INTERFACE GRÁFICA

Interface Gráfica

- E/S pode ser feita por interface gráfica
- Biblioteca Swing
  - `javax.swing.*`

AWT x Swing

- AWT (*Abstract Window Toolkit*) compõe o núcleo da JFC (*Java Foundation Classes*)
- Swing faz parte da JFC
- Biblioteca Java Swing foi definida com base em AWT
  - Usa classes AWT para gerenciamento da disposição de componentes em janelas (LayoutManagers) e interfaces com métodos para tratamento de eventos

Java Foundation Classes

- **J**ava **F**oundation **C**lasses é um conjunto de pacotes (15) usados para criação de interfaces gráficas com o usuário (GUI).
- Características da JFC:
  - Componentes gráficos Swing
  - Look&Feel adaptável
  - Recursos de arrastar e soltar
  - Java2D (gráficos em 2D)
  - Acessibilidade (tecnologias assistentes)

Java Foundation Classes

- Principais pacotes usados:
  - `javax.swing`
  - `javax.swing.event`
- **A**bstract **W**indowing **T**oolkit
  - Conjunto básico de componentes gráficos de Java para uso em GUI.
  - Muito limitados.

AWT x Swing

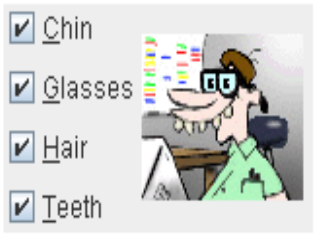
- Em geral, para cada componente AWT existe um componente Swing análogo
  - Porém, sempre precedido por "J"
  - Ex: Applet -> JApplet Frame -> Jframe
  - Existem componentes Swing que não possuem componente análogo em AWT

Exemplo de aplicação com Swing


```
public static void main(String[] args) {
    createAndShowGUI();     }

private static void createAndShowGUI() {
    // atribui uma decoração mais bonita a janela
    JFrame.setDefaultLookAndFeelDecorated(true);
    // cria e define o tamanho da janela
    JFrame frame = new JFrame("Programação em Java");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(new Dimension(450, 450));


    JLabel label = new JLabel("Teste de uso de Java Swing");
    frame.getContentPane().add(label);
    // mostra a janela
    frame.setVisible(true);
}
```

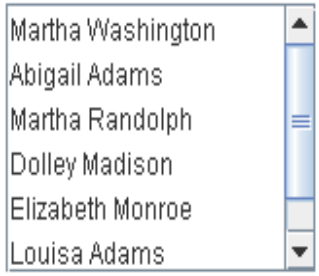
Objetos gráficos do Swing

JButton

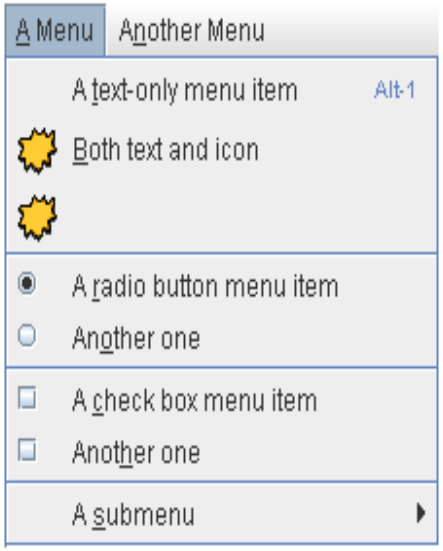


JCheckBox

JComboBox

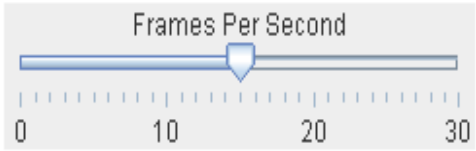


JList



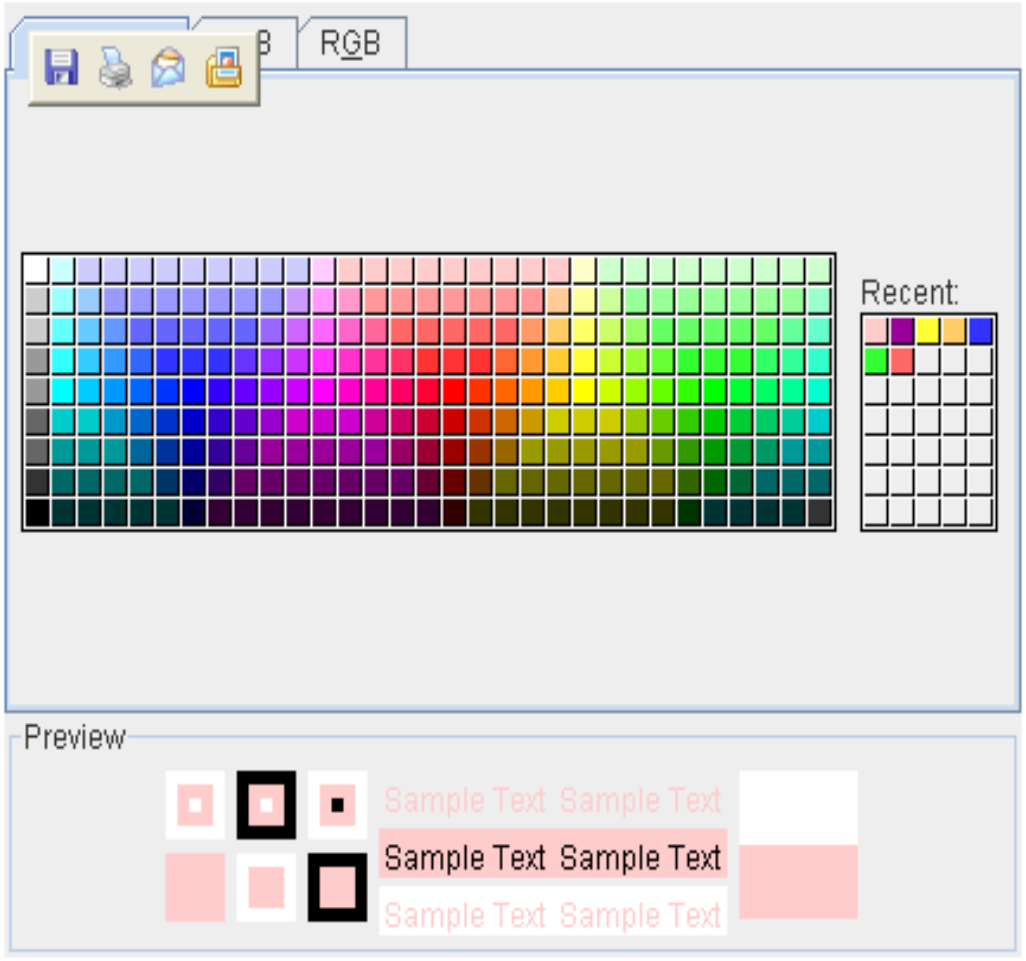
JMenu

JRadioButton

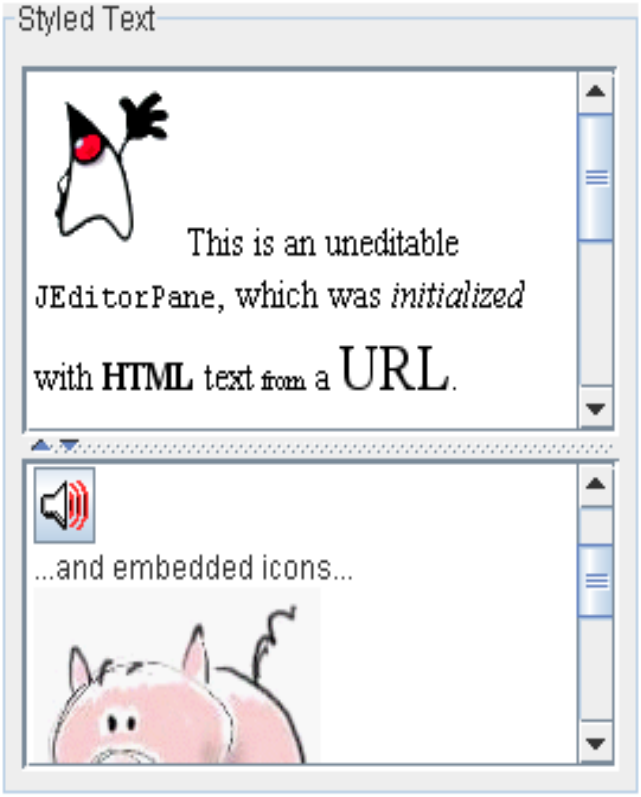


JSlider

Objetos gráficos do Swing

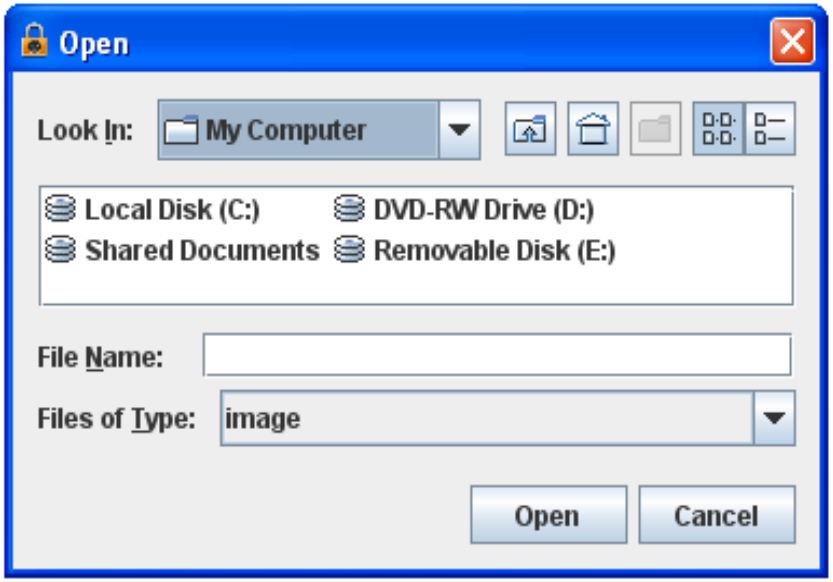


JColorChooser



JEditorPane and JTextPane

Objetos gráficos do Swing



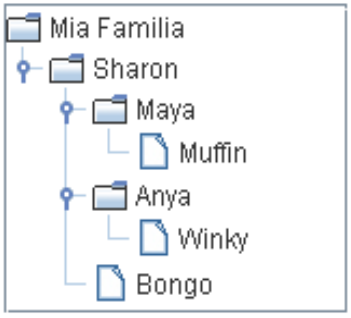
JFileChooser

Host	User	Password	Last Modified
Biocca Games	Freddy	!#asf6Awwzb	Mar 16, 2006
zabble	ichabod	Tazb!34$fZ	Mar 6, 2006
Sun Developer	fraz@hotmail.co...	AasW541!fbZ	Feb 22, 2006
Heirloom Seeds	shams@gmail....	bkz[ADF78!	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.c...	vbAf124%z	Feb 22, 2006

JTable

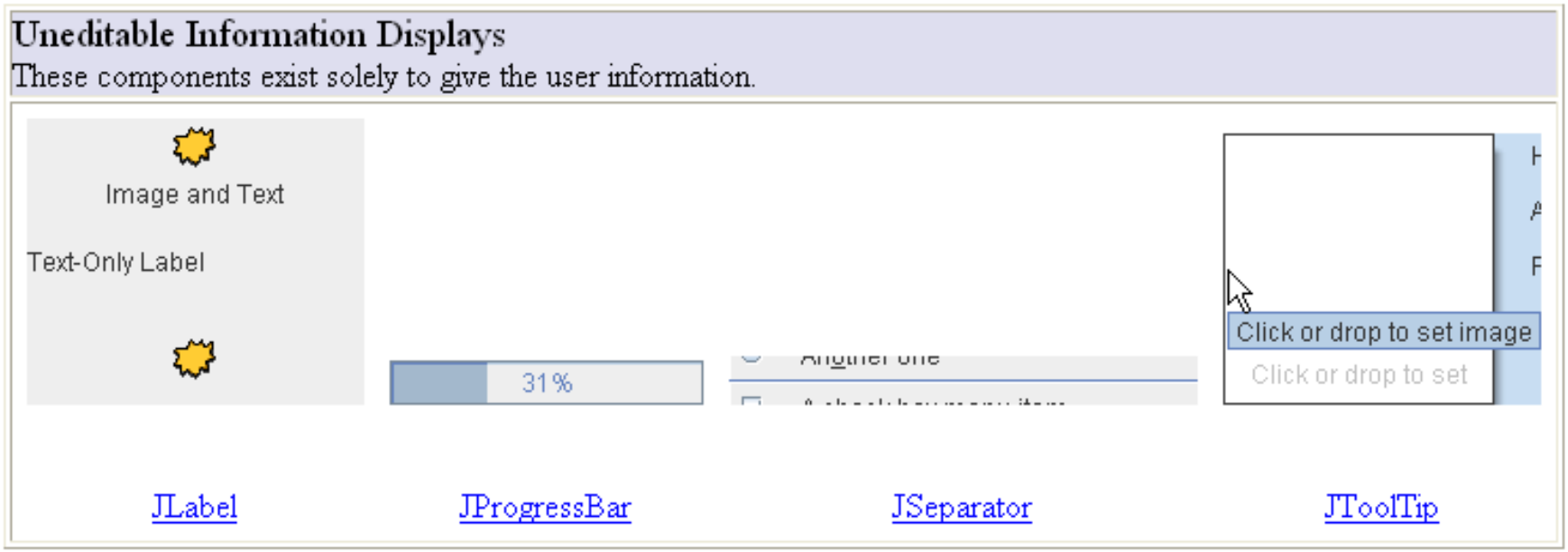
This is an editable JTextArea. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.

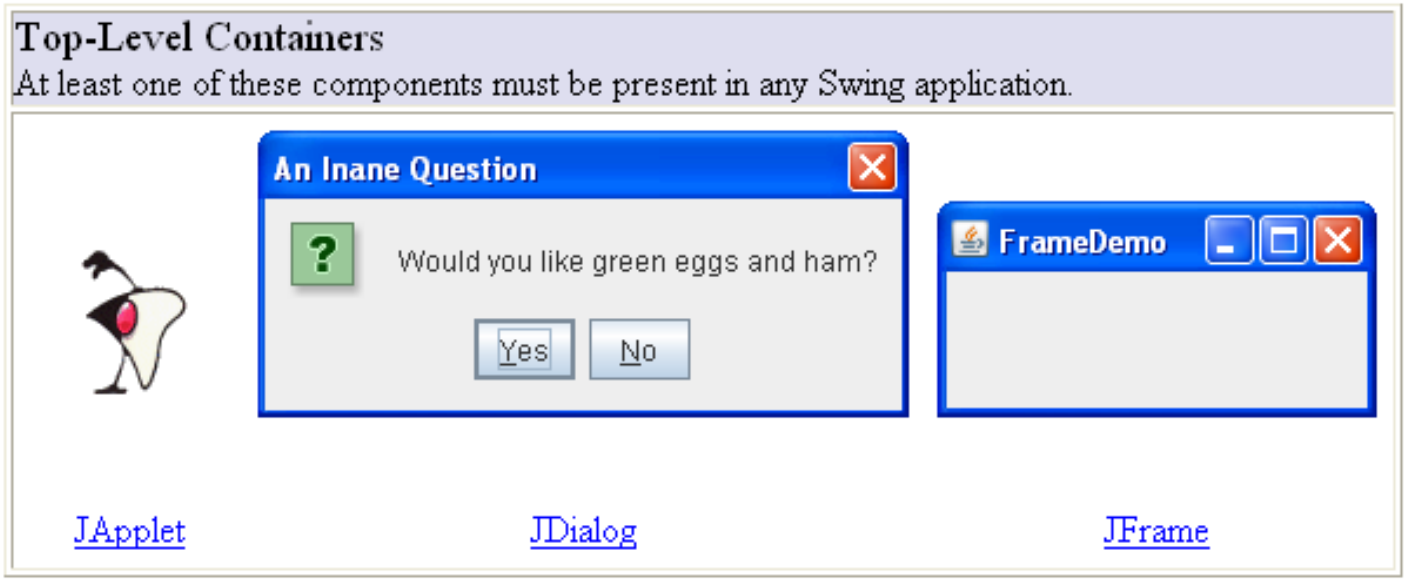
JTextArea



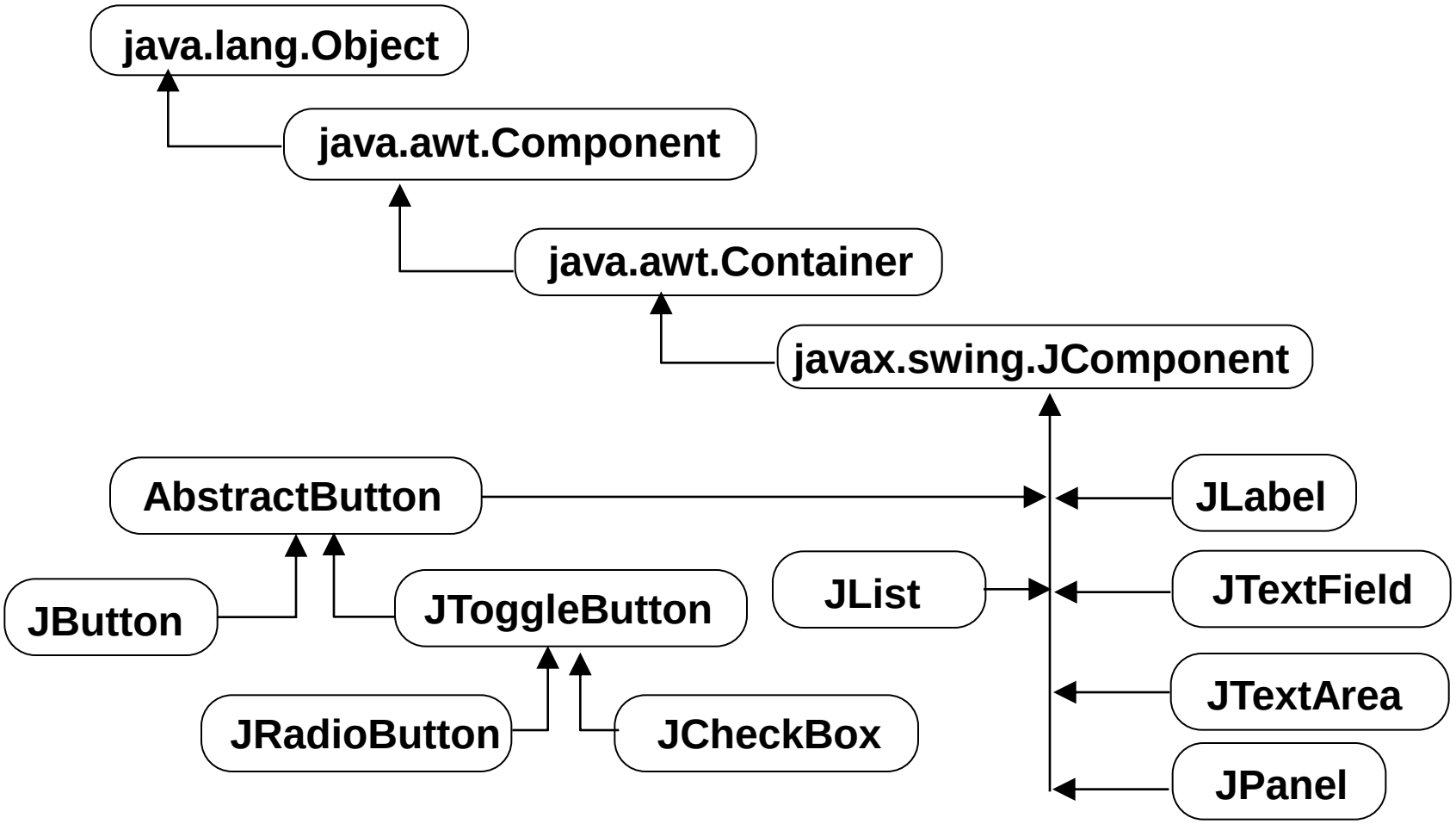
JTree

Objetos gráficos do Swing





Hierarquia


java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
AbstractButton
JLabel
JList
JTextField
JButton
JToggleButton
JTextArea
JRadioButton
JCheckBox
JPanel

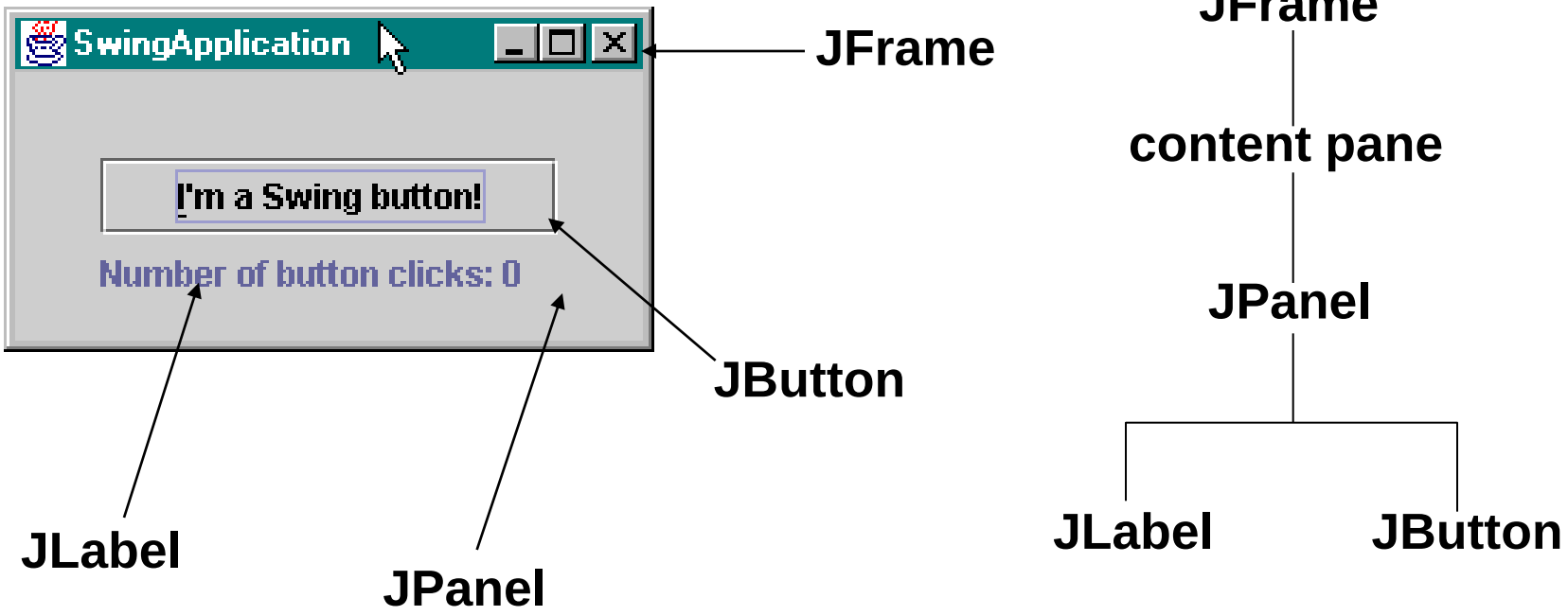
Hierarquia

- **`Component`**
  - Define um componente de interface.
  - Métodos como paint() e repaint().
- **`Container`**
  - Define um componente que pode conter outros componentes.
  - Define métodos como add() para adicionar componentes em seu interior.
  - Possui um gerenciador de layout.

Hierarquia

- **`JComponent`**
  - Superclasse da maioria dos componentes Swing.
  - Principais características:
    - *Look and Feel* adaptável
    - Atalhos de teclas
    - Tratamento de eventos
    - Tooltips

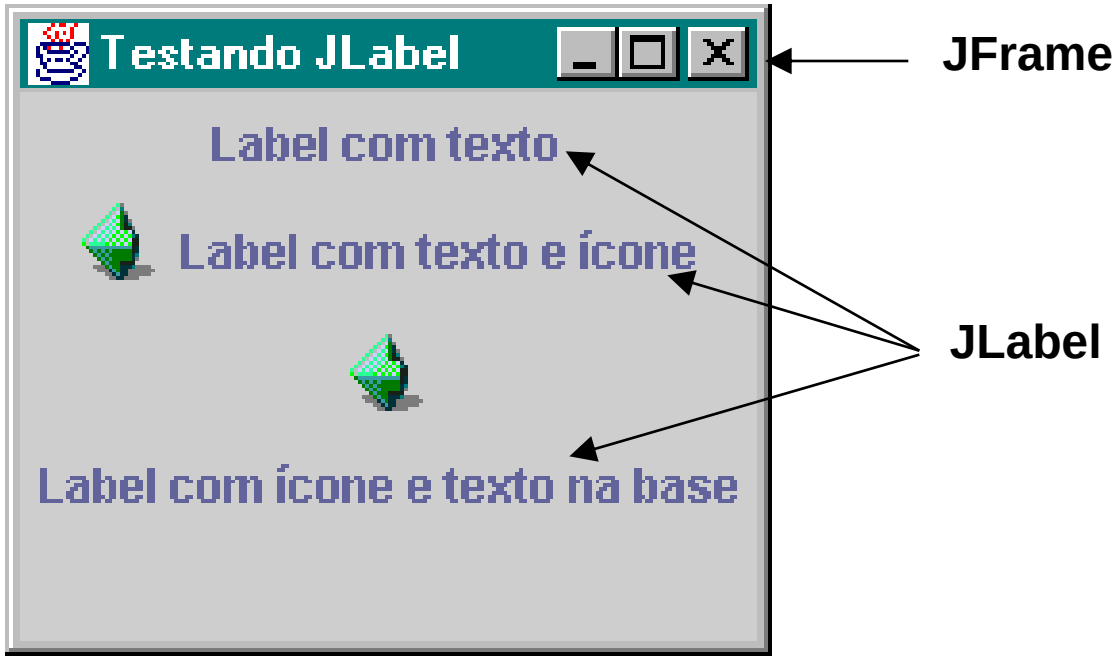
Hierarquia visual



Hierarquia visual

- **`JFrame`** é o *top level container*. Outros tipos são applets (JApplet) e janelas de diálogo (JDialog)
- Todo *top-level container* possui um **content pane** onde os componentes são adicionados
- **`JPanel`** é um container intermediário usado para agrupar e posicionar outros componentes.

Exemplo com JLabel



```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class LabelTest extends JFrame{
  private JLabel label1, label2, label3;

  public LabelTest() { ... }  //construtor

  public static void main(String[] args){
     LabelTest app = new LabelTest();

    }
}
```

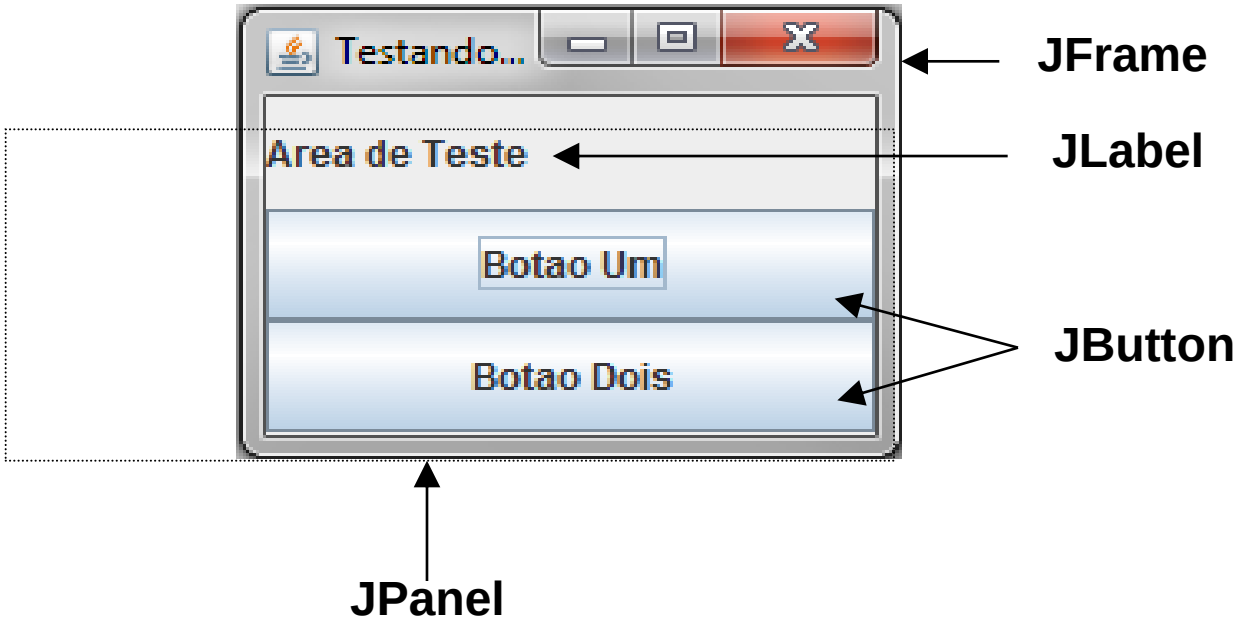
```
public LabelTest() {    //construtor
    super("Testando JLabel");

    Container c = this.getContentPane();
    c.setLayout(new FlowLayout());

    label1 = new JLabel("Label com texto");
    label1.setToolTipText("Este é o label 1");
    c.add(label1);

    Icon seta = new ImageIcon("voltar.gif");
    label2 = new JLabel("Label com texto e ícone",
                        seta,
                        SwingConstants.LEFT);
    label2.setToolTipText("Este é o label 2");
    c.add(label2);

    ...  //continua
```

```
public LabelTest() {    //continuação do anterior

    ...

    label3 = new JLabel();
    label3.setText(
              "Label com ícone e texto na base");
    label3.setIcon(seta);
    label3.setHorizontalTextPosition(
              SwingConstants.CENTER);
    label3.setVerticalTextPosition(
              SwingConstants.BOTTOM);
    label3.setToolTipText("Este é o label 3");
    c.add(label3);

    this.setSize(200, 170);
    this.setVisible(true);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

}
```

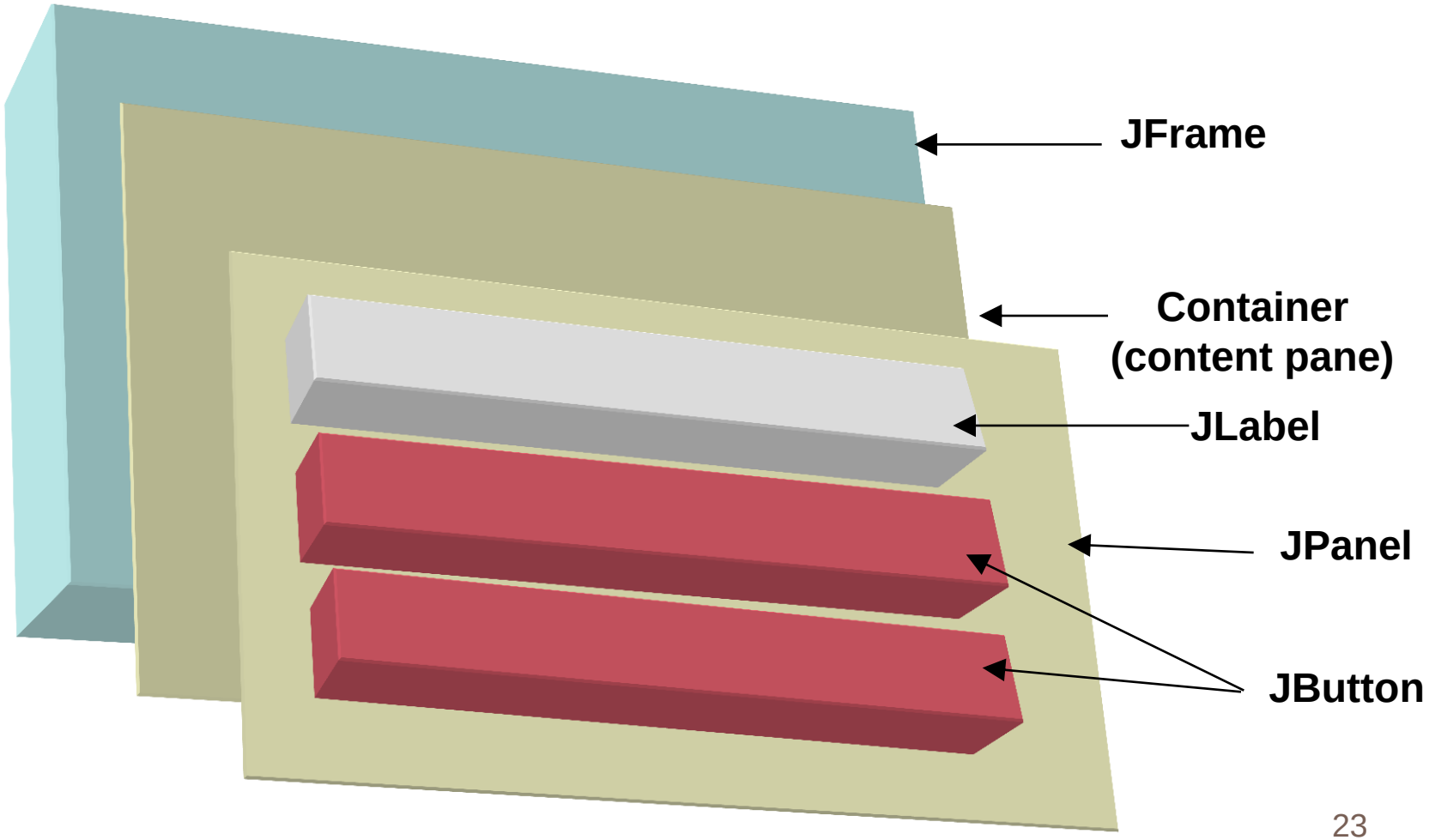
Passos para criar uma GUI

1. Declarar os componentes como propriedades da classe que estende JFrame (o *top-level container*).
2. Instanciar os componentes no construtor da classe ou na declaração.
3. Definir o gerenciador de layout do frame.
4. Registrar *listeners* para os componentes que querem tratar eventos.
5. Adicionar os componentes ao *content pane* do frame ou a algum outro container.
6. Definir o tamanho do frame com **`setSize()`**.

Outro Exemplo

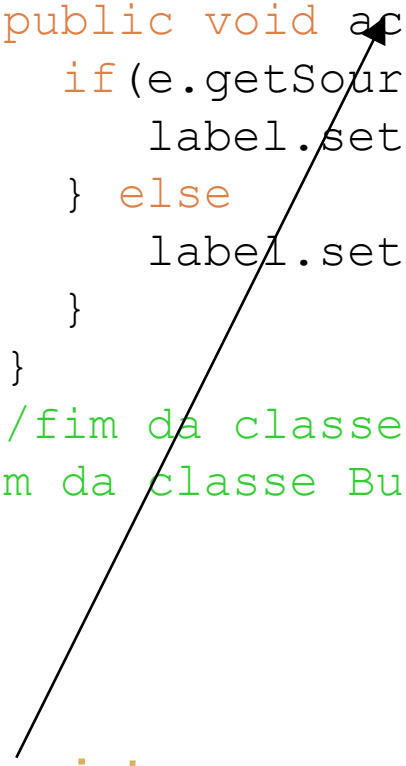


Hierarquia visual



```
public class ButtonTest extends JFrame{
   private JButton botao1, botao2;
   private JLabel label = new JLabel("Area de Teste");
   private JPanel painel;

   public ButtonTest() { ... }  //construtor

   public static void main(String[] args) {
      ButtonTest buttonTest = new ButtonTest();
   }
   private class TrataBotoes implements ActionListener {
      public void actionPerformed(ActionEvent e){
        if(e.getSource() == botao1) {
           label.setText("Pressionado botao 1");
        } else
           label.setText("Pressionado botao 2");
        }
      }
   } //fim da classe TrataBotoes
} //fim da classe ButtonTest
```

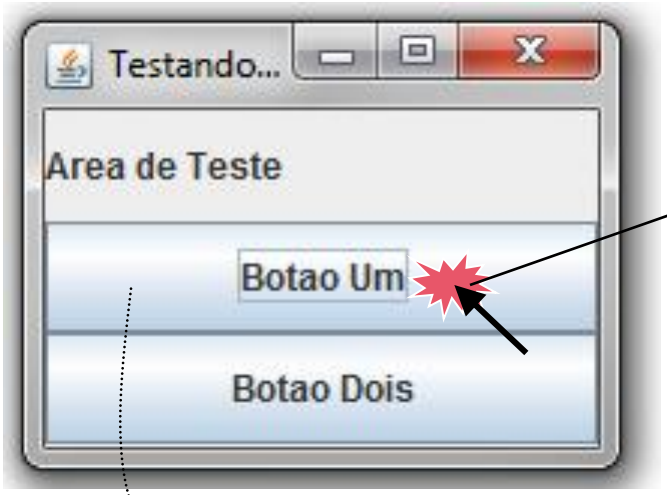
```
    private class TrataBotoes implements ActionListener {
      public void actionPerformed(ActionEvent e){
        if(e.getSource() == botao1) {
           label.setText("Pressionado botao 1");
        } else
           label.setText("Pressionado botao 2");
        }
      }
   } //fim da classe TrataBotoes
} //fim da classe ButtonTest
```

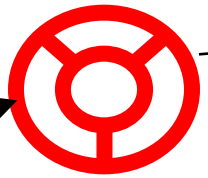


Classe interna responsável pelo tratamento dos eventos sobre os botões!

```
public ButtonTest() {  //construtor da classe ButtonTest
    super("Testando botões");
    Container c = getContentPane();

    painel = new JPanel();
    painel.setLayout(new GridLayout(0,1));
    TrataBotoes tratador = new TrataBotoes();

    painel.add(label);
    botao1 = new JButton("Botao Um");
    botao1.addActionListener(tratador);
    painel.add(botao1);

    botao2 = new JButton("Botao Dois");
    botao2.addActionListener(tratador);
    painel.add(botao2);

    c.add(painel, BorderLayout.CENTER);
    setSize(200, 100);
    setVisible(true);
}
```

O Modelo de Eventos de Java

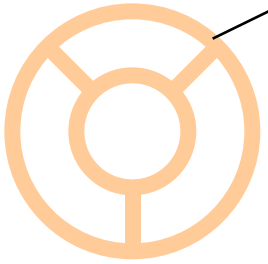




um AWTEvent

um EventListener

registro



um JComponent

- Componentes devem registrar **observadores** (*listeners)* que percebam e tratem os eventos

- Cada tipo de evento tem um **observador** específico.

O Modelo de Eventos de Java

- Criação do objeto *observador* (no construtor!):

```
TrataBotoes tratador = new TrataBotoes();
```

- Registra *observador* para os botões (idem):

```
botao1.addActionListener(tratador);
...
botao2.addActionListener(tratador);
```

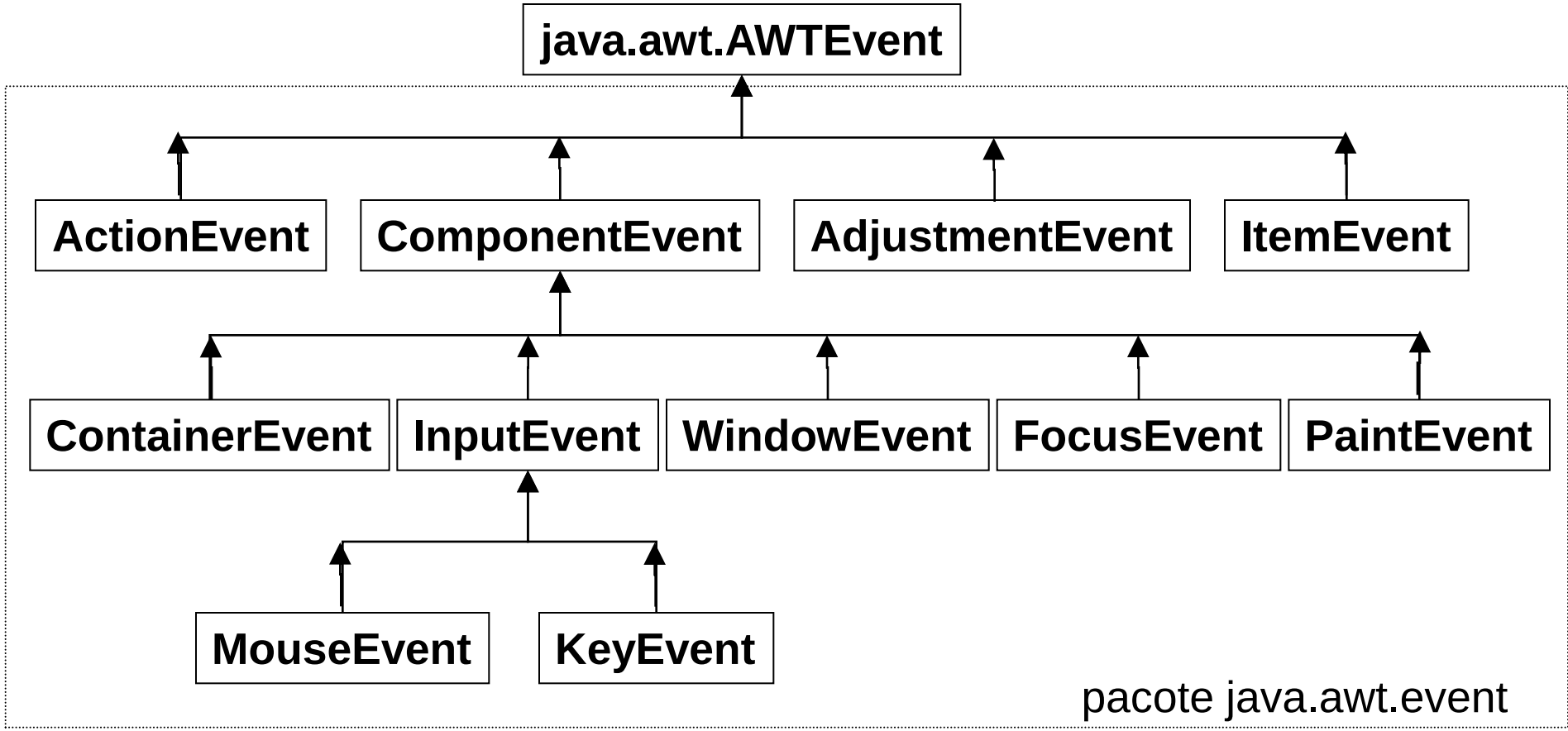
- Método do *observador* que trata o evento:

```
public void actionPerformed(ActionEvent e){
    ...
}
```

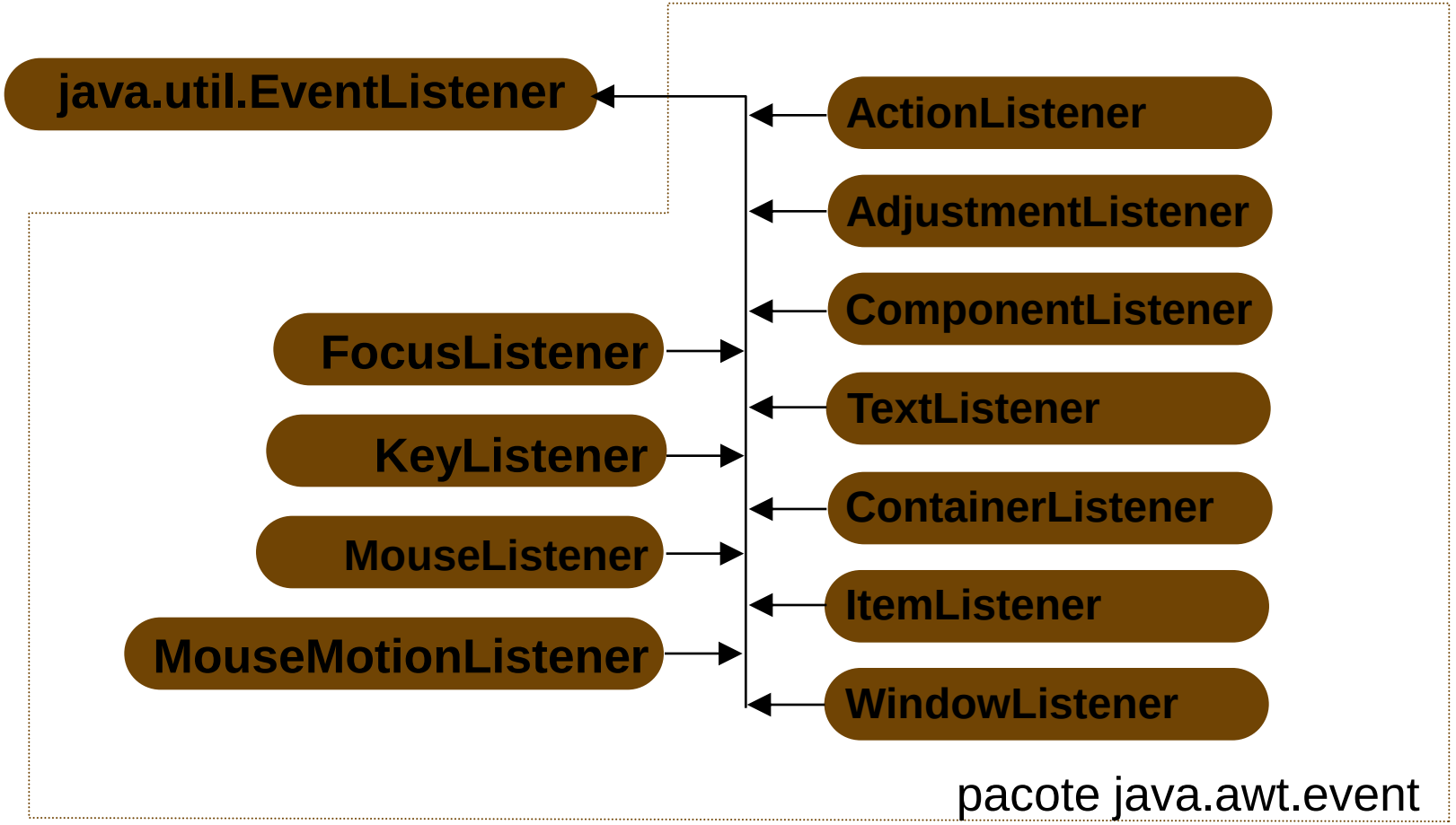
Modelo de Eventos de Java

- GUIs são orientadas a eventos, i. e., geram eventos quando o usuário interage com seus componentes.
- Interação = clicar o mouse, clicar um botão, digitar num campo de texto, fechar uma janela, mover o ponteiro,...
- A cada interação um evento é gerado e enviado ao programa.

Hierarquia de Eventos



Hierarquia de Observadores

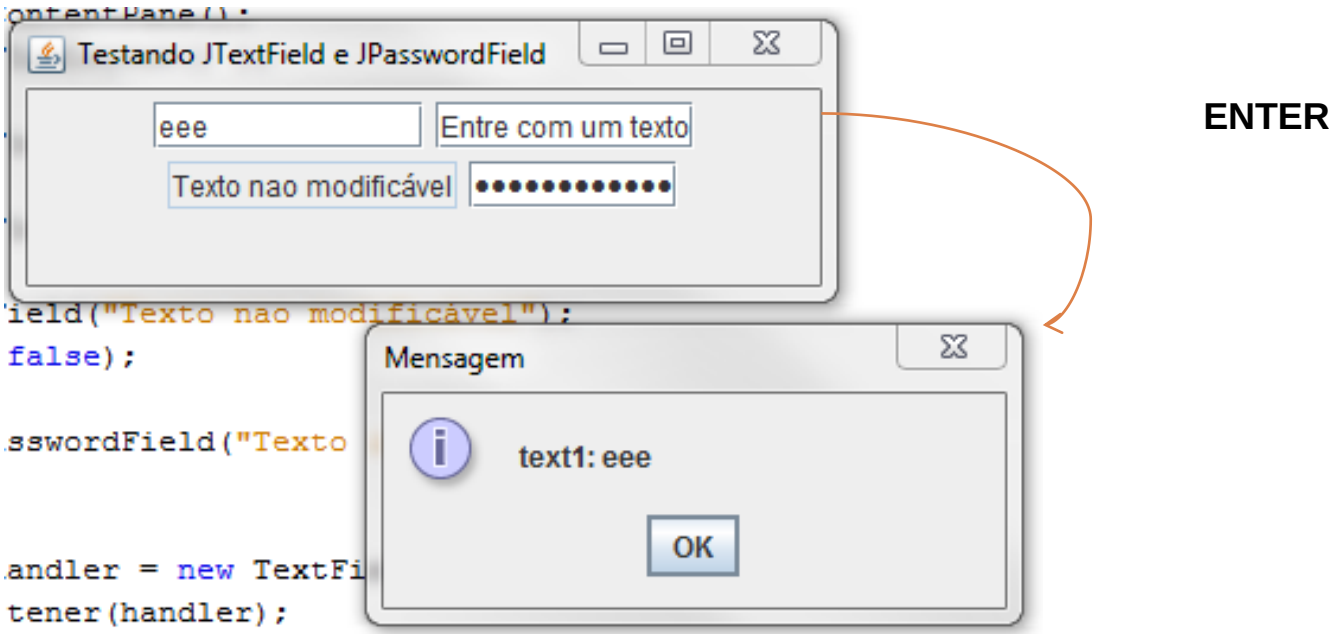


Modelo de Eventos de Java

- Eventos são classes, observadores são interfaces!

- Criar um observador de eventos para um componente consiste em criar uma classe que implemente a interface *observadora* do evento a ser tratado
  - Cada uma das interfaces observadoras define um ou mais métodos tratadores de eventos.

Exemplo


Testando JTextField e JPasswordField
eee
Entre com um texto
Texto nao modificável
Mensagem
text1: eee
OK
ENTER

```
public class TextFieldTest extends JFrame{
  private JTextField text1, text2, text3;
  private JPasswordField password;

  public TextFieldTest() {    //construtor
    ...
  }

  public static void main(String[] args){
    TextFieldTest app = new TextFieldTest();

    app.addWindowListener(
      new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          System.exit(0);
        }
      }
    );
  }
```

```
  private class TextFieldHandler
                implements ActionListener{
    public void actionPerformed(ActionEvent e){
      String s = "";
      if(e.getSource() == text1)
        s = "text1: " + e.getActionCommand();
      else if(e.getSource() == text2)
        s = "text2: " + e.getActionCommand();
      else if(e.getSource() == text3)
        s = "text3: " + e.getActionCommand();
      else if(e.getSource() == password) {
        JPasswordField pwd =
           (JPasswordField) e.getSource();
        s = "password: " +
           new String(pwd.getPassword());
      }
      JOptionPane.showMessageDialog(null, s);
    }
  }
}
```

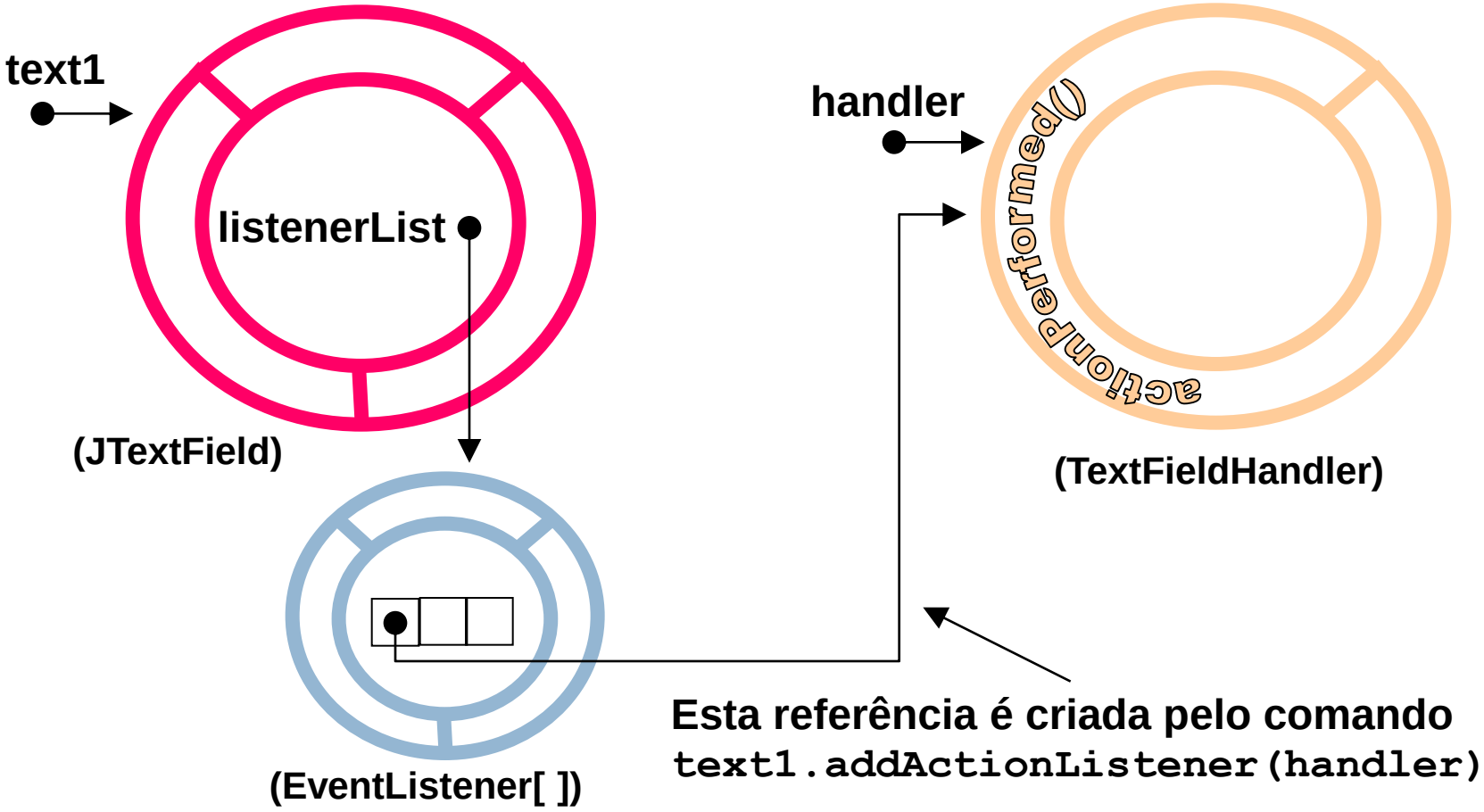
Criação da classe que implementa a interface observadora e seu respectivo método de tratamento

```
public TextFieldTest() {  //construtor
   super("Testando JTextField e JPasswordField");
   Container c = getContentPane();
   c.setLayout( new FlowLayout() );
   text1 = new JTextField(10);
   c.add(text1);
   text2 = new JTextField("Entre com um texto");
   c.add(text2);
   text3 = new JTextField("Texto nao modificável");
   text3.setEditable(false);
   c.add(text3);
   password = new JPasswordField("Texto oculto");
   c.add(password);
   TextFieldHandler handler = new TextFieldHandler();
   text1.addActionListener(handler);
   text2.addActionListener(handler);
   text3.addActionListener(handler);
   password.addActionListener(handler);
   setSize(325,100);
   setVisible(true);;
}
```

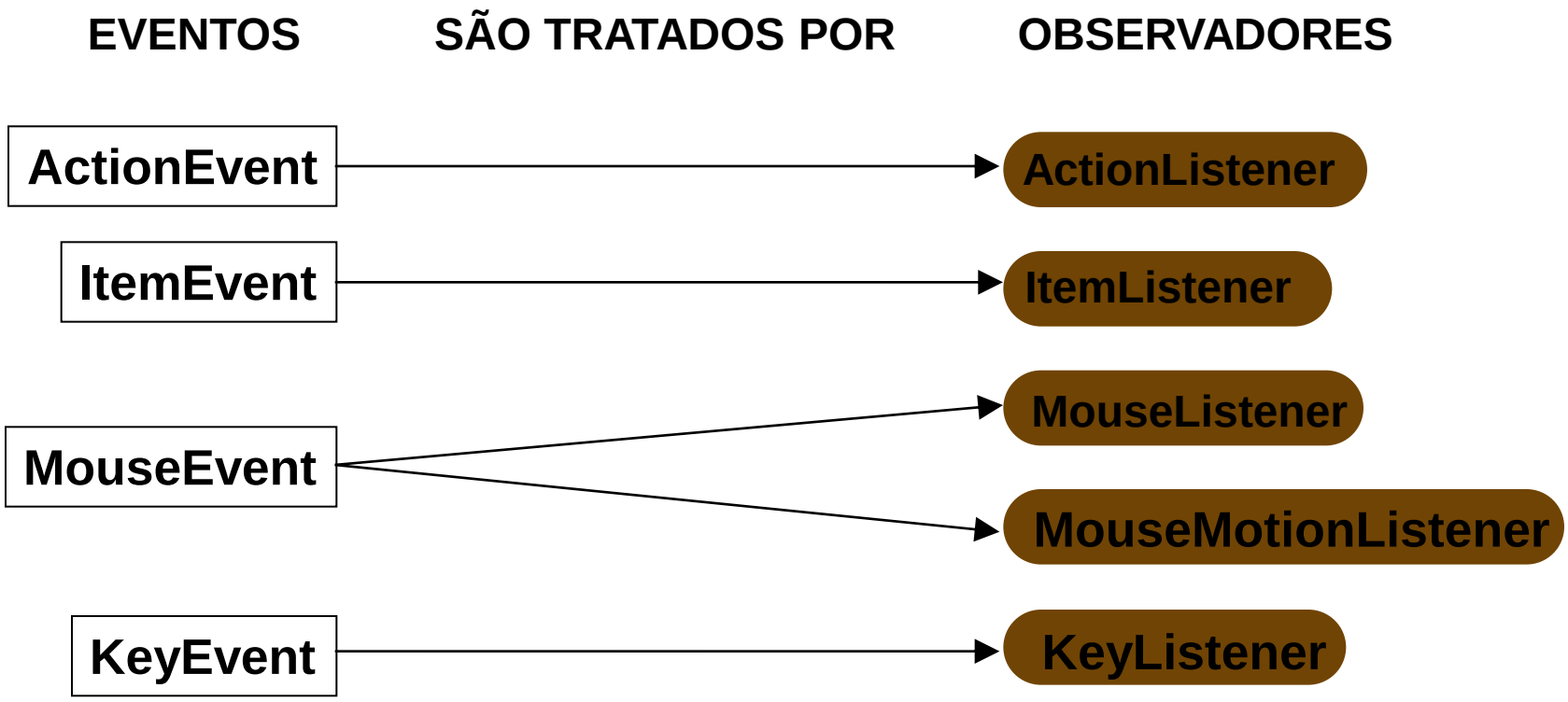
Criação e configuração dos componentes

Criação e registro do observador

Esquematicamente



Associação Evento x Listener



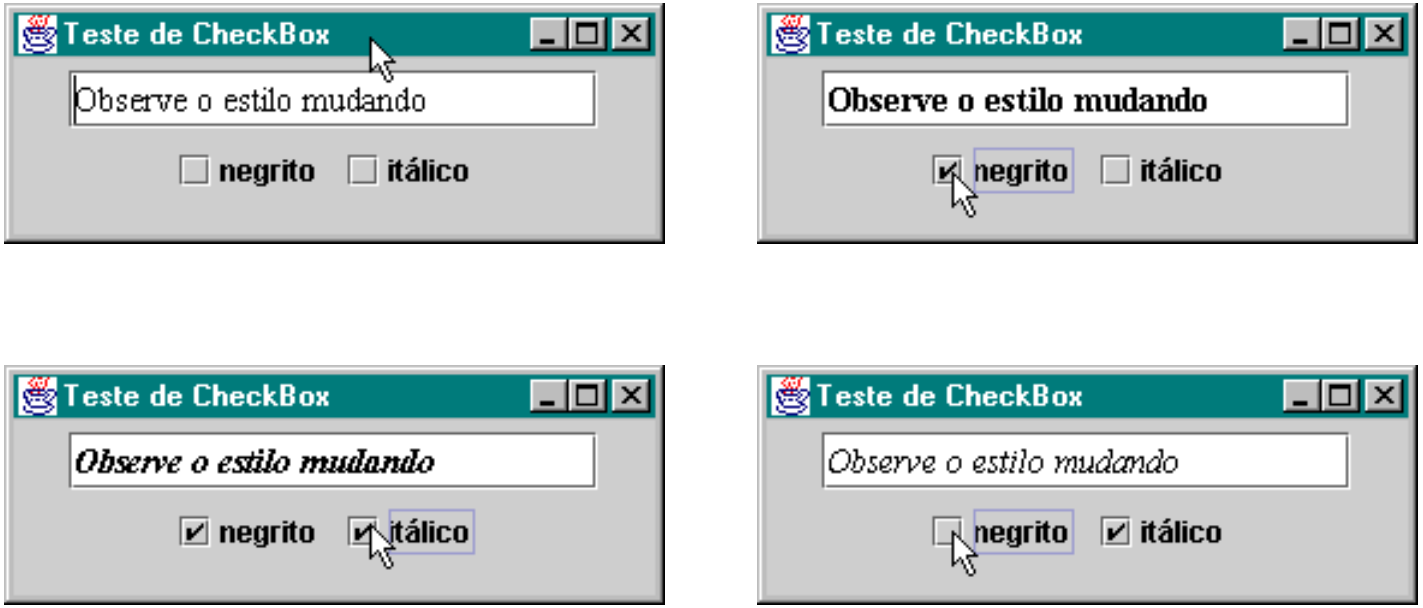
Interface ActionListener

- Eventos capturados:
  - ActionEvent
- Método(s) definidos:
  - void actionPerformed(ActionEvent e)
- Componentes que registram:
  - JComboBox, JButton, JMenuItem, JTextField, JPasswordField
- Método usado para registrar:
  - addActionListener(ActionListener al)

Interface ItemListener

- Eventos capturados:
  - ItemEvent
- Método(s) definidos:
  - void itemStateChanged(ItemEvent e)
- Componentes que registram:
  - JCheckBox, JRadioButton, JComboBox
- Método usado para registrar:
  - addItemListener(ItemListener il)

Exemplo - JCheckBox

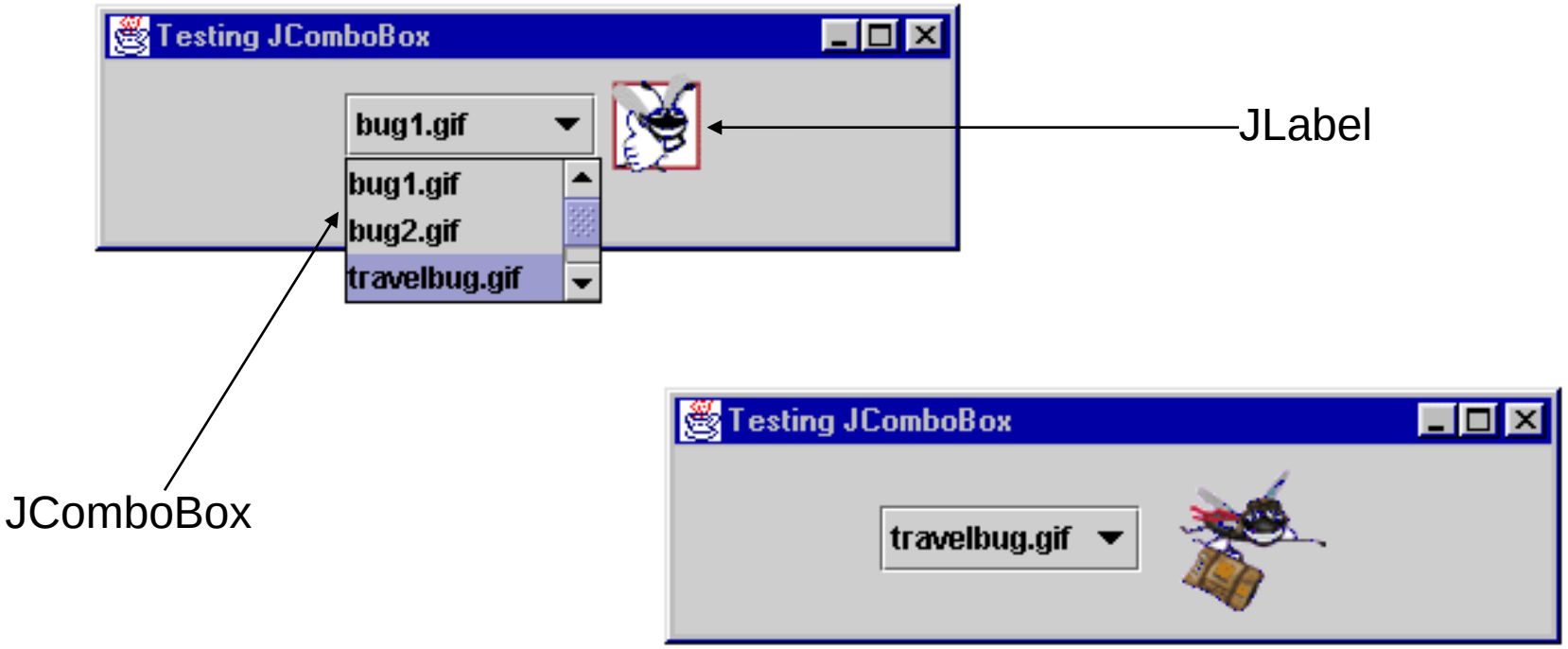


```
public class CheckBoxTest extends JFrame{
  private JTextField t;
  private JCheckBox bold, italic;

  public CheckBoxTest() { ... }

  public static void main(String[] args) {
    CheckBoxTest checkBoxTest = new CheckBoxTest();
  }

  private class CheckBoxHandler
                implements  ItemListener{
     private int valBold = Font.PLAIN;
     private int valItalic = Font.PLAIN;
```

```
  public void itemStateChanged(ItemEvent e){
        if(e.getSource() == bold)
          if(e.getStateChange() == ItemEvent.SELECTED)
            valBold = Font.BOLD;
          else
            valBold = Font.PLAIN;
        if(e.getSource() == italic)
          if(e.getStateChange() == ItemEvent.SELECTED)
            valItalic = Font.ITALIC;
          else
            valItalic = Font.PLAIN;
        t.setFont(
           new Font("TimesRoman",
                    valBold+valItalic, 14));
        t.repaint();
      }
  }
}
```

```
public CheckBoxTest() {
    super("Teste de CheckBox");
    Container c = getContentPane();
    c.setLayout(new FlowLayout());
    t = new JTextField("Observe o estilo mudando", 20);
    t.setFont(new Font("TimesRoman", Font.PLAIN, 14) );
    c.add(t);
    bold = new JCheckBox("negrito");    c.add(bold);
    italic = new JCheckBox("itálico"); c.add(italic);
    CheckBoxHandler handler = new CheckBoxHandler();
    bold.addItemListener(handler);
    italic.addItemListener(handler);
    addWindowListener( new WindowAdapter(){
          public void windowClosing(WindowEvent e){
            System.exit(0);
          }}
    );
    setSize(275, 100);
    setVisible(true);
  }
```

Exemplo - JComboBox



```
public class ComboBoxTest extends JFrame {
   private JComboBox images;
   private JLabel label;
   private String names[] =
      { "bug1.gif", "bug2.gif",
        "travelbug.gif", "buganim.gif" };
   private Icon icons[] =
      { new ImageIcon( names[ 0 ] ),
        new ImageIcon( names[ 1 ] ),
        new ImageIcon( names[ 2 ] ),
        new ImageIcon( names[ 3 ] ) };

   //Construtor nos próximos slides
   public ComboBoxTest() { ... }

//a classe continua...
```

```
    public static void main( String args[] )
    {
        ComboBoxTest app = new ComboBoxTest();

        app.addWindowListener(
            new WindowAdapter() {
                public void windowClosing(WindowEvent e)
                {
                    System.exit( 0 );
                }
            }
        );
    }
} //fim da classe ComboBoxTest
```

Classe Anônima

```
public ComboBoxTest() {
  super("Testing JComboBox");
  Container c = getContentPane();
  c.setLayout( new FlowLayout() );
  images = new JComboBox( names );
  images.setMaximumRowCount( 3 );
  images.addItemListener(
    new ItemListener() {
      public void itemStateChanged( ItemEvent e )
      {
        label.setIcon(icons[images.getSelectedIndex()]);
      }
    }
  );
  c.add( images );  label = new JLabel( icons[ 0 ] );
  c.add( label );  setSize( 350, 100 ); setVisible(true);
}
```

Classe Anônima

JComboBox

- Construtores:
  - JComboBox()
  - JComboBox(Object[] itens)
- Métodos de instância:
  - void addItem(Object item): adiciona um item ao combo
  - Object getItemAt(int index): devolve o item na posição index
  - int getItemCount(): devolve o número de itens

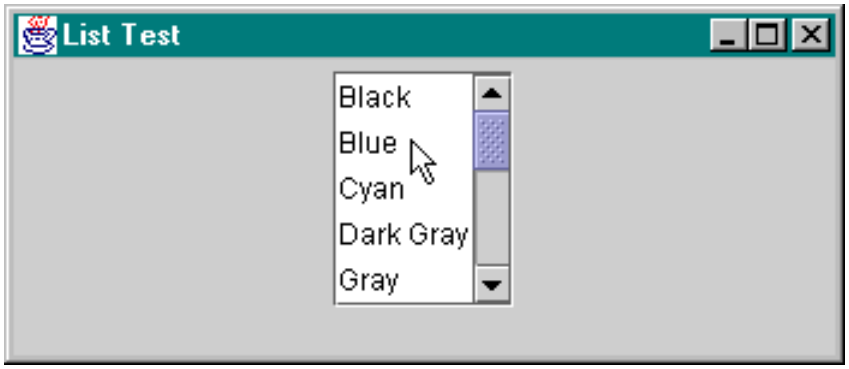
JComboBox

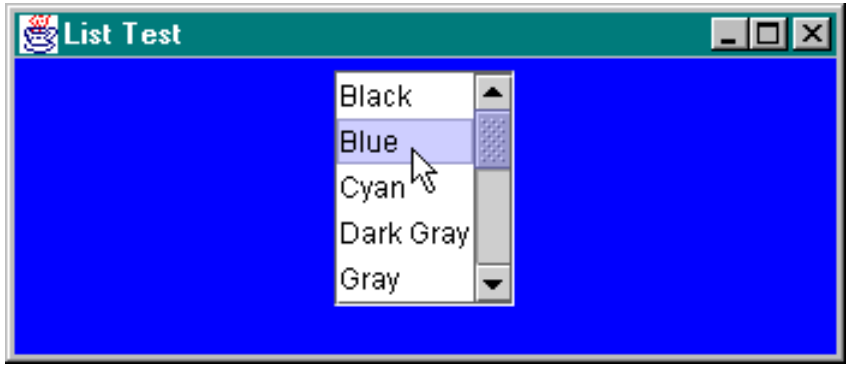
- Métodos de instância:
  - int getSelectedIndex(): devolve a posição do item selecionado ou -1 se não houver nenhum.
  - Object getSelectedItem(): retorna o item selecionado
  - void removeAllItens(): remove todos os itens do combo
  - void removeItemAt(int pos): remove item na posição pos
  - void setEnabled(boolean b): habilita/desabilita combobox
  - void setSelectedIndex(int pos): seleciona o item em pos

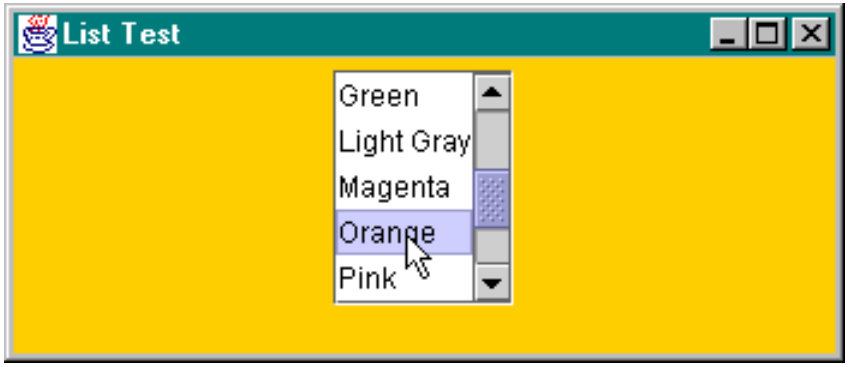
Interface ListSelectionListener

- Eventos capturados:
  - ListSelectionEvent
- Método(s) definidos:
  - void valueChanged(ListSelectionEvent e)
- Componentes que registram:
  - JList
- Método usado para registrar:
  - addListSelectionListener(ListSelectionListener lsl)

Exemplo


List Test
Black
Blue
Cyan
Dark Gray
Gray


List Test
Black
Blue
Cyan
Dark Gray
Gray


List Test
Green
Light Gray
Magenta
Orange
Pink

```
public class ListTest extends JFrame {
  private JList colorList;
  private Container c;

  private String colorNames[] =
  { "Black", "Blue", "Cyan", "Dark Gray", "Gray",
    "Green", "Light Gray", "Magenta", "Orange",
    "Pink", "Red", "White", "Yellow" };

  private Color colors[] =
  { Color.black, Color.blue, Color.cyan,
    Color.darkGray, Color.gray, Color.green,
    Color.lightGray, Color.magenta, Color.orange,
    Color.pink, Color.red,Color.white, Color.yellow
  };

  public ListTest() {...} //construtor
```

```
 public static void main( String args[] )
    {
       ListTest app = new ListTest();

       app.addWindowListener(
          new WindowAdapter() {
             public void windowClosing( WindowEvent e )
             {
                System.exit( 0 );
             }
          }
       );
    }
} // ListTest
```

```
public ListTest(){        //construtor
  super( "List Test" );
  c = getContentPane();
  c.setLayout( new FlowLayout() );

  colorList = new JList( colorNames );
  colorList.setVisibleRowCount( 5 );
  colorList.setSelectionMode(
     ListSelectionModel.SINGLE_SELECTION );
  c.add( new JScrollPane( colorList ) );
  colorList.addListSelectionListener(
     new ListSelectionListener() {
       public void valueChanged( ListSelectionEvent e ){
          c.setBackground(
          colors[ colorList.getSelectedIndex() ] );
       }
     }
  );
  setSize( 350, 150 ); setVisible(true);
}
```

Interface MouseListener

- Eventos capturados:
  - MouseEvent
- Método(s) definidos:
  - void mousePressed(MouseEvent e)
  - void mouseClicked(MouseEvent e)
  - void mouseReleased(MouseEvent e)
  - void mouseEntered(MouseEvent e)
  - void mouseExited(MouseEvent e)
- Componentes que registram:
  - Muitos
- Método usado para registrar:
  - addMouseListener(MouseListener ml)

Interface MouseMotionListener

- Eventos capturados:
  - MouseEvent
- Método(s) definidos:
  - void mouseDragged(MouseEvent e)
  - void mouseMoved(MouseEvent e)
- Componentes que registram:
  - Muitos
- Método usado para registrar:
  - addMouseMotionListener(MouseMotionListener mml)

Exemplo


Demonstrating Mouse Events
Mouse outside window


Demonstrating Mouse Events
Mouse in window


Demonstrating Mouse Events
Moved at [71, 47]


Demonstrating Mouse Events
Pressed at [112, 48]


Demonstrating Mouse Events
Dragged at [177, 56]


Demonstrating Mouse Events
Released at [177, 56]

```
public class MouseTracker extends JFrame
                implements MouseListener,
                           MouseMotionListener {
   private JLabel statusBar;

   public MouseTracker(){
      super("Demonstrating Mouse Events");

      statusBar = new JLabel();
      getContentPane().add( statusBar,
                            BorderLayout.SOUTH );

      this.addMouseListener(this);
      this.addMouseMotionListener(this);

      setSize(275, 100);
      setVisible(true);
   }  //obs: a classe não acaba aqui!
```

```
// Tratadores de MouseListener
public void mouseClicked( MouseEvent e ){
  statusBar.setText( "Clicked at [" + e.getX() +
                     ", " + e.getY() + "]" );
}
public void mousePressed( MouseEvent e ){
  statusBar.setText( "Pressed at [" + e.getX() +
                     ", " + e.getY() + "]" );
}
public void mouseReleased( MouseEvent e ){
   statusBar.setText( "Released at [" + e.getX() +
                      ", " + e.getY() + "]" );
}
public void mouseEntered( MouseEvent e ){
  statusBar.setText( "Mouse in window" );
}
public void mouseExited( MouseEvent e ){
  statusBar.setText( "Mouse outside window" );
}
```

```
    // Tratadores de MouseMotionListener
    public void mouseDragged( MouseEvent e ){
        statusBar.setText( "Dragged at [" + e.getX() +
                           ", " + e.getY() + "]" );
    }
    public void mouseMoved( MouseEvent e ){
        statusBar.setText( "Moved at [" + e.getX() +
                           ", " + e.getY() + "]" );
    }
    public static void main( String args[] ){
        MouseTracker app = new MouseTracker();
        app.addWindowListener(
            new WindowAdapter() {
                public void windowClosing( WindowEvent e ){
                    System.exit( 0 );
                }
            }
        );
    }
}
```

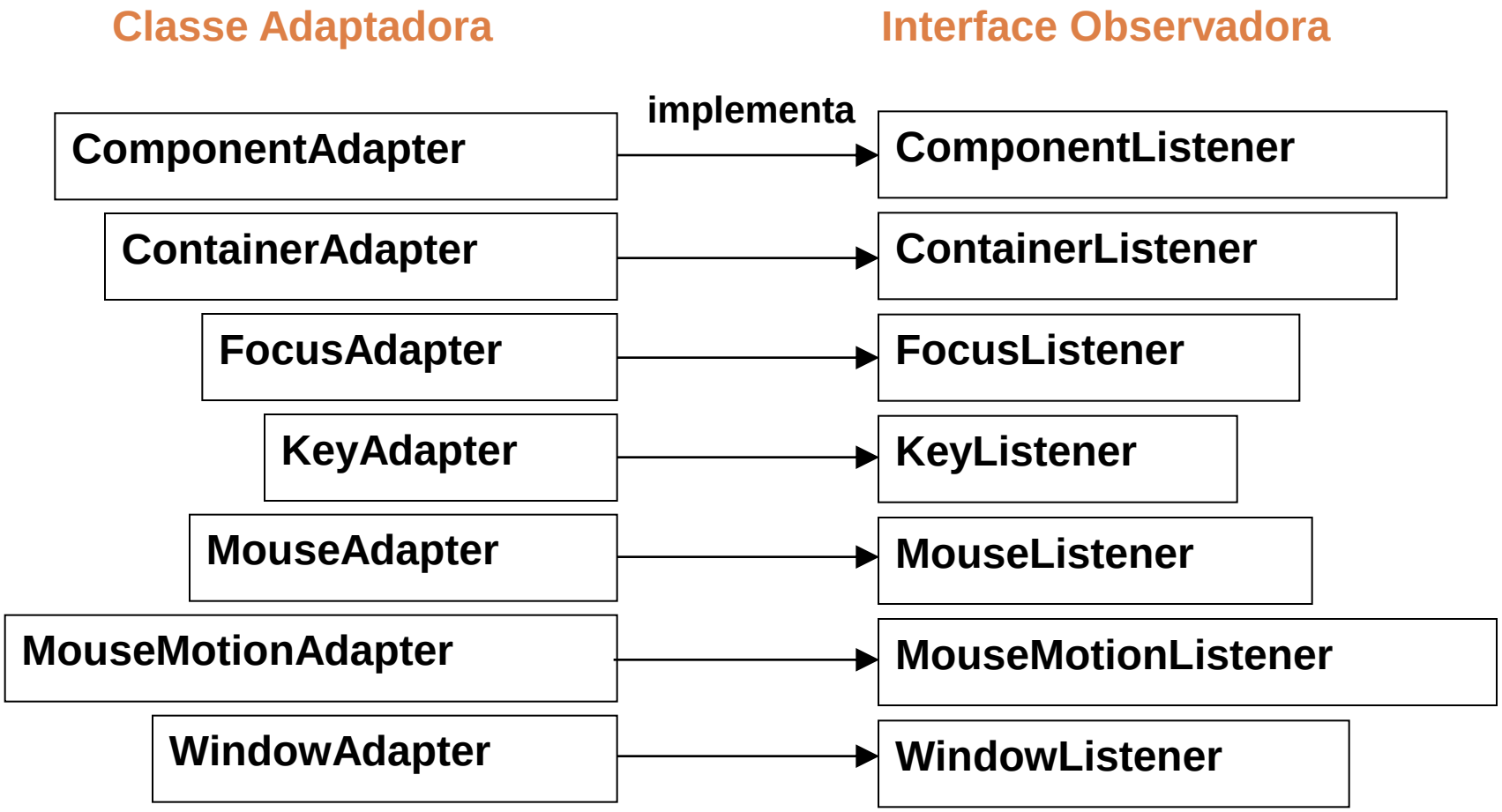
Classe MouseEvent

- Alguns métodos:
  - int getX(): Devolve a coordenada x onde o MouseEvent ocorreu.
  - int getY(): Devolve a coordenada y onde o MouseEvent ocorreu.
  - int getClickCount(): informa quantas vezes o mouse foi clicado.
  - boolean isMetaDown(): Botão da direita do mouse foi usado para clicar.
  - boolean isAltDown(): Botão do meio do mouse foi usado para clicar.

Classes Adaptadoras

- Implementam as interfaces observadoras que possuem mais de um método.
- Fornecem uma implementação *default* (vazio) para todos os métodos da interface.
- Localizadas em:
  - java.awt.event
  - javax.swing.event

Classes Adaptadoras



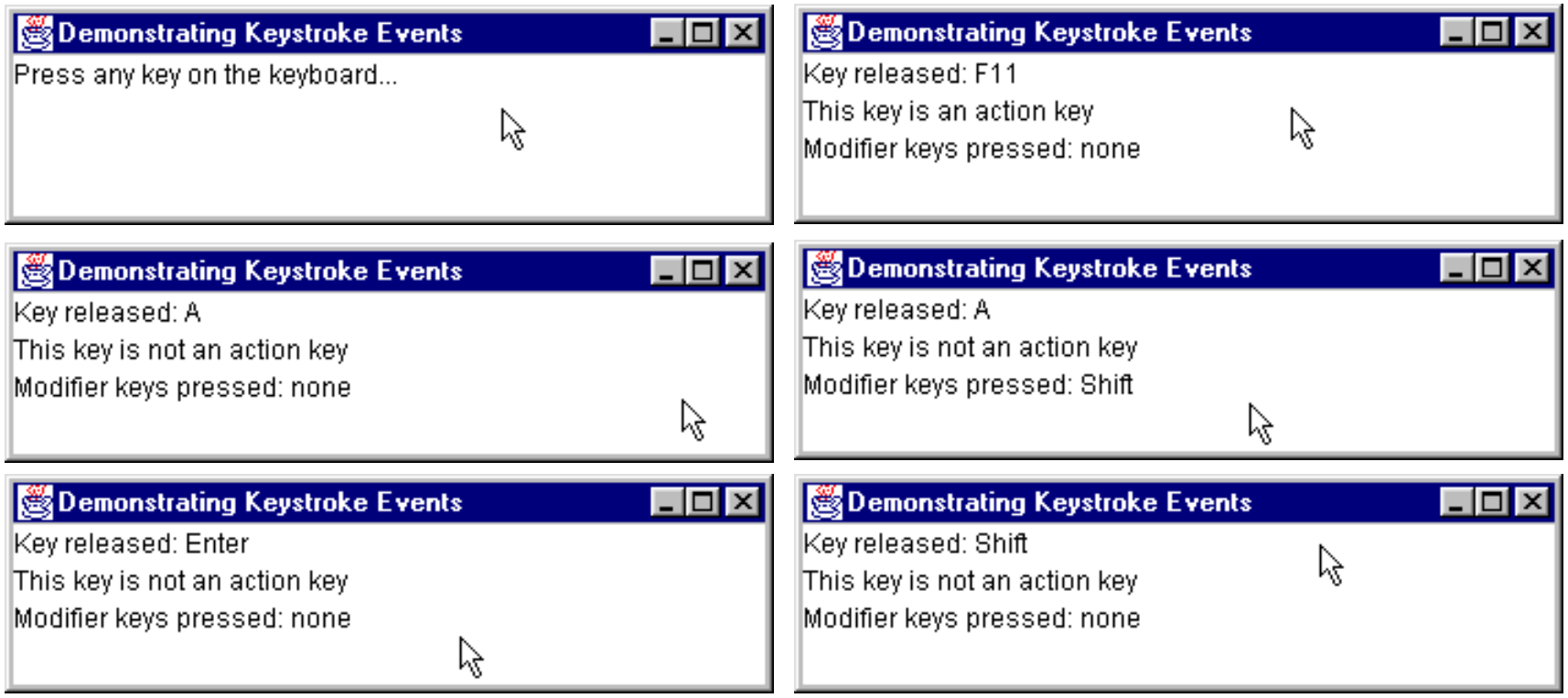
Interface KeyListener

- Eventos capturados:
  - KeyEvent
- Método(s) definidos:
  - void keyPressed(KeyEvent e)
  - void keyReleased(KeyEvent e)
  - void keyTyped(KeyEvent e)
- Componentes que registram:
  - Muitos
- Método usado para registrar:
  - addKeyListener(KeyListener mml)

Interface KeyListener

- As teclas são divididas em 3 categorias:
  - Teclas de ação
    - *Setas*, **Home**, **End**, **PageUp**, **PageDn**, *teclas de funções*, **Num Lock**, **Print Screen**, **Scroll Lock**, **Caps Lock** e **Pause**.
  - Teclas comuns
    - *Letras*, *Números*, *Símbolos*, *Espaço*, **Enter**, **Shift**, **Alt**, **Ctrl**
  - Modificadoras
    - **Shift**, **Alt** e **Ctrl**
- keyTyped() só é chamado para as teclas comuns
- keyPressed() e keyReleased() são chamados para todos os tipos de teclas.

Exemplo


Demonstrating Keystroke Events
Press any key on the keyboard...
Demonstrating Keystroke Events
Key released: F11
This key is an action key
Modifier keys pressed: none
Demonstrating Keystroke Events
Key released: A
This key is not an action key
Modifier keys pressed: none
Demonstrating Keystroke Events
Key released: A
This key is not an action key
Modifier keys pressed: Shift
Demonstrating Keystroke Events
Key released: Enter
This key is not an action key
Modifier keys pressed: none
Demonstrating Keystroke Events
Key released: Shift
This key is not an action key
Modifier keys pressed: none

```
public class KeyDemo extends JFrame
             implements KeyListener {
   private String line1 = "", line2 = "";
   private String line3 = "";
   private JTextArea textArea;

   public KeyDemo(){
      super( "Demonstrating Keystroke Events" );

      textArea = new JTextArea( 10, 15 );
      textArea.setText("Pressione qualquer tecla..." );
      textArea.setEnabled( false );
      addKeyListener( this );
      getContentPane().add( textArea );

      setSize( 350, 100 );
      setVisible(true);
   }  //obs: a classe não acaba aqui!
```

```
//continuação...
public void keyPressed( KeyEvent e ){
  line1 = "Key pressed: " +
            e.getKeyText( e.getKeyCode() );
  setLines2and3( e );
}

public void keyReleased( KeyEvent e ){
  line1 = "Key released: " +
            e.getKeyText( e.getKeyCode() );
  setLines2and3( e );
}

public void keyTyped( KeyEvent e ){
  line1 = "Key typed: " + e.getKeyChar();
  setLines2and3( e );
}
//continua...
```

```
 //continuação...
 public static void main( String args[] ){
   KeyDemo app = new KeyDemo();

   app.addWindowListener(
     new WindowAdapter() {
        public void windowClosing( WindowEvent e )
        {
           System.exit( 0 );
        }
     }
  );
 }
} //fim da classe
```

Gerenciadores de Layout

- Objetos responsáveis pelo posicionamento e redimensionamento de componentes num container.
- Gerenciadores mais usados:
  - **`FlowLayout`**
  - **`BorderLayout`**
  - **`GridLayout`**
- Herdam de **`Object`** e implementam **`LayoutManager`**

FlowLayout

- Elementos dispostos sequencialmente.
- Construtores:
  - **`FlowLayout()`**: centraliza componentes.
  - **`FlowLayout(int align)`**
- Constantes de classe:
  - **`int FlowLayout.LEFT, FlowLayout.RIGHT,FlowLayout.CENTER`**
- Método que adiciona componentes:
  - **`add(Component c)`**

BorderLayout

- Elementos dispostos em 5 regiões.
- Constantes de classe:
  - `String BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST, BorderLayout.CENTER`
- Método que adiciona componentes:
  - `add(Component c, String pos)`

GridLayout

- Elementos dispostos em linhas e colunas
- Construtores:
  - `GridLayout():uma linha e coluna`
  - `GridLayout(int l, int c)`
- Método que adiciona componentes:
  - `add(Component c)`, adiciona por linha e por coluna

Painéis

- A classe **`JPanel`** é um **`Container`**, portanto, pode ter seu próprio LM.
- Interfaces mais complexas podem usar **`JPanel`**s para compor o layout.
- Exemplo:

```
JPanel painel = new JPanel();
painel.setLayout( new FlowLayout() );
painel.add(umBotao);
this.getContentPane().add(painel);
```

Bibliografia

- **Arnold**, Ken e **Gosling**, James. *Programando em Java*. Makron Books. 1997
- **Campione**, Mary e **Walrath**, Kathy. *The Java Tutorial - Object Oriented Programming for the Internet*. Addyson Wesley. 1996
- **Deitel**, Harvey e **Deitel**, Paul. *Java: How to Program*. 3ª edição. Prentice Hall. 1999
- **Flanagan**, David. *Java in a Nutshell – A Desktop Quick Reference*. O'Reilly. 1999
- Material o Profº Frederico Costa Guedes Pereira do CEFET/PB