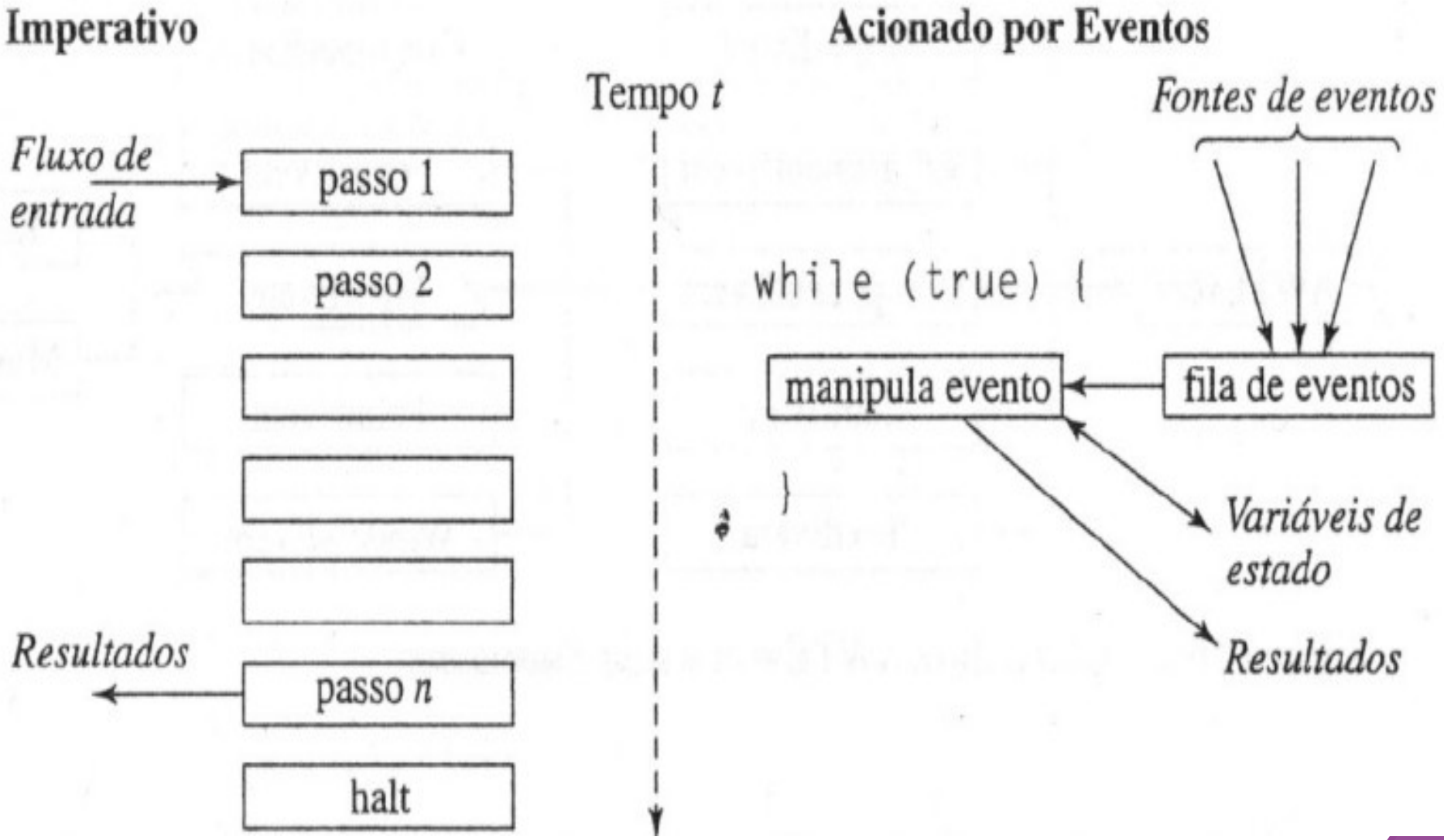
Aplicações de Linguagem de Programação Orientada a Objeto

Eventos

Professora Sheila Cáceres

Programação Orientada a Eventos

- No modelo imperativo tradicional, o programador determina a ordem de entrada dos dados.
- Em contraste, os programas orientados a eventos **não controlam** a sequência na qual ocorrem eventos de entrada; em vez disso, eles são escritos para **reagir** a qualquer sequência razoável de eventos.
- O sistema em tal paradigma é programado em sua base em um **laço de repetição** de eventos, que **recebem** repetidamente **informação** para processar e disparam uma função de **resposta** de acordo com o evento.
- Ou seja, as entradas governam a sequência de operações executadas pelo programa.

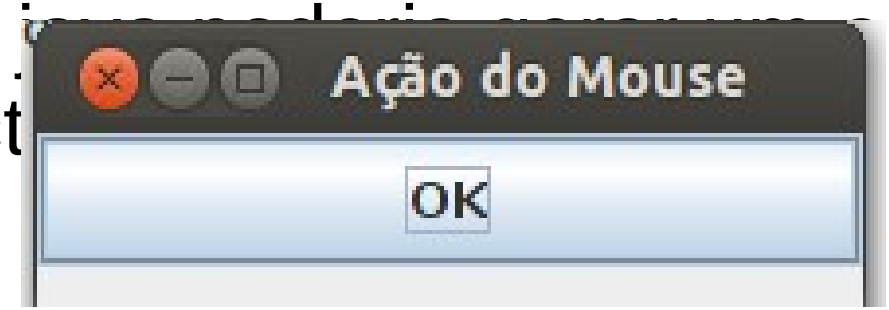

Imperativo
Fluxo de entrada
passo 1
passo 2
Resultados
passo n
halt
Acionado por Eventos
Tempo t
Fontes de eventos
while (true) {
manipula evento
fila de eventos
}
Variáveis de estado
Resultados

Programas acionados por eventos

- Um programa acionado por eventos tipicamente **não tem um ponto de parada predeterminado**, como chegar ao fim do arquivo lendo os dados.
- A entrada de um programa acionado por eventos vem de **fontes de eventos** autônomas distintas que podem ser sensores de um robô ou botões em uma aplicação interativa.
- Esses eventos ocorrem de forma assíncrona e cada um deles entra em uma fila de eventos para serem tratados
- A medida que o tempo passa, um laço de controle recupera o próximo evento dessa fila e o manipula.

Fontes de Eventos (disparadores de eventos)

- São aqueles componentes que accionam os eventos (ex em java: JButton, JLabel, JTextField).
- Em java os objetos que podem ser fontes de eventos são subclasses (ou filhos) da classe JComponent.
- Cada **fonte de evento** em uma interação com o usuário **gera** um objeto **evento**.
- Exemplo, um botão pode ser a fonte de um evento e em j bjeto evento da classe Act



Eventos

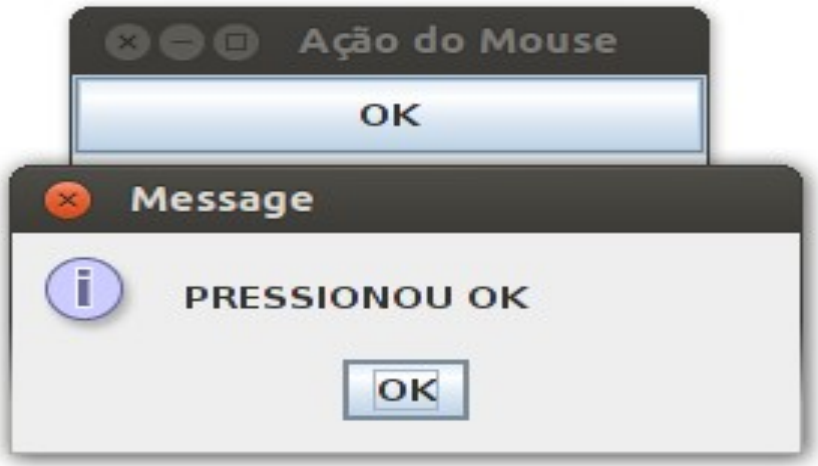
- São as ações que o usuário realiza sobre o programa.
- Exemplos de eventos típicos são
  - Clique do mouse sobre um botão
  - Escolher uma opção de um menu
  - Escrever uma caixa de texto
  - Mover o mouse
- Geralmente cada vez que se produze um evento se cria um objeto que contem informações sobre este evento.
- Em java os tipos de **eventos** que podem ocorrer são definidos pelas subclasses da classe java.Awt.Event.

Listener (Ouvinte)

- Servem para reconhecer (ouvir) a ocorrências de eventos particulares acontecendo sobre fontes de eventos para poder realizar alguma ação.
- Uma única fonte pode ter múltiplos listeners.
- Um único listener pode responder a múltiplas sources.
- As classes que detetam o evento implementam da interface Listener e portanto precisam implementar **todos** os métodos declarados na interface.
- **Adaptadores:** São classes por default que implementam os métodos dos listeners. Isto permite herdar delas implementando **somente os métodos desejados.**
- Exemplo: para que um programa possa detectar a ocorrência da seleção de um botão, o botão precisa chamar seu método *addActionListener*. Se isso não for feito, os

Métodos manipuladores (Handlers)

- Para responder aos eventos detectados pelos listeners precisamos implementar métodos especiais chamados de handlers (manipuladores).
- Em java implementamos os handlers **dentro das classes listeners**



Alguns eventos, listeners e operações em Java

Evento	Listener	Handlers
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
MouseEvent	MouseListener	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
	MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
TextEvent	TextListener	textValueChanged(TextEvent)
WindowEvent	WindowListener	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)

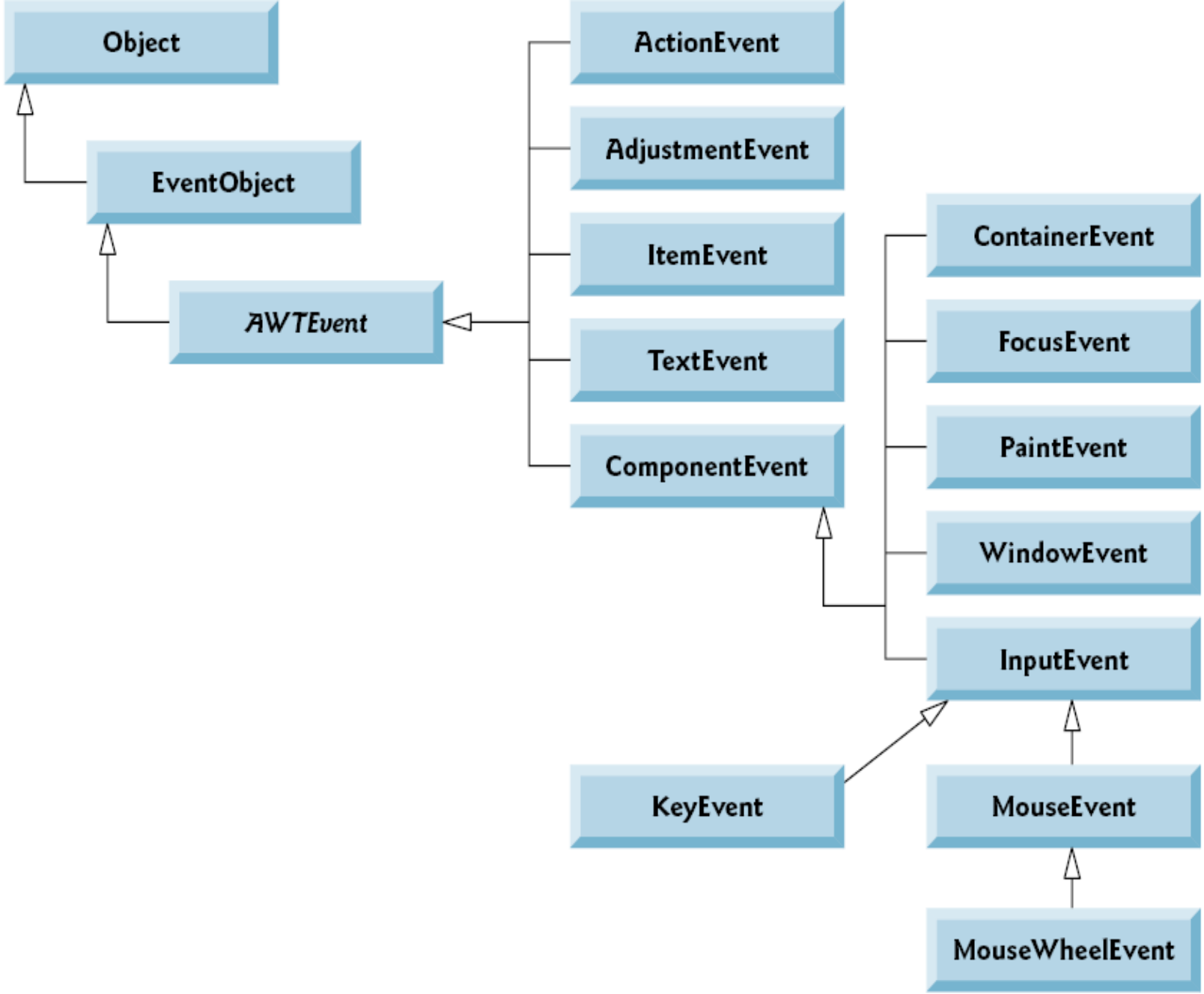


Figura 14.11 | Algumas classes de evento do pacote `java.awt.event`.

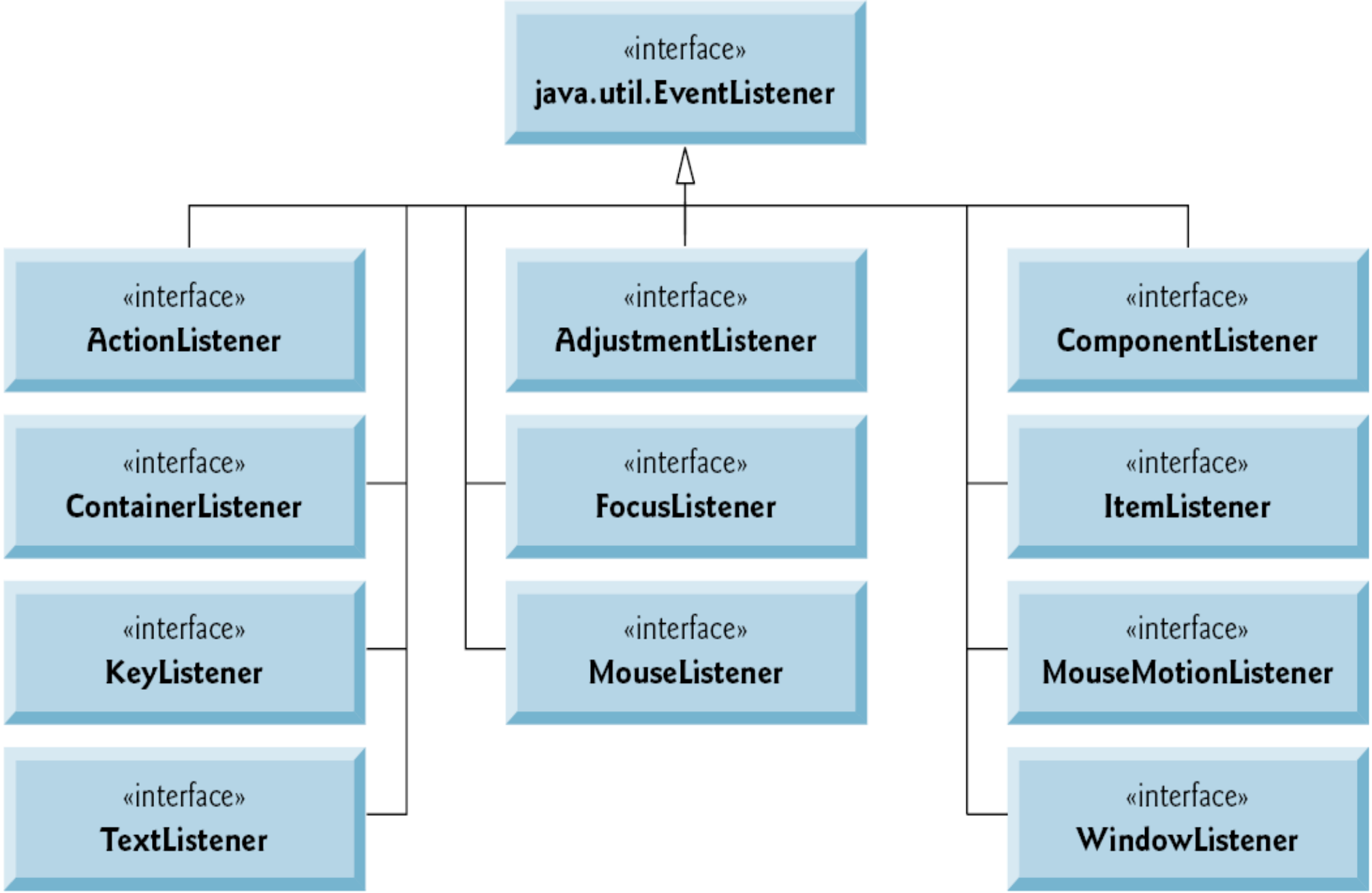
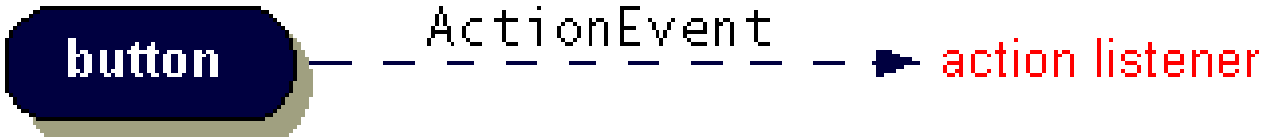
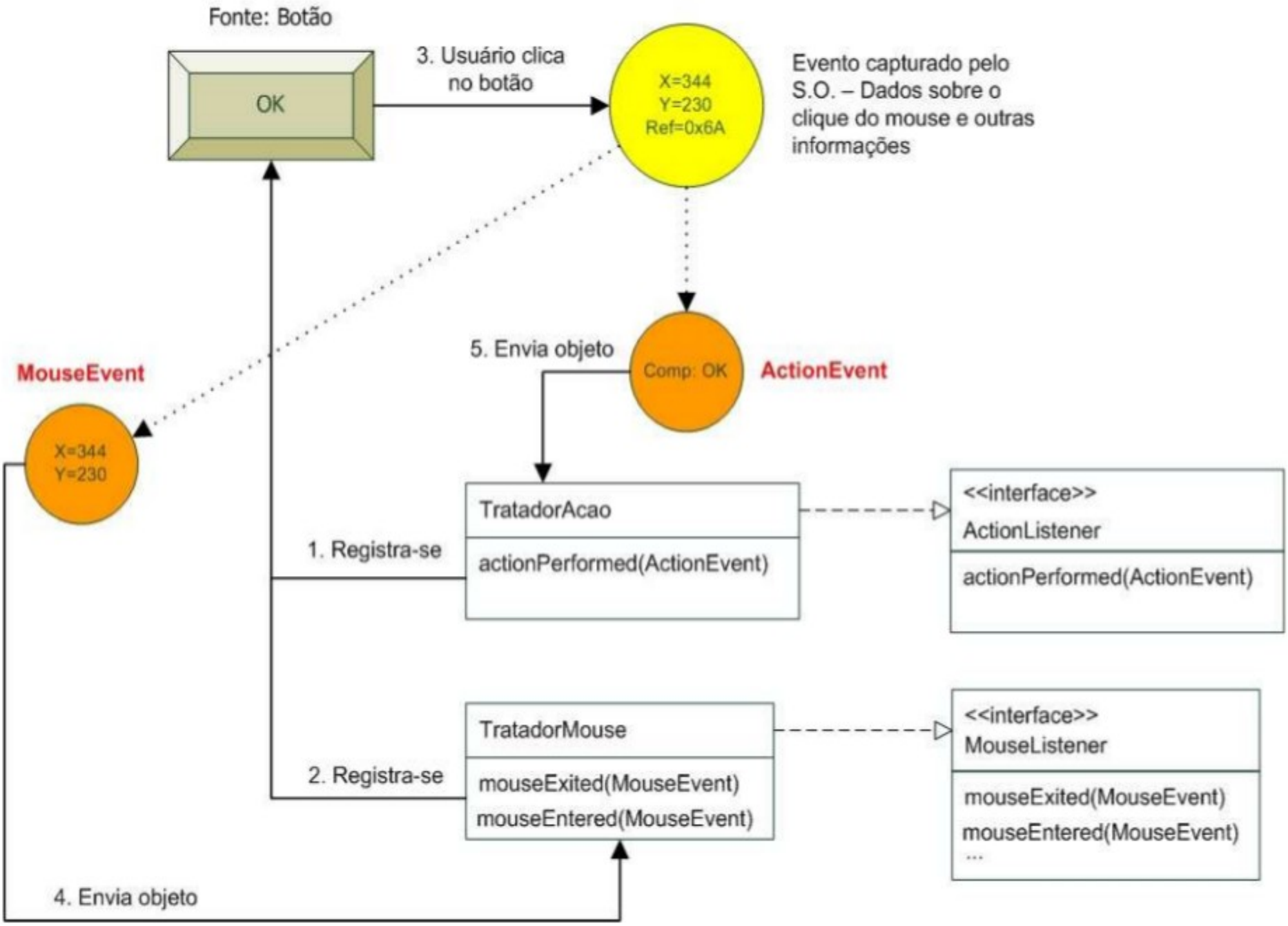


Figura 14.12 | Algumas interfaces listener de eventos comuns do pacote `java.awt.event`.

Manipulando eventos (Exemplo)

- ActionEvent (evento de enter em botão):
  - a aplicação deve criar um objeto que implementa a interface ActionListener e
  - registrar esse objeto como um “ouvinte” (listener) desse botão usando o método addActionListener
  - Quando o botão é pressionado, ele dispara um evento, o que resulta na execução do método actionPerformed do objeto ouvinte.
  - O argumento para esse método é um objeto ActionEvent que informa acerca do evento e de sua fonte.




Fonte: Botão
OK
3. Usuário clica no botão
X=344
Y=230
Ref=0x6A
Evento capturado pelo S.O. – Dados sobre o clique do mouse e outras informações
5. Envia objeto
Comp: OK
ActionEvent
MouseEvent
X=344
Y=230
TratadorAcao
actionPerformed(ActionEvent)
1. Registra-se
<<interface>>
ActionListener
actionPerformed(ActionEvent)
TratadorMouse
mouseExited(MouseEvent)
mouseEntered(MouseEvent)
2. Registra-se
<<interface>>
MouseListener
mouseExited(MouseEvent)
mouseEntered(MouseEvent)
...
4. Envia objeto

Exemplo de fonte de aplicação que trata eventos:

```
public class Aplicação{
    public static void main(String argumentos[]){
        Janela          jan  = new Janela();
        TratEventosJan trat = new TratEventosJan();
        jan.addWindowListener(TratEventos());
        jan.show();
    }
}

class Janela extends Frame{
    public Janela(){
        setBackground(Color.blue);
        add("Center", new Label("Janela da aplicação"));
    }
}

class TratEventosJan extends WindowAdapter{
    public void windowClosing(WindowEvent evento){
        System.exit(0);
    }
}
```

A primeira classe é a que define a aplicação

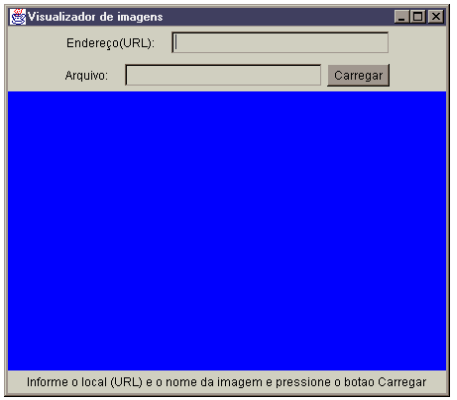
A segunda classe é a que define a interface (o tipo de janela) da aplicação

A última classe é a que define o tratador de eventos de janela da aplicação

Exemplo de fonte de aplicação que trata eventos:

```
public class Aplicação{
   public static void main(String argumentos[]){
      Janela          jan  = new Janela();
      TratEventosJan trat = new TratEventosJan();
      jan.addWindowListener(trat);
      jan.show();
   }
}

class Janela extends Frame{
   public Janela(){
      setBackground(Color.blue);
      add("Center", new Label("Janela da aplicação"));
   }
}

class TratEventosJan extends WindowAdapter{
   public void windowClosing(WindowEvent evento){
      System.exit(0);
   }
}
```

A primeira classe é responsável por criar a janela da aplicação.



A janela é criada de acordo com a classe de janela definida!

Exemplo de fonte de aplicação que trata eventos:

```
public class Aplicação{
    public static void main(String argumentos[]){
        Janela          jan  = new Janela();
        TratEventosJan trat = new TratEventosJan();
        jan.addWindowListener(trat);
        jan.show();
    }
}

class Janela extends Frame{
    public Janela(){
        setBackground(Color.blue);
        add("Center", new Label("Janela da aplicação"));
    }
}

class TratEventosJan extends WindowAdapter{
    public void windowClosing(WindowEvent evento){
        System.exit(0);
    }
}
```

Em seguida, é criado o objeto tratador de eventos.

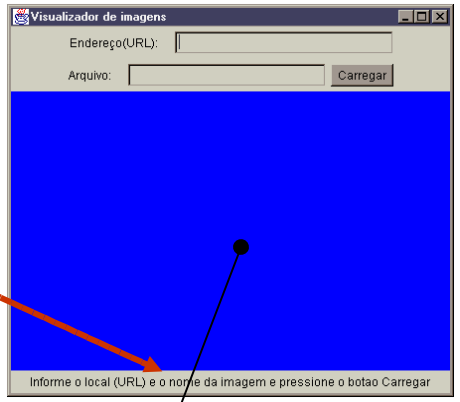
trat
(tratador de eventos)

Ele é criado de acordo com a classe definida para tratar eventos!

Exemplo de fonte de aplicação que trata eventos:

```
public class Aplicação{
   public static void main(String argumentos[]){
      Janela          jan  = new Janela();
      TratEventosJan trat = new TratEventosJan();
      jan.addWindowListener(trat);
      jan.show();
   }
}


class Janela extends Frame{
   public Janela(){
      setBackground(Color.blue);
      add("Center", new Label("Janela da aplicação"));
   }
}


class TratEventosJan extends WindowAdapter{
   public void windowClosing(WindowEvent evento){
      System.exit(0);
   }
}
```

Finalmente, o tratador de eventos é adicionado à janela:



trat
(tratador de eventos)

Podemos adicionar o listener em outros lugares

Assim, todo evento de janela gerado nesta janela é capturado e tratado pelo objeto tratador de eventos.

Como e onde colocar um listener

- Podemos colocar um listener de 4 maneiras:
  - A) Em uma classe externa (exemplo anterior)
  - B) Em uma classe interna
  - C) Em uma classe interna anônima
  - D) Como parte da própria classe

Onde colocar um Listener

B) Podemos colocar em uma ***classe interna***

```
public class Outer {
  private class Inner implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      ...
    }
  }

  public Outer() {
    JButton myButton = new JButton();
    myButton.addActionListener(new Inner());
  }
}
```

Onde colocar um ActionListener

C) Podemos colocar em uma ***classe interna anônima***

```
public class Outer {
  public Outer() {
    JButton myButton = new JButton();
    myButton.addActionListener(
      new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          ...
        }
      }
    );
  }
}
```

Onde colocar um ActionListener

```
public class GUIDemo extends JFrame implements ActionListener {
     protected int numCl = 0;
     protected JButton btnClique;
     protected JLabel lblNumClicks = null;

     public GUIDemo() { super("Demonstrando eventos");initialize(); }
     protected void initialize() {
          lblNumClicks = new JLabel("Num. clicks=" + numCl);
          btnClique = new JButton("Clique em mim!");
          btnClique.addActionListener(this);
          this.setLayout(new BorderLayout());
          this.add(btnClique, BorderLayout.NORTH);
          this.add(lblNumClicks, BorderLayout.SOUTH);
          this.pack();
     }
     public void actionPerformed(ActionEvent e) {
          numCl++;
          lblNumClicks.setText("Num Clicks:" + numCl);
     }
}
```

```
public class TestaGUIDemo {
     public static void main(String[] args) {
          GUIDemo gui=new GUIDemo();
          gui.setSize(200, 75);
          gui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          gui.setVisible(true);
     }
}
```

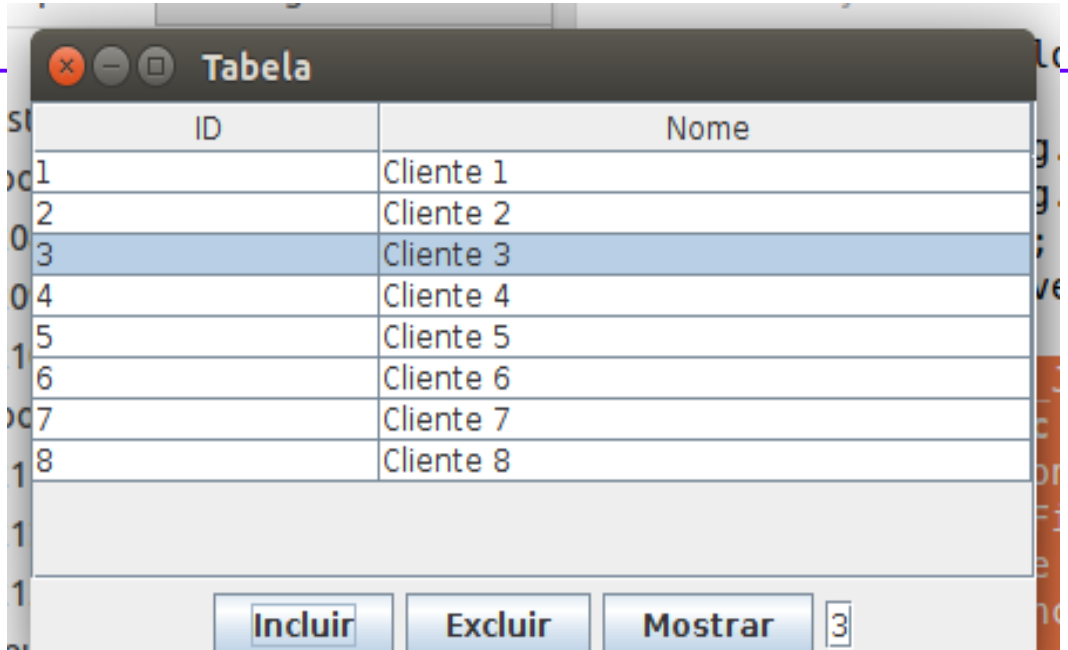
D) Podemos colocá-la como parte da própria classe

Mais componentes e seus eventos

JTable

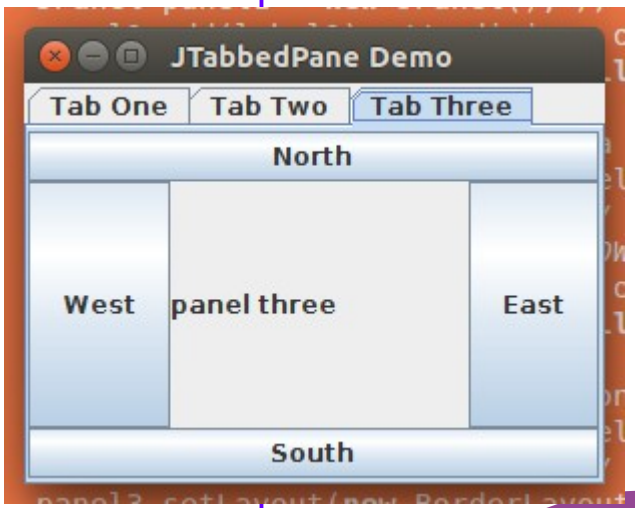
```
public class Ex14_JTable extends JFrame implements ActionListener {
	private static final long serialVersionUID = 1L;
	private JButton btnExcluir, btnIncluir, btnMostrar;
	private JTextField texto;
	private JTable tabela;
	private int incCod = 0;

	public Ex14_JTable() {
		setTitle("Tabela");
		setSize(441, 300);
		setLocation(100, 50);
		Container P = getContentPane();
		P.setLayout(new BorderLayout());
		tabela = new JTable();
		tabela.setModel(new DefaultTableModel(new Object[][] {}, new String[] {
"ID", "Nome" }));

		// define largura das colunas -- permite ou não, ajustar a largura
		tabela.getColumnModel().getColumn(0).setPreferredWidth(20);
		tabela.getColumnModel().getColumn(0).setResizable(false);
		tabela.getColumnModel().getColumn(1).setPreferredWidth(150);
		tabela.getColumnModel().getColumn(1).setResizable(true);

		JScrollPane rolagemTabela = new JScrollPane(tabela);
		P.add(rolagemTabela, "Center");

		// criação do Painel de baixo
		JPanel PTabSul = new JPanel();
		btnIncluir = new JButton("Incluir");
		PTabSul.add(btnIncluir);
		btnExcluir = new JButton("Excluir");
		PTabSul.add(btnExcluir);
		btnMostrar = new JButton("Mostrar");
		PTabSul.add(btnMostrar);
		texto = new JTextField("Nome Selecionado");
		P.add(PTabSul, "South");
		btnExcluir.addActionListener(this);
		btnIncluir.addActionListener(this);
		btnMostrar.addActionListener(this);
		PTabSul.add(texto);
	}
```

```
public void actionPerformed(ActionEvent evt) {
            Object origem = evt.getSource();
            if (origem == btnIncluir) {
                  DefaultTableModel dtm = (DefaultTableModel) tabela.getModel();
                  dtm.addRow(new Object[] { new Integer(++incCod),"Cliente " + incCod });
            }
            if (origem == btnExcluir) {
                  int linhas[] = tabela.getSelectedRows();
                  DefaultTableModel dtm = (DefaultTableModel) tabela.getModel();
                  for (int i = (linhas.length - 1); i >= 0; i--)
                        dtm.removeRow(linhas[i]);
            }
            if (origem == btnMostrar) {
                  if (tabela.getSelectedRow() >= 0)
                        texto.setText(tabela.getValueAt(tabela.getSelectedRow(),
                                    tabela.getSelectedColumn()).toString());
            }
      }

      public static void main(String args[]) {
            Ex14_JTable fr = new Ex14_JTable();
            fr.setVisible(true);
      }
}
```



JTabbedPane

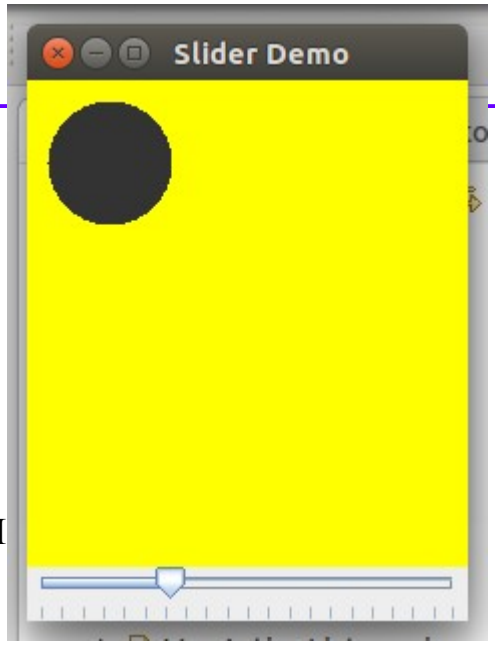
```
public class Ex15_JTabbedPane extends JFrame {
      public Ex15_JTabbedPane() {
            super("JTabbedPane Demo ");
            JTabbedPane tabbedPane = new JTabbedPane(); // cria JTabbedPane

            // configura o pane11 e o adiciona ao JTabbedPane
            JLabel label1 = new JLabel("panel one", SwingConstants.CENTER);
            JPanel panel1 = new JPanel(); // cria o primeiro painel
            panel1.add(label1); // adiciona o rótulo ao painel
            tabbedPane.addTab("Tab One", null, panel1, "Primeiro Panel");

            // configura panel2 e o adiciona ao JTabbedPane
            JLabel label2 = new JLabel("panel two", SwingConstants.CENTER);
            JPanel panel2 = new JPanel(); // cria o segundo panel
            panel2.setBackground(Color.YELLOW); // configura fundo como amarelo
            panel2.add(label2); // adiciona o rótulo ao painel
            tabbedPane.addTab("Tab Two", null, panel2, "Segundo Panel");

            // configura o panel3 e o adiciona ao JTabbedPane
            JLabel label3 = new JLabel("panel three");
            JPanel panel3 = new JPanel(); // cria o terceiro painel
            panel3.setLayout(new BorderLayout()); // utiliza o borderlayout
            panel3.add(new JButton("North"), BorderLayout.NORTH);
            panel3.add(new JButton("West"), BorderLayout.WEST);
            panel3.add(new JButton("East"), BorderLayout.EAST);
            panel3.add(new JButton("South"), BorderLayout.SOUTH);
            panel3.add(label3, BorderLayout.CENTER);
            tabbedPane.addTab("Tab Three", null, panel3, "Terceiro Panel");
            add(tabbedPane); // adiciona JTabbedPane ao frame
      } // fim do construtor JTabbedPaneFrame

      public static void main(String args[]) {
            Ex15_JTabbedPane tabbedPaneFrame = new Ex15_JTabbedPane();
            tabbedPaneFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            tabbedPaneFrame.setSize(250, 200); // configura o tamanho do frame
            tabbedPaneFrame.setVisible(true); // exibe o frame
      } // fim do main
} // fim da classe JTabbedPaneFrame
```



JSlider

```
public class Ex16_JSlider extends JFrame {
      private JSlider diameterJSlider; // controle deslizante p/ selecionar o diâmetro
      private Ex16_OvalPanel myPanel; // painel para desenhar o círculo

      public Ex16_JSlider() {
            super("Slider Demo");
            myPanel = new Ex16_OvalPanel(); // cria painel para desenhar o círculo
            myPanel.setBackground(Color.YELLOW); // configura o fundo como amarelo
            // configura o JSlider para controlar o valor diâmetro
            diameterJSlider = new JSlider(SwingConstants.HORIZONTAL, 0, 200, 10);
            diameterJSlider.setMajorTickSpacing(10); //uma marca de medida a cada 10
            diameterJSlider.setPaintTicks(true); // pinta marcas no controle
                                                          // deslizante
            // registra o ouvinte de evento do JSlider
            diameterJSlider.addChangeListener(new ChangeListener() //classe interna
anônima
            {
                  // trata da alteração de valor do controle deslizante
                  public void stateChanged(ChangeEvent e) {
                        myPanel.setDiameter(diameterJSlider.getValue());
                        System.out.println(diameterJSlider.getValue());
                  } // fim do método stateChanged
            } // fim da classe interna anônima
            ); // fim da chamada para addChangeListener

            add(diameterJSlider, BorderLayout.SOUTH); // adiciona o controle
                                                            // deslizante ao
frame
            add(myPanel, BorderLayout.CENTER); // adiciona o painel ao frame
      } // fim do construtor SliderFrame

      public static void main(String args[]) {
            Ex16_JSlider sliderFrame = new Ex16_JSlider();
            sliderFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            sliderFrame.setSize(220, 270); // configura o tamanho do frame
            sliderFrame.setVisible(true); // exibe o frame
      }
} // fim da class classe SliderFrame
```

```
class Ex16_OvalPanel extends JPanel {
      private int diameter = 10; // diâmetro padrão de 10

      // desenha uma oval do diâmetro especificado
      public void paintComponent(Graphics g) {
            super.paintComponent(g);
            g.fillOval(10, 10, diameter, diameter); // desenha um círculo
      } // fim do método paintComponent

      // valida e configura o diâmetro, então pinta novamente
      public void setDiameter(int newDiameter) {
            // se o diâmetro for inválido , assume o padrão de 10
            diameter = (newDiameter >= 0 ? newDiameter : 10);
            repaint(); // pinta o painel novamente
      } // fim do método setDiameter

      // utilizado pelo gerenciador de layout para determinar o tamanho preferido
      public Dimension getPreferredSize() {
            return new Dimension(200, 200);
      } // fim do método getPreferredSize

      // utilizado pelo gerenciador de layout para determinar o tamanho mínimo
      public Dimension getMinimumSize() {
            return getPreferredSize();
      } // fim do método getMinimumSize
} // fim da classe OvalPanel
```



Bibliografia

Algumas Figuras e/ou slides foram extraídos do material apresentado a seguir:

- Slides do prof Paulo Borba, Orientação a Objetos em Java, Ufpe.
- Slides from S.N. Kamin, D. Mickunas, E. Reingold, “An Introduction to Computer Science Using Java (2nd Edition)”.