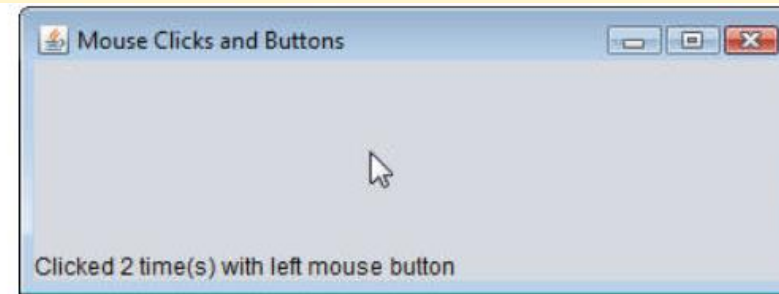
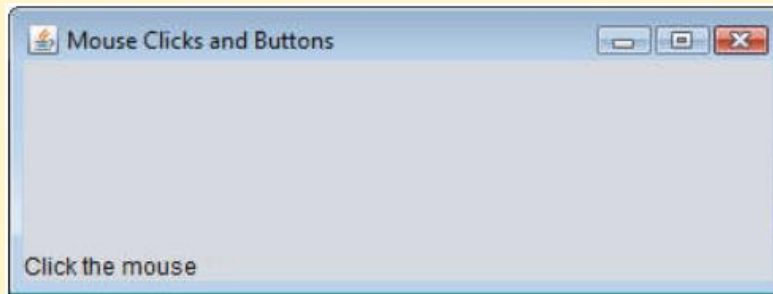
```

```

24 // classe interna para tratar eventos de mouse
25 private class MouseClickHandler extends MouseAdapter
26 {
27     // trata evento de clique de mouse e determina qual botão foi pressionado
28     @Override
29     public void mouseClicked(MouseEvent event)
30     {
31         int xPos = event.getX(); // obtém a posição x do mouse
32         int yPos = event.getY(); // obtém a 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 em statusBar
45     }
46 }
47 } // fim da classe MouseDetailsFrame

```

```
1 // Figura 12.32: 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);
12        mouseDetailsFrame.setVisible(true);
13    }
14 } // fim da classe MouseDetails
```





Exemplo 2 - Desenhando com o mouse

```
1 // Figura 12.34: PaintPanel.java
2 // Classe de adaptadores utilizada para implementar rotinas de tratamento de evento.
3 import java.awt.Point;
4 import java.awt.Graphics;
5 import java.awt.event.MouseEvent;
6 import java.awt.event.MouseMotionAdapter;
7 import java.util.ArrayList;
8 import javax.swing.JPanel;
9
10 public class PaintPanel extends JPanel
11 {
12     // lista das referências Point
13     private final ArrayList<Point> points = new ArrayList<>();
14
15     // configura GUI e registra rotinas de tratamento de evento de mouse
16     public PaintPanel()
17     {
```

```

18         // trata evento de movimento de mouse do frame
19         addMouseListener(
20             new MouseMotionAdapter() // classe interna anônima
21             {
22                 // armazena coordenadas da ação de arrastar e repinta
23                 @Override
24                 public void mouseDragged(MouseEvent event)
25                 {
26                     points.add(event.getPoint());
27                     repaint(); // repinta JFrame
28                 }
29             }
30         );
31     }
33 // preenche ovais em um quadro delimitador de 4 x 4 nas localizações espec. na janela
34     @Override
35     public void paintComponent(Graphics g)
36     {
37         super.paintComponent(g); // limpa a área de desenho
38
39         // desenha todos os pontos
40         for (Point point : points)
41             g.fillOval(point.x, point.y, 4, 4);
42     }
43 } // fim da classe PaintPanel

```



```
1 // Figura 12.35: Painter.java
```

```
2 // Testando o JPanel.
```

```
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 o JFrame
```

```
12         JFrame application = new JFrame("A simple paint program");
```

```
13
```

```
14         JPanel paintPanel = new JPanel();
```

```
15         application.add(paintPanel, BorderLayout.CENTER);
```

```
16
```

```
17         // cria um rótulo e o coloca em SOUTH do 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);
```

```
23         application.setVisible(true);
```

```
24     }
```

```
25 } // fim da classe Painter
```





Manipulando eventos do **teclado**

- A interface **KeyListener** possui os seguintes métodos:
 - keyPressed(KeyEvent e)
 - keyReleased(KeyEvent e)
 - keyTyped(KeyEvent e)



Manipulando eventos do teclado

- A classe **KeyEvent** possui os seguintes métodos úteis:
 - `String getKeyText(int);`
 - `int getKeyCode();`
 - `char getKeyChar();`
 - `boolean isActionKey();`
 - `String getKeyModifiersText(int)`
 - `int getModifiers();`
 - `boolean isShiftDown();` `isMetaDown();`
 - `boolean isControlDown();` `isAltDown();`

```

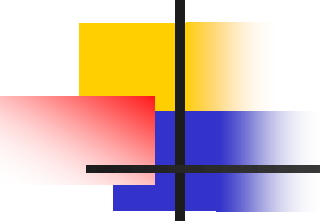
1 // Figura 12.36: KeyDemoFrame.java
2 // Tratamento de evento de teclado.
3 import java.awt.Color;
4 import java.awt.event.KeyListener;
5 import java.awt.event.KeyEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JTextArea;
8
9 public class KeyDemoFrame extends JFrame implements KeyListener
10 {
11     private final String line1 = ""; // primeira linha da área de texto
12     private final String line2 = ""; // segunda linha da área de texto
13     private final String line3 = ""; // terceira linha da área de texto
14     private final JTextArea textArea; // área de texto para exibir a saída
15
16     // construtor KeyDemoFrame
17     public KeyDemoFrame()
18     {
19         super("Demonstrating Keystroke Events");
20
21         textArea = new JTextArea(10, 15); // configura JTextArea
22         textArea.setText("Press any key on the keyboard...");
23         textArea.setEnabled(false);
24         textArea.setDisabledTextColor(Color.BLACK);
25         add(textArea); // adiciona área de texto ao JFrame
26
27         addKeyListener(this); // permite que o frame processe os eventos de teclado
28     }

```

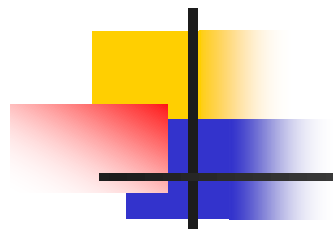
```

30 // trata pressionamento de qualquer tecla
31 @Override
32 public void keyPressed(KeyEvent event)
33 { // mostra a tecla pressionada
34     line1 = String.format("Key pressed: %s",
35                           KeyEvent.getKeyText(event.getKeyCode()));
36     setLines2and3(event); // configura a saída das linhas dois e três
37 }
38
39 // trata liberação de qualquer tecla
40 @Override
41 public void keyReleased(KeyEvent event)
42 { // mostra a tecla liberada
43     line1 = String.format("Key released: %s",
44                           KeyEvent.getKeyText(event.getKeyCode()));
45     setLines2and3(event); // configura a saída das linhas dois e três
46 }
47
48 // trata pressionamento de uma tecla de ação
49 @Override
50 public void keyTyped(KeyEvent event)
51 {
52     line1 = String.format("Key typed: %s", event.getKeyChar());
53     setLines2and3(event); // configura a saída das linhas dois e três
54 }

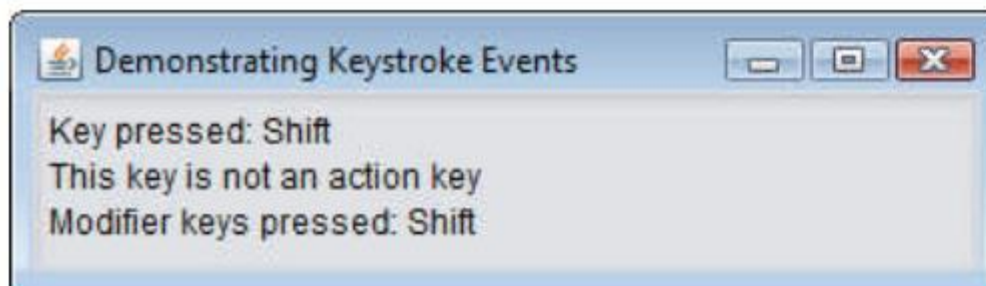
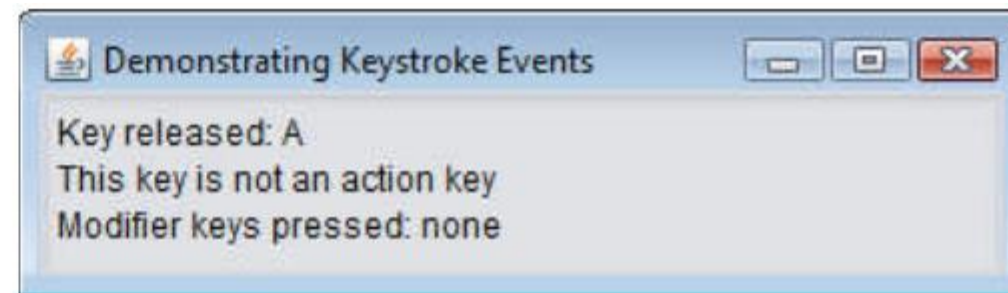
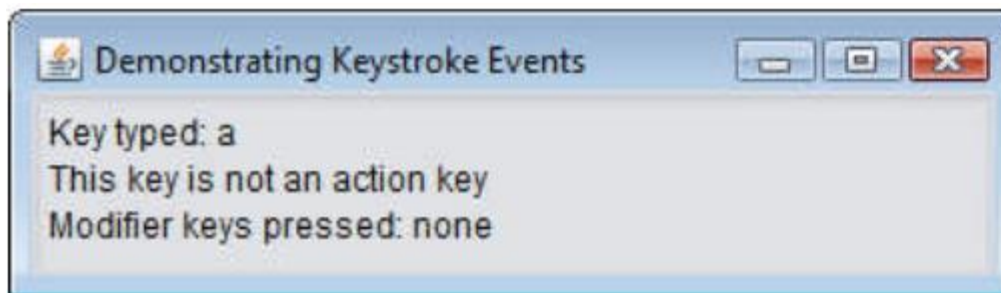
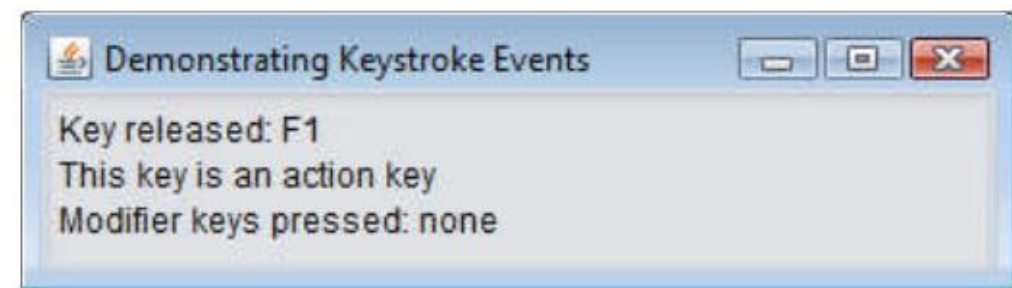
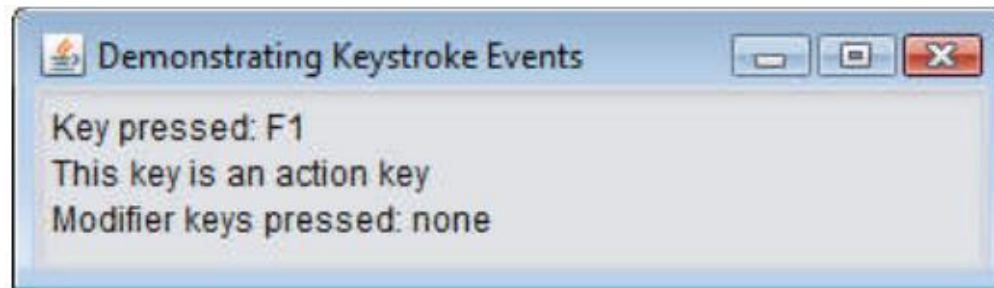
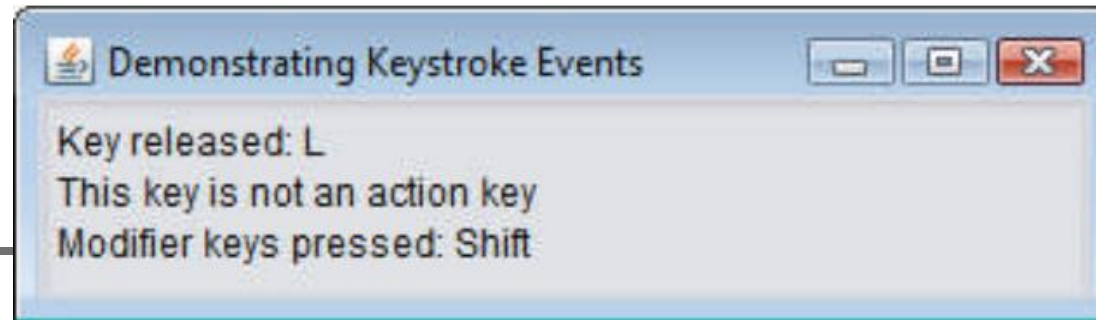
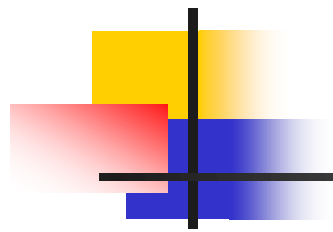
```



```
56 // configura segunda e terceira linhas de saída
57 private void setLines2and3(KeyEvent event)
58 {
59     line2 = String.format("This key is %san action key",
60         (event.isActionKey() ? "" : "not "));
61
62     String temp = KeyEvent.getKeyModifiersText(event.getModifiers());
63
64     line3 = String.format("Modifier keys pressed: %s",
65         (temp.equals("") ? "none" : temp)); // modificadores de saída
66
67     textArea.setText(String.format("%s\n%s\n%s\n",
68         line1, line2, line3)); // gera saída de três linhas de texto
69 }
70 } // fim da classe KeyDemoFrame
```



```
1 // Figura 12.37: 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);
12        keyDemoFrame.setVisible(true);
13    }
14 } // fim da classe KeyDemo
```





Referencias

- Java How to Program 3, 4, 5, 6 , 7, 8, 9, 10 ed.
Deitel e Deitel