Módulo II
Interface com usuário – GUI

UniverCidade - Prof. Ismael H F Santos

Ementa

- Modulo II – Interface com o usuário (GUI)
  - Swing / AWT
  - Gerenciadores de Layout
  - Características e Componentes Principais
  - Orientação a Eventos
  - Aplicações MDI
  - Conclusões

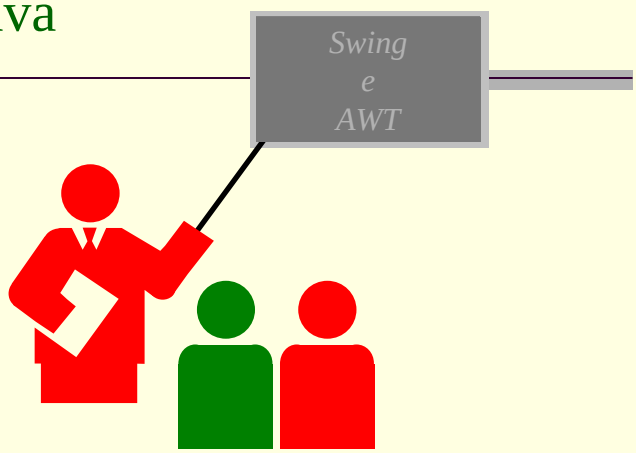
Bibliografia

- *Linguagem de Programação JAVA*
  - *Ismael H. F. Santos, Apostila UniverCidade, 2002*
- *The Java Tutorial: A practical guide for programmers*
  - *Tutorial on-line: http://java.sun.com/docs/books/tutorial*
- *Java in a Nutshell*
  - *David Flanagan, O´Reilly & Associates*
- *Just Java 2*
  - *Mark C. Chan, Steven W. Griffith e Anthony F. Iasi, Makron Books.*
- *Java 1.2*
  - *Laura Lemay & Rogers Cadenhead, Editora Campos*

Livros

- **Core Java 2**, Cay S. Horstmann, Gary Cornell
  - Volume 1 (Fundamentos)
  - Volume 2 (Características Avançadas)
- **Java: Como Programar**, Deitel & Deitel
- **Thinking in Patterns with JAVA**, Bruce Eckel
  - **Gratuito**. http://www.mindview.net/Books/TIJ/

POO-Java



APIs GUI

- *JDK 1.0*
  - *AWT - Abstract Window Toolkit: Write Once Run Anywhere, Applets*
  - *Componentes de peso pesado (peers) que usam recursos do SO*
  - *Look & feel nativo da plataforma final (incompatibilidades !)*
  - *Eventos contidos em ondas:* manipulação de eventos limitada
- *JDK 1.1*
  - *AWT com arquitetura JavaBeans (contrib. Inprise/Borland),* modelo de *delegação de eventos, PME – propriedades, métodos e eventos*
  - *JFC: Java Foundation Classes – esforço Sun, Netscape e IBM para criar uma biblioteca gráfica de interface com usuário para o desenvolvimento de aplicações Java interativas (1997 JavaOne)*

APIs GUI

- *JFC*
  - *Pluggable Look & feel (configurável)*
  - *Acessibilidade: suporte a tecnologias assessoras (Accessibility technologies) tais como leitores de tela e displays em Braille.*
  - *Suporte a Drag & Drop: transferência de dados via clipboard, inclusive entre componentes de uma aplicação Java e uma aplicação nativa.*
  - *Internacionalização*

APIs GUI

- *JFC (cont.)*
  - *Swing GUI Components:*
    - *Swing 1.0: Arquitetura MVC, Diagramadores & Listeners do AWT;*
    - *Swing 1.1: Biblioteca de Componentes: Desktops virtuais (MDI), Objetos Action, Containers aninhados, Bordas compostas, Classes de diálogo padronizadas e customizáveis, Componentes de alto nível como Table, Tree, FileChooser, ColorChooser, Manipulação de texto poderosa, suporte a HTML, operação sem mouse, menus contextuais (popup), Undo: Capacidade genérica de desfazer operações;*
  - *Java 2D API: permite ao desenvolvedor incorporar gráficos 2D, texto e imagens em aplicações e em applets. Impressão de alta qualidade, Double-buffering automático.*

Pacote Padrão **awt**

- Abstração do sistema nativo
- *Toolkit* gráfico e de interface
- Pacote **awt**
  - elementos de interface
  - diagramadores
  - ferramentas gráficas
- Pacote **awt.event**
  - eventos

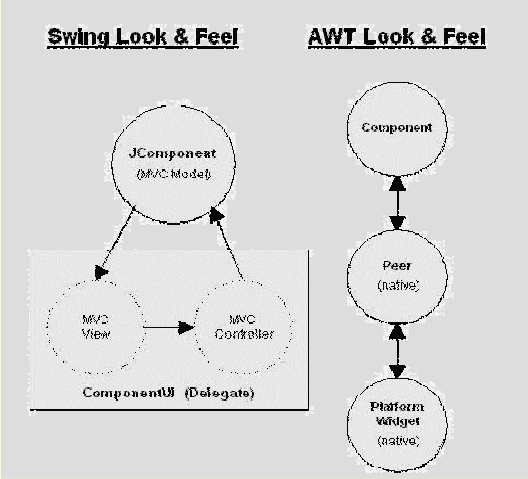
Swing *versus* AWT

- *Swing usa* componentes leves (lightweight), n*on-peer-based GUI toolkit. Os c*omponentes do Swing são implementados sem código nativo
  - maior portabilidade.
  - maior consistência de uso entre plataformas

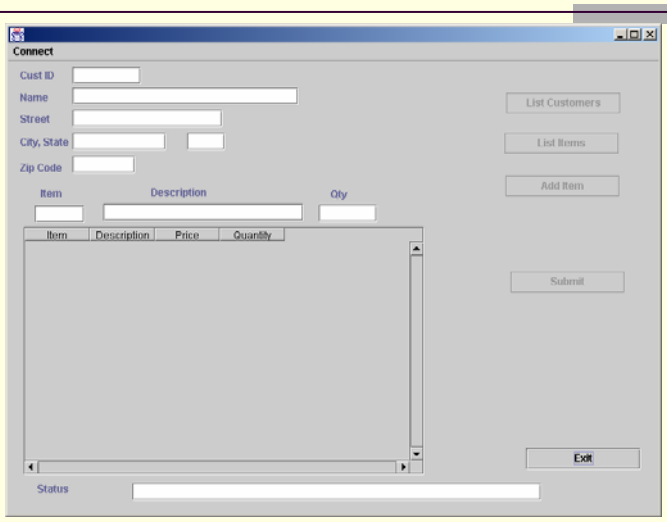
- Pacotes **javax.swing** e **javax.swing.event** são os mais usados

javax.accessibility	javax.swing.plaf
javax.swing.text.html	javax.swing.text.parser
javax.swing.border	javax.swing.plaf.metal
javax.swing.text.rtf	javax.swing.colorchooser
javax.swing.plaf.multi	javax.swing.tree
javax.swing.event	javax.swing.table
javax.swing.undo	javax.swing.filechooser
javax.swing.text	javax.swing javax.swing.plaf.basic

Look & Feel Plugável



Exemplo de aplicação Swing



Construção de uma aplicação Swing

- De uma forma geral, uma aplicação Swing contém os seguintes passos.
  1. Importação dos pacotes Swing
  2. Seleção da aparência ("look & feel")
  3. Definição do contêiner de mais alto nível
  4. Definição dos componentes gráficos
  5. Adição dos componentes a um container
  6. Adição de bordas em componentes
  7. Manipulação de eventos

Importando os pacotes Swing

- A linha a seguir importa o pacote principal para aplicações Swing:

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

- A maioria das aplicações Swing também precisam de dois pacotes AWT:

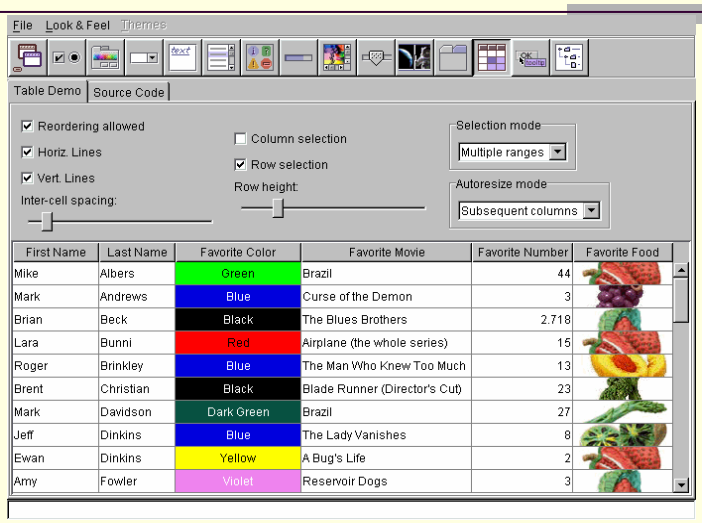
```
import java.awt.*;
import java.awt.event.*;
```

Selecionando o “look & feel“

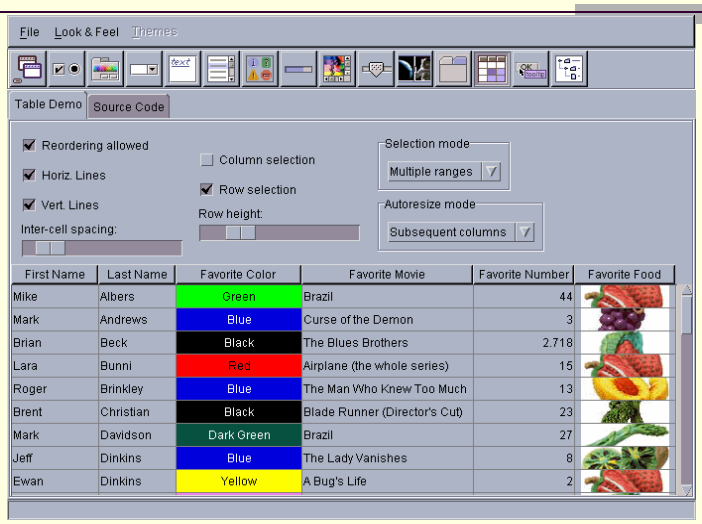
- Swing permite que o programador escolha a aparência (look & feel) mas apropriada através do método setLookAndFeel. Esse método recebe como argumento um nome inteiramente qualificado de uma subclasse de LookAndFeel.
- Exemplos:

```
try {
    UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName());
                .... OU ....
    UIManager.setLookAndFeel(
            "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
 } catch (Exception e) { }

 JFrame.setDefaultLookAndFeelDecorated(true);

 # Swing properties
 swing.defaultlaf=com.sun.java.swing.plaf.windows.WindowsLookAndFeel
```

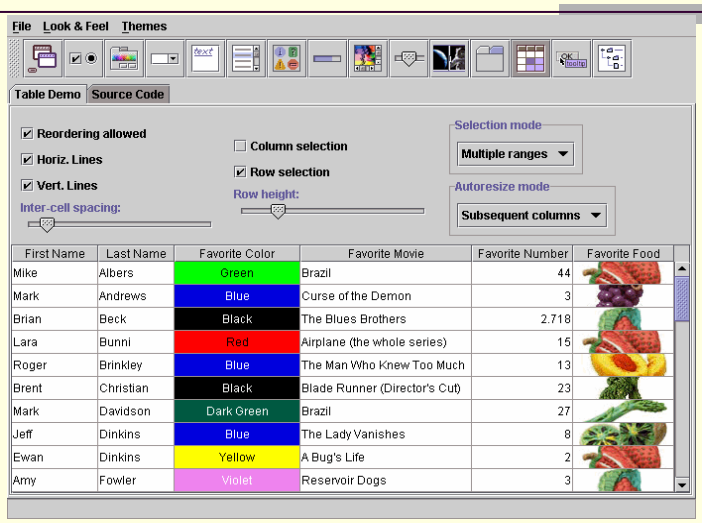

Windows Look and Feel



Motif Look and Feel



Java Look and Feel

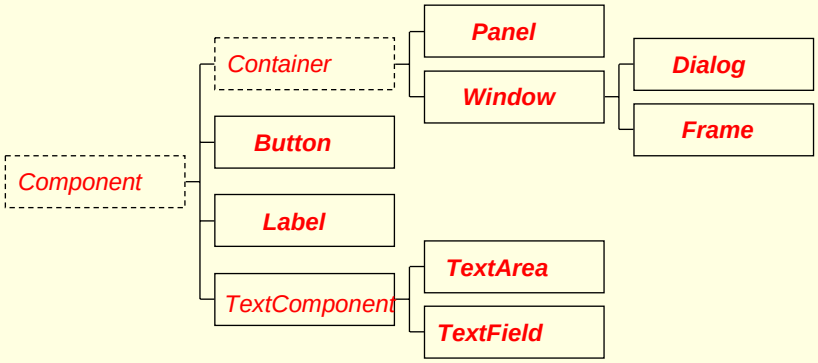


Elementos de Interface AWT

- *Components e Containers*
- **java.awt.Component**
  - base de todos os componentes
  - serializável
  - abstrata
- **java.awt.Container**
  - estende **Component**
  - abstrata

Classes do pacote java.awt

- Hierarquia Básica de Classes de java.awt



Pacote java.awt

- Class Component
  - Modela um elemento de interface.
  - Define métodos, a princípio, comuns a qualquer elemento de interface.
  - Métodos:

```
public void setForeground(Color c)
public void setEnabled(boolean b)
public Container getParent()
public void addMouseListener(MouseListener l)
```


A classe Container

- Class Container
  - Modela um objeto de interface que pode conter outros objetos, um agrupador. Fornece métodos para adicionar e remover componentes e para trabalhar com layout.
  - Um contêiner pode conter outros contêineres (porque todo contêiner é um componente)
- Métodos:

```
public Component add(Component comp)
public void add(Component comp, Object
  constraint)
public void remove(Component comp)
public boolean isAncestorOf(Component c)
public Component[ ] getComponents()
public LayoutManager getLayout()
void setLayout( LayoutManager mgr )
```

Containers Concretos de java.awt

- Panel
  - Representa um grupo de elementos
  - Deve ser incluído em outro *container*
  - Usado para estruturar a interface
- Frame
  - Estende java.awt.Window
  - Representa uma janela
  - Possui título e borda
  - Pode possuir menu
- Dialog

Exemplo

```
import java.awt.*;

public class Mundo {
  public static void main(String[] args) {
    Frame janela = new Frame("Mundo");
    Label mensagem = new Label("Olá Mundo!");
    janela.add(mensagem);
    janela.pack();
    janela.setVisible(true);
  }
}
```

Componentes Concretos java.awt

- Botão
  - Modela um botão

```
public void setLabel(String label)
public void setActionCommand(String command)
public void addActionListener(ActionListener l)
public void removeActionListener(ActionListener l)
```

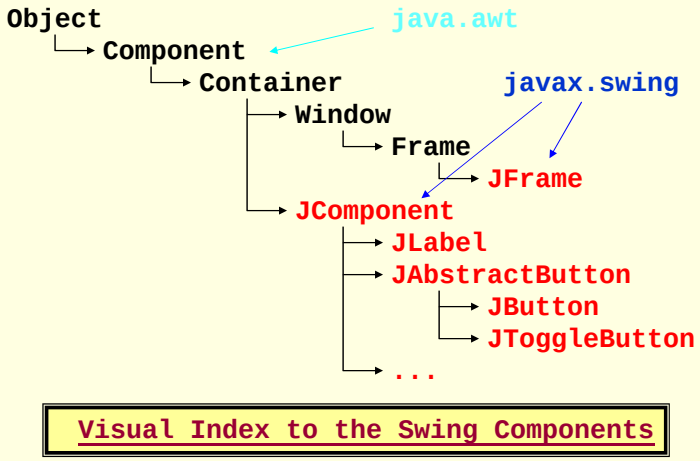
- *Label*
  - Modela um texto não editável

```
public void setText(String text)
public void setAlignment(int align)
```


Componentes Concretos java.awt

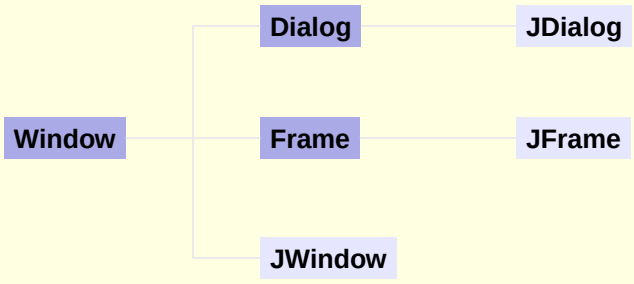
- Texto
- *CheckBox*
- Lista
- Lista *dropdown*
- *Canvas*

Classes do pacote javax.swing



Visual Index to the Swing Components

Classes de Janelas javax.swing



A classe JComponent

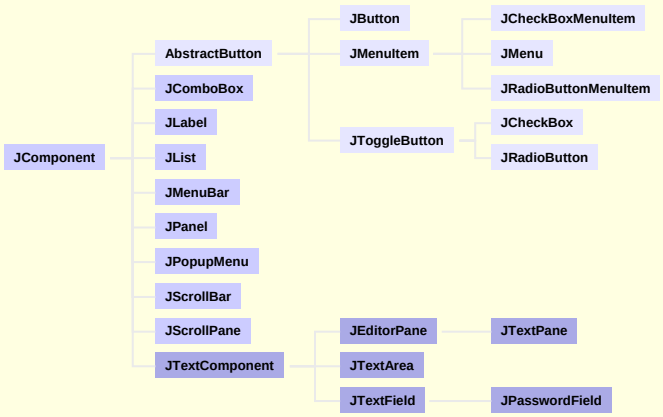
- A classe JComponent é a superclasse de todos os componentes Swing.
- Os objetos JButton, JCheckbox, e JTextField são todos exemplos de objetos das subclasses de JComponent.
- A classe JComponent é uma subclasse direta da classe java.awt.Container que, por sua vez, é uma subclasse direta de java.awt.Component

Classe **JComponent**

- Superclasse de muitos elementos do Swing, disponibiliza métodos para controlar:
  - *Tool tips*
  - Bordas
  - Propriedades
- Métodos de JComponent

```
void setToolTipText(String text)
String getToolTipText()
void setBorder(Border border)
Border getBorder()
final void putClientProperty(Object key, Object val)
final Object getClientProperty(Object key)
```

Hierarquia JComponent



A classe JComponent (cont.)

- Classes definidas no pacote javax.swing e subclasses (diretas ou indiretas) de JComponent:
  - JButton
  - JLabel
  - JMenu
  - JMenuItem
  - JTextField
  - JTable
  - JSlider - Simula um slider
  - JProgressBar – Exibe o progresso de uma operação.
- Comportamento comum de um JComponent
  - Definir dimensões
  - Modificar cor
  - Definir fontes
  - Atrelar ajuda de contexto (tool tip)

Pacotes Swing

javax.* – extensões padrão (substitui ***com.sun.java.****)
- ***swing.**** – componentes, modelos e interfaces
  - ***border*** – estilos de bordas
  - ***colorchooser*** – palheta de cores
  - ***event*** – eventos e *listeners* específicos do Swing
  - ***filechooser*** – seletor de arquivos
  - ***plaf.**** – pacotes de *look & feel* plugável
  - ***table*** – componente tabela
  - ***text.**** – *framework* de documentos
    - ***html.**** – suporte para HTML 3.2
    - ***rtf*** – suporte para Rich Text Format
  - ***tree*** – componente árvore
  - ***undo*** – suporte para desfazer operações

Container em javax.swing

- Swing possui vários componentes que são contêineres de primeiro nível (top-level container)
  - raiz de uma *"containment hierarchy"*
- Esse contêineres sao usados como o "arcabouço" das GUIs: JApplet, JDialog, JFrame, and JWindow
  - aplicações tipicamente possuem pelo menos uma hierarquia com um JFrame como raiz (janela principal)
  - applets Swing contém uma hierarquia com JApplet como raiz

Container em javax.swing

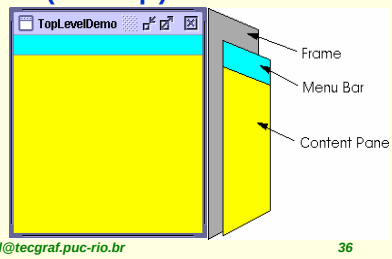
- Contêineres de primeiro nível possuem um contêiner separado chamado de ”content pane” que contém os elementos visíveis. Os demais componentes são adicionados no ”content pane”.
- Para adicionar um componente a um JApplet, JDialog, JFrame, ou JWindow, você deve primeiro invocar o método getContentPane e adicionar o componente ao objeto retornado.

Classe **JFrame**

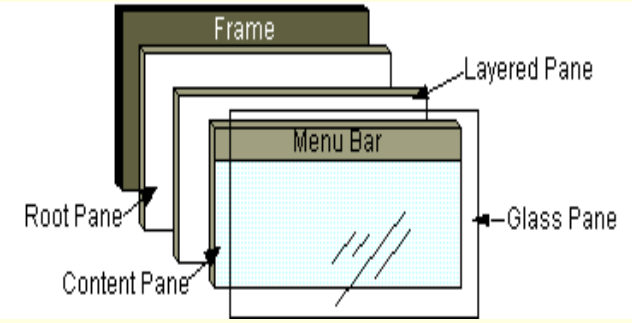
- Modela uma janela do sistema nativo
- Equivalente, no Swing, à classe **Frame** do AWT
- Métodos de JFrame

```
JRootPane getRootPane()
Container getContentPane()
void setJMenuBar(JMenuBar menubar)
JMenuBar getJMenuBar()
void setDefaultCloseOperation(int op)
```

  - **EXIT_ON_CLOSE**
  - **HIDE_ON_CLOSE**
  - **DISPOSE_ON_CLOSE**
  - **DO_NOTHING_ON_CLOSE**



Estrutura de um **JFrame**



© The Java ™ Tutorial

Camadas do **JFrame**

- RootPane
  - gerencia as demais camadas
  - botão “default”
- LayeredPane
  - Contém a *menu bar* e o ContentPane
  - Pode conter subcamadas (*Z order*)
- ContentPane
  - contém os componentes visíveis
- GlassPane
  - invisível por *default*
  - interceptação de eventos/pintura sobre uma região

Exemplo de **JFrame**

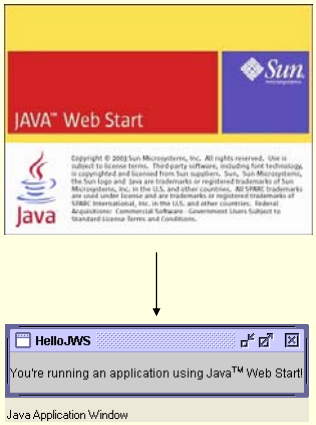
```
import javax.swing.*;
public class HelloWorldSwing {
  /**
   * Create the GUI and show it.  For thread safety, this
   * should be invoked from the event-dispatching thread.
   */
  private static void createAndShowGUI() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);
    //Create and set up the window.
    JFrame frame = new JFrame("HelloWorldSwing");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    //Add the ubiquitous "Hello World" label
    JLabel label = new JLabel("Hello World");
```


Exemplo de **JFrame**

```
    frame.getContentPane().add(label);
    //Display the window.
    frame.pack(); frame.setVisible(true);
  }
  public static void main(String[] args) {
    //Schedule a job for the event-dispatching thread
    //creating and showing this application's GUI.
    SwingUtilities.invokeLater(new Runnable(){
      public void run() {
            createAndShowGUI();
        }
    });
  }
}
```

Java Web Start

- E uma tecnologia que simplifica a distribuição de aplicações. Clicando em um link de uma pagina Web a aplicação será baixada para a maquina do cliente e o JavaWebStart se encarregara de fazer o download de todos os arquivos necessários.
- Ele tambem se encarregara de cuidar do update da aplicacao guardando-a em cache.
- Java Web Start Docs - HelloJWS



Classe JPanel

- Modela um *container* sem decoração, normalmente utilizado para estruturar a interface. Equivalente, no Swing, à classe **Panel** do AWT
- Métodos de JPanel

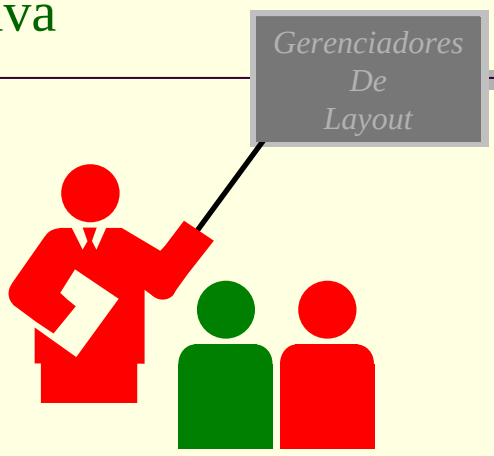
```
JPanel()
JPanel(LayoutManager mgr)
void setLayout(LayoutManager mgr)
Component add(Component comp)
void add(Component c, Object constraints)
```

Exemplo de JPanel

```
JFrame f = new JFrame("Teste");
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JButton b1 = new JButton("Botão 1");
JButton b2 = new JButton("Botão 2");
JPanel p = new JPanel();
p.add(b1);
p.add(b2);
f.getContentPane().add(p);
f.pack();
f.setVisible(true);
```

Exercícios – Questão 23

POO-Java



Gerenciadores de Layout

- Ambientes como o Visual Basic e Delphi usam coordenadas (x,y) para definir a localização de componentes na interface gráfica.
- Swing usa **Gerenciadores de Layout** (Layout Managers) para controlar os componentes serão posicionados.
- Sem um gerenciador de layout, os componentes podem ser movidos para posições inesperadas quando a tela é redimensionada.
- Existem diferentes estilos de arrumação
  - como fluxo de texto
  - orientada pelas bordas
  - em forma de grade, e outros...

Gerenciadores de Layout (cont.)

- **BorderLayout**
  - Divide o contêiner em 5 seções: North, South, East, West, Center
- **BoxLayout**
  - Coloca os componentes em uma única linha ou coluna
- **FlowLayout**
  - Componentes posicionados da esquerda para a direita e de cima para baixo
  - "Flui" para uma nova linha quando necessário.
- **GridLayout**
  - Posiciona componentes em uma grade de linhas e colunas
  - Força os componentes a terem o mesmo tamanho
- **NullLayout**
  - O programador é responsável pelo posicionamento de cada componente

GridLayout

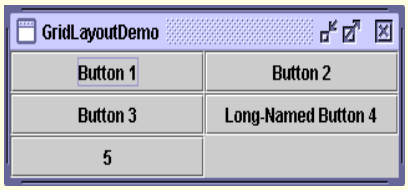
- Posiciona componentes em uma grade de linhas e colunas, com todas as células são de tamanho igual.
- Principais Construtores
  - p.setLayout(new **GridLayout**()); // One row. Columns expand.
  - p.setLayout(new **GridLayout**(*rows*, *cols*));
  - p.setLayout(new **GridLayout**(*rows*, *cols*, *hgap*, *vgap*));
- Não se pode escolher aleatoriamente em que célula posicionar um componente.
  - Adicione-os na ordem “de cima para baixo” e da “esquerda para a direita”.
  - Conseqüentemente, não é possível deixar uma célula “indefinida”.

Exemplo: GridLayout

- Exemplo

```
panel.setLayout(new GridLayout(0,2));
panel.add(new JButton("Button 1"));
panel.add(new JButton("Button 2"));
panel.add(new JButton("Button 3"));
panel.add(new JButton("Long-Named Button 4"));
panel.add(new JButton("5"));
```

GridLayoutDemo

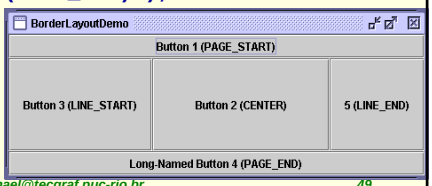


BorderLayout

- Composto de 5 áreas conforme abaixo, sendo que a área central toma o maior espaço possível enquanto as demais áreas se expandem o necessário para preencher os espaços restantes.

```
...//Container panel = aFrame.getContentPane()...
panel.setLayout(new BorderLayout());
JButton b = new JButton("Button 1 (PAGE_START)");
panel.add(b, BorderLayout.PAGE_START);
b = new JButton("Button 2 (CENTER)");
b.setPreferredSize(new Dimension(200, 100));
panel.add(b, BorderLayout.CENTER);
b = new JButton("Button 3 (LINE_START)");
panel.add(b, BorderLayout.LINE_START);
b = new JButton("Long-Named Button 4(PAGE_END)");
panel.add(b,BorderLayout.PAGE_END);
b = new JButton("5 (LINE_END)");
panel.add(b,BorderLayout.LINE_END);
```

BorderLayoutDemo

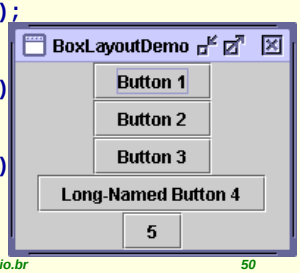


BoxLayout

- Empilha seus componentes em linha ou em coluna de acordo com opção selecionado.

```
...//Container panel = aFrame.getContentPane()...
panel.setLayout(new BoxLayout(pane, BoxLayout.Y_AXIS));
JButton b = new JButton("Button 1");
b.setAlignmentX(Component.CENTER_ALIGNMENT); panel.add(b);
b = new JButton("Button 2");
b.setAlignmentX(Component.CENTER_ALIGNMENT); panel.add(b);
b = new JButton("Button 3");
b.setAlignmentX(Component.CENTER_ALIGNMENT);
panel.add(b);
b = new JButton("Long-Named Button 4");
b.setAlignmentX(Component.CENTER_ALIGNMENT)
panel.add(b);
b = new JButton("5");
b.setAlignmentX(Component.CENTER_ALIGNMENT)
panel.add(b);
```

BoxLayoutDemo



FlowLayout

- Empilha seus componentes em linha ou em coluna de acordo com opção selecionado.

```
...//Container panel = aFrame.getContentPane()...
panel.setLayout(new FlowLayout());
JButton b = new JButton("Button 1"); panel.add(b);
b = new JButton("Button 2"); panel.add(b);
b = new JButton("Button 3"); panel.add(b);
b = new JButton("Long-Named Button 4");
   panel.add(b);
b = new JButton("5"); panel.add(b);
```

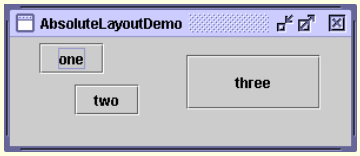
FlowLayoutDemo

NullLayout

- Posiciona seus componentes de acordo com a posicao absoluta

```
...//Container panel = aFrame.getContentPane()...
panel.setLayout(null);     JButton b1,b2,b3;   Dimension size;
b1 = new JButton("one"); panel.add(b1);
b2 = new JButton("two"); panel.add(b2);
b3 = new JButton("three"); panel.add(b3);
Insets insets = panel.getInsets();
size = b1.getPreferredSize();
b1.setBounds(25+insets.left, 5+insets.top, size.width, size.height);
size = b2.getPreferredSize();
b2.setBounds(55+insets.left, 40+insets.top, size.width, size.height);
size = b3.getPreferredSize();
b3.setBounds(150+insets.left, 15+insets.top, size.width+50, size.height+20);

...//In the main method:
Insets insets = frame.getInsets();
frame.setSize(300+insets.left+insets.right,
              125+insets.top+insets.bottom);
```

NullLayoutDemo

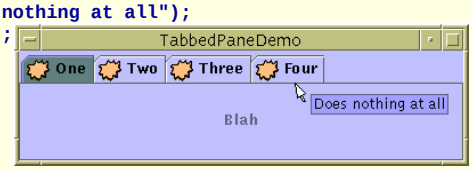


JTabbedPane

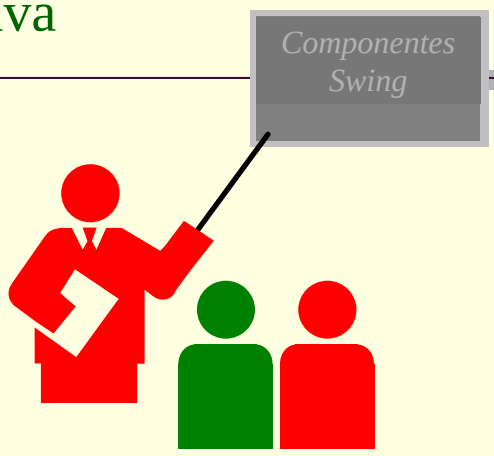
- JTabbedPane permiti que diversos componentes, em geral JPanel, compartilhem um mesmo espaço.

```
JTabbedPane tP = new JTabbedPane(); JComponent p1, p2, p3, p4;
ImageIcon icon = createImageIcon("images/middle.gif");
p1 = makeTextPanel("Panel #1");
tP.addTab(“One", icon, p1, "Does nothing");
tP.setMnemonicAt(0, KeyEvent.VK_1);
p2 = makeTextPanel("Panel #2");
tP.addTab("Two", icon, p2, "Does twice as much nothing");
tP.setMnemonicAt(1, KeyEvent.VK_2);
p3 = makeTextPanel("Panel #3");
tP.addTab("Three", icon, p3,“nothing"); tP.setMnemonicAt(2, KeyEvent.VK_3);
p4 = makeTextPanel( "Panel #4 (has a preferred size of 410 x 50).");
p4.setPreferredSize(new Dimension(410, 50));
tP.addTab(“Four", icon, p4, "Does nothing at all");
tP.setMnemonicAt(3, KeyEvent.VK_4);
```

TabbedPaneDemo



POO-Java



Classe JLabel

- Essa classe modela um texto e/ou imagem não editável, isto é, sem interação com o usuário. É o equivalente, no Swing, ao **Label** do AWT, só que com mais recursos
- Pode-se controlar tanto o alinhamento horizontal como o vertical, e o **JLabel** pode passar o foco para outro elemento
- Pode também manipular conteúdo HTML
  - Se o texto possuir "<html>...</html>", o conteúdo é apresentado como HTML.
  - JLabel fontes são ignoradas se HTML é usado. Nesse caso, todo o de fontes deve ser realizado através de tags HTML.
  - Deve ser usado <P>, e não
, para forçar uma quebra de linha. O suporte HTML ainda é incipiente.

Métodos de JLabel

```
void setText(String text)
void setIcon(Icon icon)
void setIconTextGap(int gap)
void setHorizontalAlignment(int a)
void setVerticalAlignment(int a)
void setLabelFor(Component c)
void setDisplayedMnemonic(int key)
void setDisplayedMnemonic(char aChar)
```

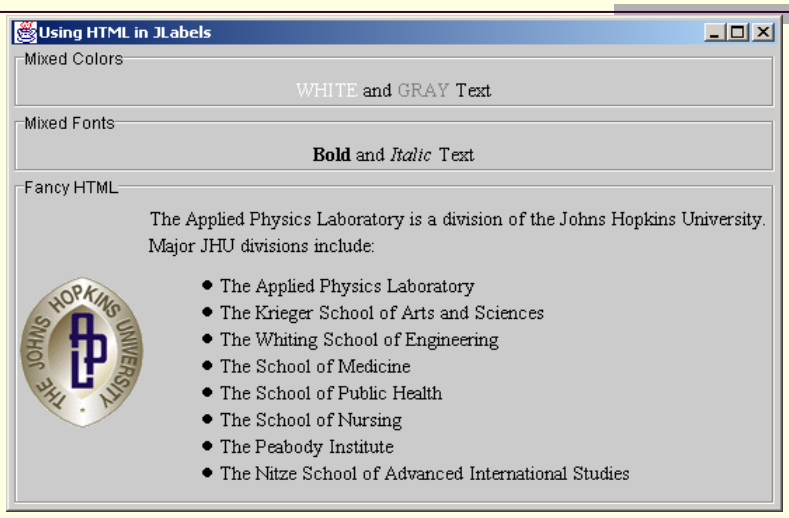
Exemplo de **JLabel**

```
JFrame f = new JFrame("Teste");
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JLabel l = new JLabel("Isso é um JLabel");
l.setIcon(new ImageIcon("javalogo.gif"));
Container cp = f.getContentPane(); cp.add(l);
f.pack();
f.show();
```

JLabel

```
String labelText =
  "<html><FONT COLOR=WHITE>WHITE</FONT> and " +
  "<FONT COLOR=GRAY>GRAY</FONT> Text</html>";
JLabel coloredLabel =
  new JLabel(labelText, JLabel.CENTER);
...
labelText =
  "<html><B>Bold</B> and <I>Italic</I> Text</html>";
JLabel boldLabel =
  new JLabel(labelText, JLabel.CENTER);
labelText =
  "<html>The Applied Physics Laboratory is..." +
  "of the Johns Hopkins University." +
  "<P>" + ... "...</html>";
```


JLabel



Classe JButton

- Modela um *push-button*
  - Sendo uma sub-classe de **AbstractButton**, herda os métodos **getLabel** e **setLabel** que permitem consultar e alterar seu texto. Permite que o botão seja cadastrado como *default button* do **RootPane**
- Métodos de JButton

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

void setLabel(String label)
String getLabel()
```


Métodos de **JButton**

- Instanciação (exemplos)
  - JButton btnOk = new javax.swing.JButton();
  - JButton proximo = new JButton(new ImageIcon("right.gif"));
  - JButton proximo = new JButton("Next", rightArrow);
  - JButton button = new JButton("I'm a button!");
  - button.setMnemonic(KeyEvent.VK_I); // tecla “I” com sendo a tecla de atalho
- Principais métodos
  - void setText(String)
  - void setToolTipText(String)
  - void addActionListener(ActionListener)
  - void setBounds(int x, int y, int width, int height)
  - void setVisible(boolean)
  - void setEnabled(boolean)

Exemplo de **JButton**

```
JFrame f = new JFrame("Teste");
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JButton b1 = new JButton("Botão 1");
JButton b2 = new JButton("Botão 2");
Container cp = f.getContentPane();
cp.setLayout(new GridLayout(1,0));
cp.add(b1);
cp.add(b2);
f.pack();
f.show();
```


Classe **JRadioButton**

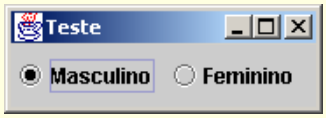
- Modela um botão de escolha que pode ser marcado e desmarcado
- Objetos do tipo **JRadioButton** são organizados em grupos
- Apenas um único botão de um grupo pode estar marcado em um dado momento

- Métodos de JRadioButton

```
JRadioButton(String label)
JRadioButton(String label, boolean state)

boolean isSelected()
void setSelected(boolean state)
```

Exemplo de **JRadioButton**

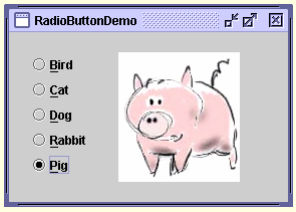
```
JFrame f = new JFrame("Teste");
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JRadioButton bm = new
  JRadioButton("Masculino“,true);
JRadioButton bf = new JRadioButton("Feminino");
ButtonGroup bg = new ButtonGroup();
bg.add(bm);
bg.add(bf);
Container cp = f.getContentPane();
cp.setLayout(new FlowLayout());
cp.add(bm);
cp.add(bf);
f.pack();
f.setVisible(true);
```



Classe ButtonGroup

- Cria um “escopo” de exclusão para um grupo de botões
- Basta criar um **ButtonGroup** e adicionar a ele os **JRadioButton**s que compõem o grupo
- Métodos de ButtonGroup

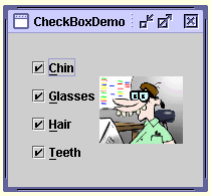
```
void add(AbstractButton b)
ButtonModel getSelection()
boolean isSelected(ButtonModel
void setSelected(ButtonModel m,
                 boolean state)
```



RadioButtonDemo

Classe **JCheckBox**

- Modela um botão de escolha que pode ser marcado e desmarcado
- Métodos JCheckBox

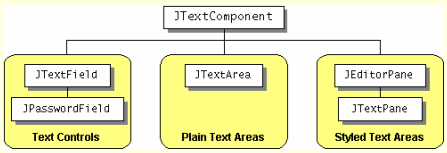



```
public JCheckBox(String label)
public JCheckBox(String label,
                 boolean state)

public boolean isSelected()
public void setSelected(boolean state)
```

CheckBoxDemo

Classe **JTextComponent**

- Classe abstrata que modela o que há de comum entre diferentes elementos de edição de texto




```
public void setText(String t)
public String getText()
public void setEditable(boolean b)
```

Classe **JTextField**

- Cria campo de edição de texto de uma linha

```
JTextField()
JTextField(String text)
JTextField(int columns)
JTextField(String text, int columns)
void setColumns(int columns)
```

- Principais métodos

```
setBounds(int x, int y, int width, int height)
setEnabled(boolean)
setText(String)
String getText()
requestFocus()
```


Classe **JPasswordField**

- Estende **JTextField**
- Caracteres digitados não são exibidos

```
JPasswordField()
JPasswordField(int columns)
JPasswordField(String text, int columns)
char[] getPassword()
void setEchoChar(char c)
```

JFormattedTextField - Máscara de Entrada

- Estende **JTextField** (a partir de 1.4)
- Permite ao desenvolvedor especificar o conjunto de caracteres aceitos como entrada do campo.
- A classe **MaskFormatter** é usada para formatar e editar Strings. O comportamento da classe MaskFormatter é controlado por um tipo de máscara de String que especifica os caracteres válidos que podem ser digitados naquele campo.

JFormattedTextField

- Cep

```
MaskFormatter mask = null;
try {
  mask = new MaskFormatter("#####-###");
  mask.setPlaceholderCharacter('_');
} catch (java.text.ParseException exc) {
  ......
}
JFormattedTextField cep =
    new JFormattedTextField(mask);
```

- Telefone

```
mask = new MaskFormatter("(##)####-####");
....
```

Caractere	Descrição
#	Qualquer **número** válido, usa Character.isDigit.
'	Caractere de escape, usado para o escape de qualquer caractere de formato especial.
U	Qualquer caractere(Character.isLetter). Todas as letras minúsculas são transformadas em maiúsculas.
L	Qualquer caractere(Character.isLetter). Todas as letras são transformadas para minúsculas.
A	Qualquer caractere ou número (Character.isLetter ou Character.isDigit).
?	Qualquer caractere.
*	Qualquer coisa.
H	Qualquer caractere hexadecimal (**0**-9, **a**-f ou **A**-F).

Classe **JTextArea**

- Cria um campo de edição de texto com múltiplas linhas

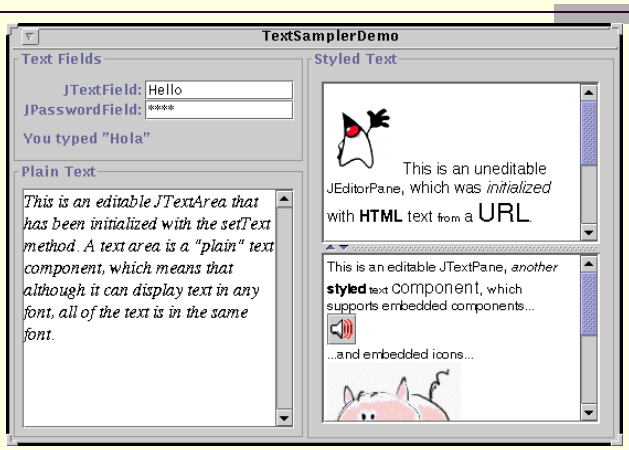
```
JTextArea(int rows, int columns)
JTextArea(String text, int rows, int columns)
void append(String t)
void insert(String t, int pos)
void setLineWrap(boolean wrap)
void setWrapStyleWord(boolean word)
```


Classe **JTextPane**

- Componente de texto que suporta atributos representados graficamente (*styled text*)
- Permite o uso de diversas “fontes” no texto
- Permite a inclusão de imagens e de outros componentes

Elementos de Edição de Texto

JTextArea



JEditorPane

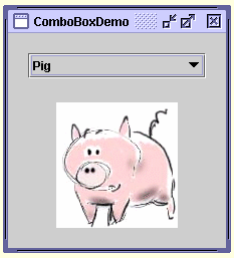
JTextPane

TextSamplerDemo

© The Java ™ Tutorial

JComboBox

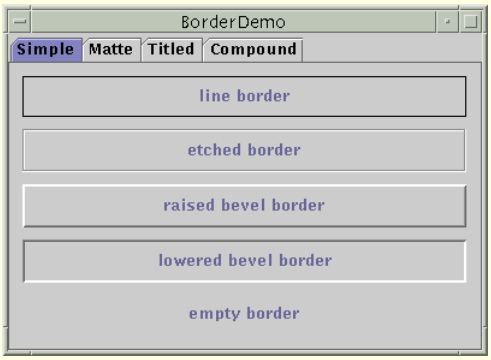
- Um componente que combina um botão um campo (editável ou não) e uma lista “drop-down”.
- Principais métodos
  - void setBounds(int x, int y, int width, int height)
  - void requestFocus()
  - void setEnabled(boolean)
  - int getSelectedIndex()
  - void setSelectedIndex(int)
  - void removeAllItems()
  - void addItem(String)
  - String getSelectedItem()
  - void getSelectedItem(String)
  - void addActionListener (ActionListener)



Bordas

- O Swing permite a criação de **bordas** (molduras) envolvendo componentes
- O método **`setBorder`** de **JComponent** permite "emoldurar" um componente com uma borda

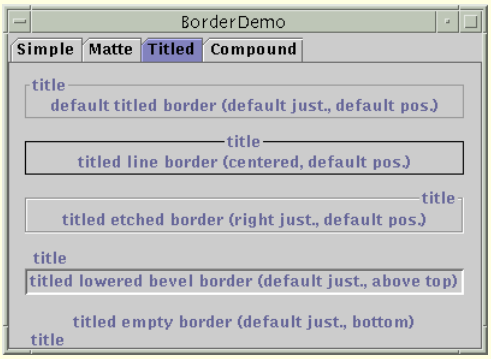
Exemplos de Bordas



© The Java ™ Tutorial

`BorderDemo`

Mais exemplos...



© The Java ™ Tutorial

BorderFactory

- As bordas do Swing foram projetadas para serem compartilhadas
- Ao invés de criar explicitamente uma instância de uma das classes de bordas, deve-se usar um dos métodos de BorderFactory para obter uma instância compartilhada de um tipo de borda

- Métodos:

```
static Border createLineBorder(Color color)
static Border createEtchedBorder()
static TitledBorder createTitledBorder(String t)
static TitledBorder createTitledBorder(Border b,
                                       String t)
```

Painéis, Bordas e Botões

`JCheckBox`

`JRadioButton`

Classe **JList**

- Modela um elemento que apresenta uma lista de opções ao usuário
- Permite a seleção simples (um único elemento), ou múltipla (vários elementos)
- Métodos de JList

```
JList(Object[] listData)
void setSelectionMode(int selectionMode)
int getSelectedIndex()
int[] getSelectedIndices()
Object getSelectedValue()
Object[] getSelectedValues()
```

JList - Modos de Seleção

- Exemplo JList

```
JFrame f = new JFrame("Teste");
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
String[] nomes = {"a", "b", "c", "d", "e", "f"};
JList list = new JList(nomes);
f.getContentPane().add(list); //f.add(list) em Java1.5!
f.pack(); f.setVisible(true);
```

- Modos de Seleção
  - SINGLE_SELECTION
    - Apenas um elemento pode ser selecionado
  - SINGLE_INTERVAL_SELECTION
    - Um intervalo contíguo de elementos pode ser selecionado
  - MULTIPLE_INTERVAL_SELECTION
    - Não há restrições sobre a seleção (“default”)

Eventos de Seleção

- Eventos de seleção são gerados sempre que a seleção de uma lista é alterada.
- Esses eventos podem ser tratados através da adição de um ListSelectionListener.
- A interface ListSelectionListener pertence ao pacote javax.swing.event e define apenas um método: `valueChanged`

Exemplo de **ListSelectionListener**

```
class MeuListener implements ListSelectionListener {
  public void valueChanged (ListSelectionEvent e) {
    if (e.getValueIsAdjusting()) return;
    JList lista = (JList)e.getSource();
    if (lista.isSelectionEmpty()) {
      …
    }
    else {
      int index = lista.getSelectedIndex();
      String val = (String)lista.getSelectedValue();
      …
    }
  }
}
```

Exercícios – Questão 24

Barras de Rolagem

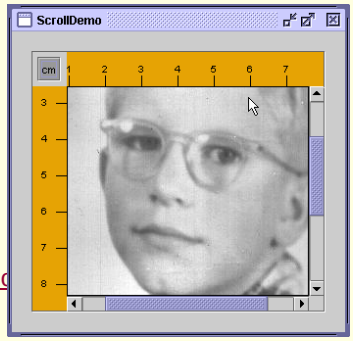
- No Swing, ao invés de repetir o código que implementa o uso de barras de rolagem (*scrollbars*) em diferentes elementos, foi criado um elemento de interface cujo único objetivo é fornecer essa funcionalidade
- A classe JScrollPane modela esse elemento e a interface Scrollable define o que cada elemento deve ter para poder ser tratado por um JScrollPane

Elementos *Scrollable*

- As classes JList, JTextComponent, JTree e JTable implementam a interface Scrollable e, portanto, suas instâncias podem ser usadas com JScrollPane

```
//Em um container que use BorderLayout:
textArea = new JTextArea(5, 30);
...
JScrollPane sP= new JScrollPane(textArea);
...
setPreferredSize(new Dimension(450, 110));
...
add(sP, BorderLayout.CENTER);
```

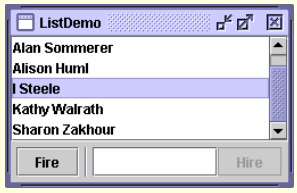
ScrollDemo



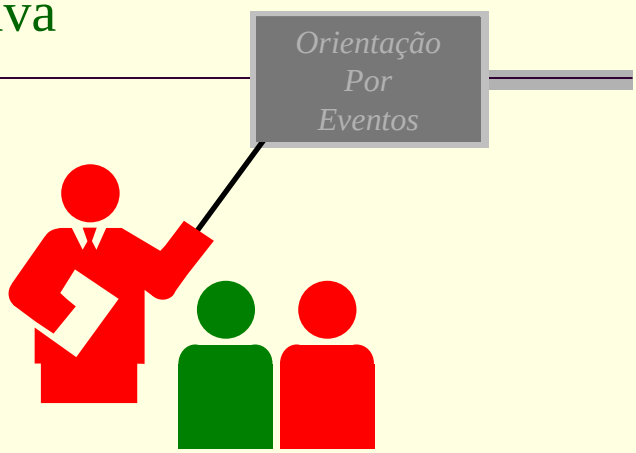
Exemplo de Scrollable **JList**

```
JFrame f = new JFrame("Teste");
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
String[] nomes = {“Alan Sommerer", “Alison Huml",
                  “I Steele", “Kathy Walrath",
                  “Sharon Zakhour"};
JList list = new JList(nomes);
list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
list.setLayoutOrientation(JList.HORIZONTAL_WRAP);
list.setVisibleRowCount(-1);
...
JScrollPane listScroller = new JScrollPane(list);
listScroller.setPreferredSize(new Dimension(250, 80));
f.getContentPane().add(listScroller);
f.pack();
f.setVisible(true);
```

ListDemo



POO-Java



Orientação por Eventos

- Motivação
  - Um modelo de programação que tornou-se bastante difundido com o uso de interfaces gráficas foi a *programação orientada por eventos*.
  - Segundo esse modelo, o programa deixa de ter o controle do fluxo de execução, que passa a um sistema encarregado de gerenciar a interface.
  - Assim, o programa passa a ser chamado pelo sistema quando algum *evento* é gerado na interface.

Orientação por Eventos

Mecanismo de Callback

- Para que o programa possa ser chamado pelo sistema, ele deve registrar uma função para cada evento de interface que ele desejar tratar. Tais funções são chamadas de *callbacks* por serem 'chamadas de volta' pelo sistema.
- Esse modelo é ortogonal ao modelo de orientação por objetos. É perfeitamente possível projetar um sistema OO que use o modelo de orientação por eventos para tratar eventos de interface, comunicações, etc.
- Porém, temos um problema: uma linguagem puramente OO não possui o conceito de *função*. Como resolver então?

Orientação por Eventos

Solução – Objeto Callback

- A solução é utilizar um *objeto* que faça o papel de callback. Ou seja, onde registraríamos uma função, passamos a registrar um objeto. Assim, quando o sistema precisar executar a callback, ele deverá executar um determinado método do objeto. Esse método, então, fará o tratamento do evento.

Análise de Caso – java.awt

- O pacote **java.awt** é um pacote padrão de Java que implementa um sistema de interface. Esse sistema possui janelas, textos, botões e diversos outros elementos.

Pacote java.awt

- Como a maioria dos sistemas de interface, o AWT é um sistema orientado por eventos, ou seja, o tratamento dos eventos deve ser feito por meio de callbacks.

Callbacks em Java

- Como Java é uma linguagem puramente OO, callbacks devem ser implementadas através de objetos.
- Um objeto que implementa uma callback é chamado de *listener*.

Pacote java.awt

Listeners & Eventos

- Os listeners, como dito anteriormente, fazem o papel das callbacks. Listeners são definidos por interfaces e podem estar aptos a tratar mais de um tipo de evento. Quando um listener tem um de seus métodos chamados, ele recebe um parâmetro descrevendo o evento ocorrido. Esse parâmetro é um objeto: existem classes para modelar diferentes grupos de eventos.

Pacote java.awt.event

- As interfaces que definem os listeners e as classes que definem os eventos pertencem ao pacote java.awt.event. Esse pacote, por sua vez, pertence ao pacote java.awt.

Listeners

- Definem interfaces que representam um grupo de *callbacks*
- São registrados junto aos componentes
- Exemplo: java.awt.event.MouseListener

```
public abstract void mouseClicked(MouseEvent e)
public abstract void mousePressed(MouseEvent e)
public abstract void mouseReleased(MouseEvent e)
public abstract void mouseEntered(MouseEvent e)
public abstract void mouseExited(MouseEvent e)
```


Registro de Listeners

- Mecanismo de *callbacks*
- Implementação da interface
- Uso de *local inner classes*

```
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    System.out.println("Botão pressionado");
  }
});
```

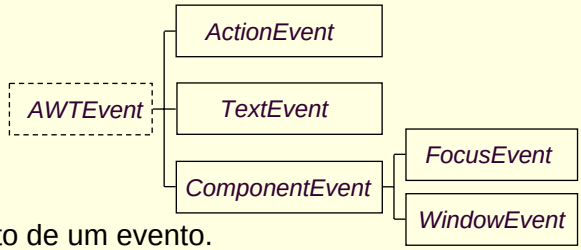
Eventos

- Trazem informações sobre o evento ocorrido
- São passados aos *listeners* (*callbacks*)
- Exemplo: java.awt.event.MouseEvent

```
public int getX()
public int getY()
public int getClickCount()
```


Pacote java.awt.event

- Hierarquia de Classes Eventos de java.awt.event



- AWTEvent
  - Modelo abstrato de um evento.
  - Cada classe que estende AWTEvent modela um *grupo* de eventos, daí a necessidade de se colocar um identificador que indique qual evento do grupo efetivamente ocorreu.

```
public int getID()
```

Classes de Eventos

- AncestorEvent
- CaretEvent
- ChangeEvent
- DocumentEvent
- HyperlinkEvent
- InternalFrameEvent
- ListDataEvent
- ListSelectionEvent
- MenuDragMouseEvent
- MenuEvent
- MenuKeyEvent
- PopupMenuEvent
- TableColumnModelEvent
- TableModelEvent
- TreeExpansionEvent
- TreeModelEvent
- TreeSelectionEvent
- UndoableEditEvent

Pacote java.awt.event

Ação que dispara o evento	Tipo de listener
Usuário clica em um botão, pressiona return em uma caixa de texto, ou seleciona um item de menu.	ActionListener
Usuário fecha um frame (janela principal aplicação)	WindowListener
Usuário pressione o botão do mouse enquanto o cursor está sobre um componente.	MouseListener
Usuário move o mouse sobre um componente.	MouseMotionListener
Usuário move o wheel sobre um componente	MouseWheelListener
Componente se torna visível	ComponentListener
Componente obtém o foco do teclado	FocusListener
Usuário pressiona alguma tecla	KeyListener
Item selecionado muda em uma tabela ou lista	ListSelectionListener

<JavaTutorial>/uiswing/events/eventsandcomponents.html

Pacote java.awt.event (cont.)

- Cada evento é representado por um objeto que fornece informação acerca de sua fonte.
- Cada componente pode ter vários listeners registrados. Do mesmo modo, um único objeto pode se registrar como listener de diversos componentes.
- Todos os eventos para cada componente em particular podem ser consultado em:

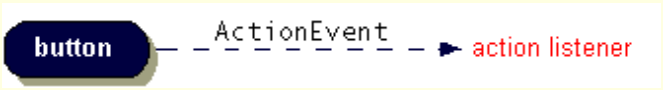
  <JavaTutorial>/uiswing/events/api.html

Manipulando eventos

- 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.

Manipulando eventos (cont.)

- De uma forma geral:
  - Uma classe X que estiver interessada em processar eventos de ação deve implementar a interface ActionListener. Um objeto de X é registrado com um listener usando o método addActionListener desse componente.
  - Quando a ação ocorre, o método actionPerformed é então invocado.



Manipulando eventos (cont.)

- Exemplo

```
button.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent e) {
    numClicks++;
    label.setText(labelPrefix + numClicks);
  }
});
```

- No exemplo acima, o que está em vermelho é chamado de classe interna anônima (classe aninhada).

Classes internas

- Com uso de classe interna

```
public class ListenerSaida extends WindowAdapter {
  public void windowClosing(WindowEvent event) {
    System.exit(0);
  }
}
…
frame.addWindowListener(new ListenerSaida());
```

- Sem uso de classe interna

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


Pacote java.awt.event

- WindowEvent
  - Modela os eventos que podem ocorrer em uma janela. Essa classe declara constantes que identificam os diversos eventos.

```
public static final int WINDOW_OPENED
public static final int WINDOW_CLOSING
public static final int WINDOW_CLOSED
public static final int WINDOW_ICONIFIED
public static final int WINDOW_DEICONIFIED
public static final int WINDOW_ACTIVATED
public static final int WINDOW_DEACTIVATED

public Window getWindow()
```

Pacote java.awt.event

- WindowListener
  - Modela a callback de um evento do tipo **WindowEvent**. Essa interface declara um método para cada evento do grupo.

```
public abstract void windowOpened(WindowEvent e)
public abstract void windowClosing(WindowEvent e)
public abstract void windowClosed(WindowEvent e)
public abstract void windowIconified(WindowEvent e)
public abstract void windowDeiconified(WindowEvent e)
public abstract void windowActivated(WindowEvent e)
public abstract void windowDeactivated(WindowEvent e)
```


Pacote java.awt.event

- Implementando um Listener
  - Para criarmos um listener para um evento de janela devemos criar uma classe que implemente a interface **WindowListener**. Nessa classe, codificaremos, então, o método correspondente ao evento que desejamos tratar. Porém, não podemos implementar uma interface e deixar de codificar algum método. Assim, precisaremos implementar todos os sete métodos definidos !!!
    - Será que não existe alguma alternativa mais inteligente ?

Pacote java.awt.event

- Adapters
  - No caso anterior, seis implementações seriam vazias pois só desejávamos responder a um único evento. Como essa é uma situação comum, o pacote **event** define *adaptadores* para todas as interfaces de listeners que têm mais de um método. Um adaptador é uma classe que implementa o listener e dá implementações vazias para todos os métodos.
  - No nosso exemplo anterior, o botão para fechar a janela não funciona. Isso acontece porque não criamos um listener que fosse chamado para tratar esse evento e, efetivamente, fechasse a janela. Agora podemos criar esse listener

Exemplo 1: WindowAdapter

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

public class Mundo {
  public static void main(String[] args) {
    Frame janela = new Frame("Mundo");
    janela.add(new Label("Olá Mundo!"));
    janela.addWindowListener(new FechaMundo(janela));
    janela.pack();janela.setVisible(true);
  }
}
class FechaMundo extends WindowAdapter {
  Frame janela;
  FechaMundo(Frame janela) { this.janela = janela;  }
  public void windowClosing(WindowEvent e) {
    janela.dispose(); System.exit(0);
  }
}
```

Exemplo 2: WindowAdapter

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

public class Mundo {
  Frame janela;
  public void roda() {
    class FechaMundo extends WindowAdapter {
      public void windowClosing(WindowEvent e) {
        janela.dispose();  System.exit(0);
      }
    }
    janela = new Frame("Mundo");
    janela.add(new Label("Olá Mundo!"));
    janela.addWindowListener(new FechaMundo());
    janela.pack(); janela.setVisible(true);
  }
  public static void main(String[] args) {
    new Mundo().roda();
  }
}
```


Aplicação de Classes Aninhadas

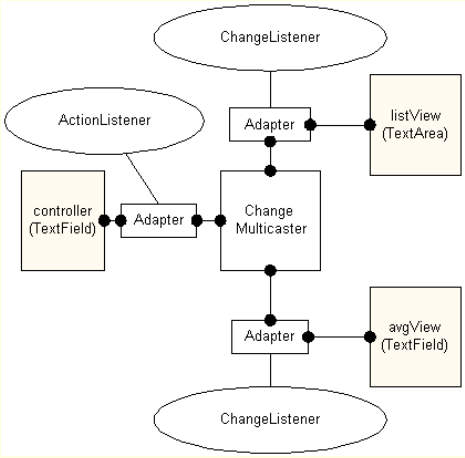
- Classes Aninhadas
  - Note que, como acontece no exemplo, o uso de uma classe que implementa um listener geralmente está restrito a um único local.
  - Para não criarmos diversas classes (uma para cada callback) cuja utilização é bem pontual, a linguagem Java nos permite criar e instanciar uma classe simultaneamente, no meio de um trecho de código.
  - Usando uma classe aninhada para criar o listener no momento em que ele é necessário, podemos simplificar bastante o código do exemplo.

Exemplo 3: WindowAdapter

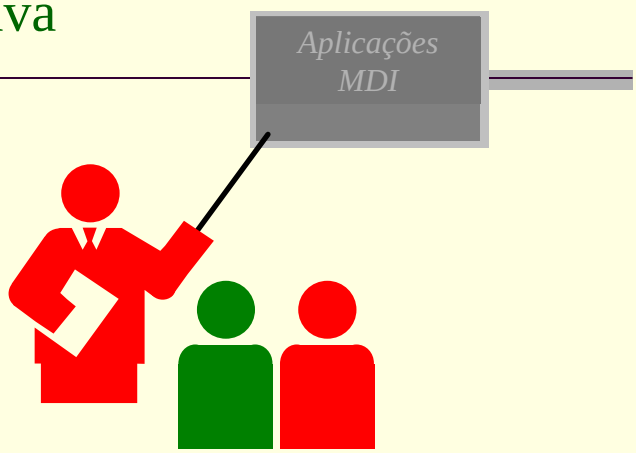
```
import java.awt.*;
import java.awt.event.*;
public class Mundo {
  public static void main(String[] args) {
    final Frame janela = new Frame("Mundo");
    janela.add(new Label("Olá Mundo!"));
    janela.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        janela.dispose(); System.exit(0);
      }
    });
    janela.pack(); janela.setVisible(true);
  }
}
}
```

Exercícios – Questão 25

Delegação de Eventos



POO-Java



Aplicações MDI

- MDI: Multiple Document Interface
  - Programa tem uma janela de “desktop” que contém as demais janelas.
  - As janelas filhas podem ser minimizadas e movidas dentro do desktop, mas não removidas.
  - Se o desktop for minimizado, todas as janelas filhas também o são.
- Suporte para MDI no Swing
  - JDesktopPane
    - Serve como a janela mãe (desktop) das demais janelas.
  - JInternalFrame
    - Semelhante a um frame, exceto que ele está restringido a permanecer dentro do objeto JDesktopPane.

JInternalFrame

- É um contêiner para ser utilizado em aplicações MDI.
- Construtor principal
  - public JInternalFrame(String title,
    boolean resizable,
    boolean closeable,
    boolean maximizable,
    boolean iconifiable)
- Outros contrutores úteis
  - JInternalFrame()
  - JInternalFrame(String *title*)
  - JInternalFrame(String *title,* boolean *resizable*)
  - JInternalFrame(String *title*, boolean *resizable*, boolean *closable*)

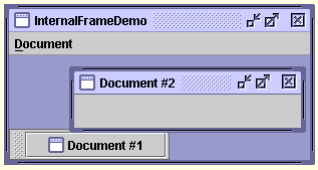
JInternalFrame

- Outros métodos:
  - moveToFront()
  - moveToBack()
  - setLocation (int x, int y)
  - setSize(int largura, int comprimento);
  - setMinimumSize(new Dimension(largura, comprimento));
  - setTitle(String);
  - setMaximizable(boolean);
  - setIconifiable(boolean);
  - setClosable(boolean);
  - setResizable(boolean);
  - setJMenuBar(JMenuBar);
  - pack()

JDesktopPane

- Um objeto JDesktopPane é um contêiner para construir uma aplicação de múltiplos documento (MDI). Serve como a janela mãe (desktop) das demais janelas.
- Um objeto JDesktopPane pode conter diversos objetos JInternalFrame.
- Principais métodos
  - add(JInternalFrame)
  - cascadeFrames()

InternalFrameDemo

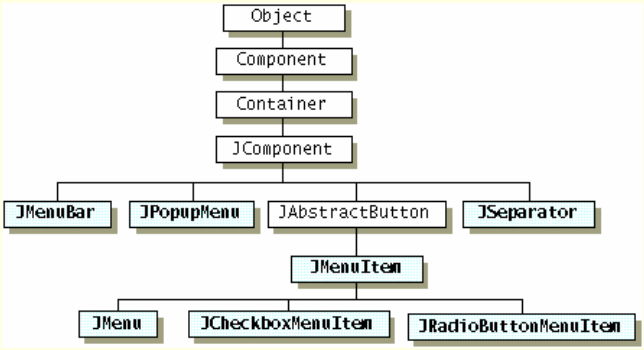


Trabalhando com menus

- A biblioteca Swing disponibiliza menus comuns (*pull-down*) e menus *pop-up*
- Um menu pode conter itens de menu e separadores, sendo que os itens podem ter texto e imagem, além de poderem ser marcados (como um *checkbox*)
- As classes do Swing a serem utilizadas são JMenuBar, JMenu e JMenuItem.
  - JMenuBar: uma barra de menus.
  - JMenu: cada um dos menus suspenso que podem ser "pendurados" em uma barra de menus.
  - JMenuItem: cada opção a ser adicionada a um menu.

Menus

- Principal construtor de JMenu:
  - private JMenu menuCadastros = new JMenu("Cadastros");
- Hierarquia dos elementos de Menu



Classe **JMenuBar**

- Modela uma barra de menu, fixa em uma janela da aplicação
- Na barra de menu pode-se colocar os menus da aplicação, sendo que o menu de ajuda (*help*) possui um tratamento diferenciado
- Métodos de **JMenuBar**

```
JMenu add(JMenu c)
JMenu getMenu(int index)
int getMenuCount()

void setHelpMenu(JMenu menu)
JMenu getHelpMenu()
```

JMenuBar

```
// Criando o objeto JMenuBar
   private JMenuBar menuBar = new JMenuBar();
...
// Adicionando objetos JMenu ao objeto JMenuBar
   menuBar.add(menuVenda);
...
// Adicionando um menu de gerenciamento de janelas à barra de
   menus.
   menuBar.add(new WindowMenu(desktop));

// Adicionando a barra de menus à aplicação MDI (JFrame)
   setJMenuBar(menuBar);
```


Classe **JMenu**

- Modela um menu que pode ser colocado na barra de menus ou dentro de um outro menu. Pode conter:
  - itens de menu
  - Separadores
- Métodos de **JMenu**

```
JMenu(String s)
JMenu(String s, boolean tear-off)

JMenuItem add(String name)
JMenuItem add(JMenuItem menuItem)
JMenuItem insert(JMenuItem i, int p)

void addSeparator()
void insertSeparator(int pos)
```

JMenu

```
// Criando o objeto JMenu
   JMenu menuVenda = new JMenu("Venda");
...
//Adicionando objetos JMenuItem ao objeto JMenu
   menuVenda.add(menuEfetuarVenda);
...
// Adicionando um separador de itens ao objeto JMenu.
   menuVenda.addSeparator();
...
// Adicionando um mnemônico
   menuVenda.setMnemonic('V');
...
```


Classe **JMenuItem**

- Modela um item de menu
- É superclasse de **JMenu**, uma vez que um menu também pode ser um item de menu
- É sub-classe de **JAbstractButton**, logo, um item de menu é um botão
- Métodos de **JMenuItem**

```
JMenuItem(String text)
JMenuItem(String text, Icon icon)

void setMnemonic(char mnemonic)
void setAccelerator(KeyStroke keyStroke)
void setEnabled(boolean b)

void addActionListener(ActionListener l)
```

JMenuItem

```
// Criando o objeto JMenuItem
private JMenuItem menuEfetuarVenda = new JMenuItem("Venda...");
...
// Adicionando um mnemônico
 menuVenda.setMnemonic('V');        ...
 ks =
 KeyStroke.getKeyStroke(KeyEvent.VK_X,ActionEvent.ALT_MASK)
 menuSistemaSair.setAccelerator(ks);
...
// Definindo um listener para o objeto JMenuItem.
 menuEfetuarVenda.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent ae) {
     /*
     ** Código a ser executado quando o
           ** usuário clica no item de menu.
     */
    }
 });
```


JMenuBar, JMenu, JMenuItem

- Típico código para manipulação de menus:

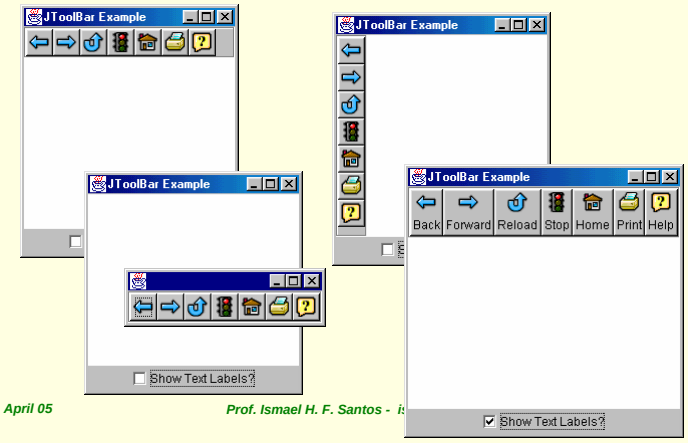
```
JFrame f = new JFrame("Teste");
JMenuBar menuBar = new JMenuBar();
JMenu    menuVenda = new JMenu(“Venda”);
JMenuItem menuEfetuarVenda= new JMenuItem(“Efetuar”);

 menuBar.add(menuVenda);
 menuVenda.add(menuEfetuarVenda);
 menuEfetuarVenda.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
      // Código do item de menu.
   }
});
menuEfetuarVenda.setAccelerator(KeyStroke.getKeyStroke(
                          KeyEvent.VK_E, KeyEvent.CTRL_MASK));
menuEfetuarVenda.setMnemonic(‘E');

f.setJMenuBar(menuBar); f.pack(); f.setVisible(true);
```

Exemplo de Menus

```
JFrame f = new JFrame("Teste");
JMenuBar b = new JMenuBar();
JMenu m = b.add(new JMenu("Arquivo"));
m.setMnemonic('a');
JMenuItem i = m.add("Sair");
i.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ev) {
    System.exit(0);
  }});
i.setAccelerator(KeyStroke.getKeyStroke(
                  KeyEvent.VK_S, KeyEvent.CTRL_MASK));
i.setMnemonic('s');
f.setJMenuBar(b);
f.pack();
f.setVisible(true);
```


JToolBar

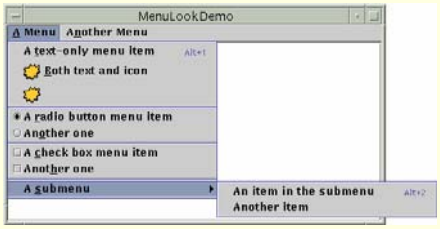
- Similar a uma painel para botões.
- Dockable: pode ser arrastada.



Menus de Escolha

- **JMenuItem** possui duas sub-classes que permitem a escolha de opções:
  - **JCheckBoxMenuItem**
  - **JRadioButtonMenuItem**
- Essas classes modelam itens de menu que funcionam da mesma maneira que os botões de escolha que vimos

MenuLookDemo



Classe **JPopupMenu**

- Modela um menu *pop-up*, isto é, um menu que pode ser aberto sobre um elemento qualquer de interface, fora da barra de menu
- Assim como um menu comum, um menu *pop-up* pode conter itens de menu e separadores
- Métodos de **JPopupMenu**

```
JPopupMenu()
JPopupMenu(String label)
JMenuItem add(String name)
JMenuItem add(JMenuItem menuItem)
void addSeparator()
void pack()
void show(Component c, int x, int y)
```

Exemplo de **JPopupMenu**

```
JFrame f = new JFrame("Teste");
JPopupMenu p = new JPopupMenu();
JMenu m = new JMenu("Arquivo");
m.add("Sair"); p.add(m);
f.addMouseListener(new MouseAdapter() {
  public void mousePressed(MouseEvent ev) {
    if (ev.isPopupTrigger())
      p.show((Component)ev.getSource(), ev.getX(), ev.getY());
  }
  public void mouseReleased(MouseEvent ev) {
    if (ev.isPopupTrigger())
      p.show((Component)ev.getSource(), ev.getX(), ev.getY());
  }});
f.pack();
f.setVisible(true);
```


Diálogos Pré-definidos

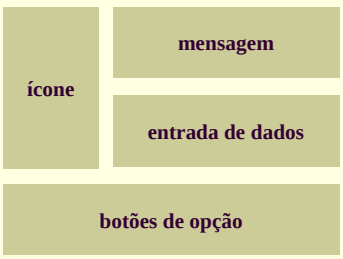
- O Swing oferece um conjunto de diálogos simples pré-definidos para uso em interações breves com o usuário
  - mensagens de erro, de alerta
  - obtenção de uma confirmação
  - entrada de um único campo de texto
- Esses diálogos são *modais*

JOptionPane

- Classe que permite exibir diferentes tipos de caixas de diálogo.
- Cinco métodos estáticos principais
  - JOptionPane.showMessageDialog
    - Icon, message, OK button
  - JOptionPane.showConfirmDialog
    - Icon, message, and buttons:
      OK, OK/Cancel, Yes/No, or Yes/No/Cancel
  - JOptionPane.showInputDialog (2 versões)
    - Icon, message, textfield or combo box, buttons
  - JOptionPane.showOptionDialog
    - Icon, message, array de butões ou de outros componentes

Classe **JOptionPane**

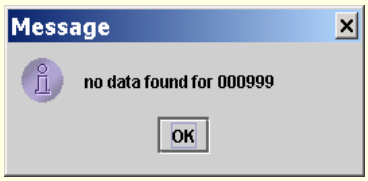
- Métodos estáticos para a criação desses diálogos simples
- Estrutura básica:



JOptionPane (cont.)

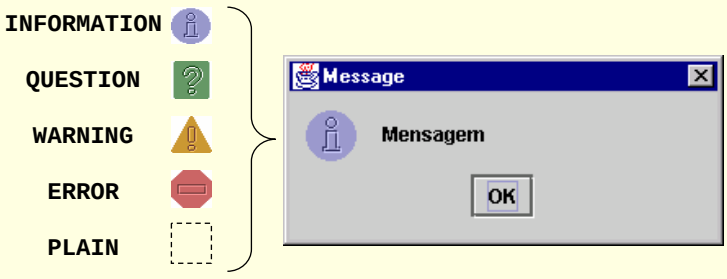
- Exemplo de uso de JOptionPane

```
JOptionPane.showMessageDialog(null,
            "no data found for " + parameter);
```

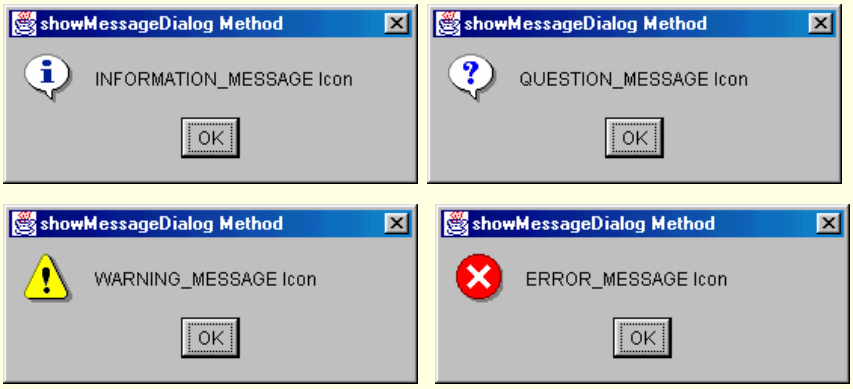


MessageDialog

- Exibe uma mensagem e aguarda OK do usuário



JOptionPane Message Dialogs

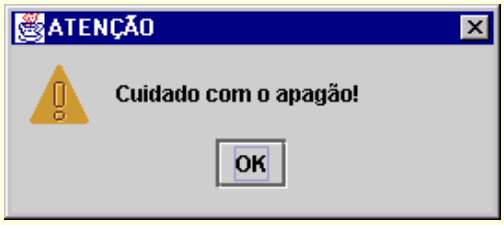


Método **showMessageDialog**

```
void showMessageDialog(Component parentComponent,
                       Object message);
void showMessageDialog(Component parentComponent,
                       Object message,
                              String title,
                              int messageType);
void showMessageDialog(Component parentComponent,
                       Object message,
                              String title,
                              int messageType,
                              Icon icon);
```

Exemplo de **MessageDialog**

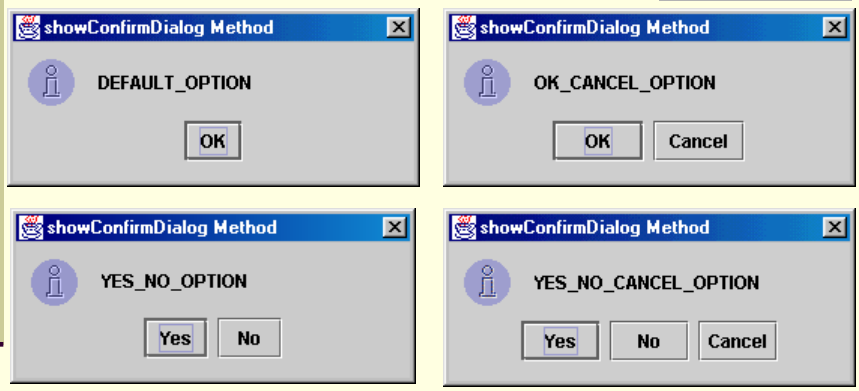
```
JOptionPane.showMessageDialog(janela,
                              "Cuidado com o apagão!",
                              "ATENÇÃO",
                              JOptionPane.WARNING_MESSAGE);
```



ConfirmDialog

- Exibe uma mensagem e obtém uma confirmação (YES/NO, OK/CANCEL)
- Conjuntos de botões de opção (***optionType***):
  - **`JOptionPane.DEFAULT_OPTION`**
  - **`JOptionPane.YES_NO_OPTION`**
  - **`JOptionPane.YES_NO_CANCEL_OPTION`**
  - **`JOptionPane.OK_CANCEL_OPTION`**

JOptionPane: Confirmation Dialogs



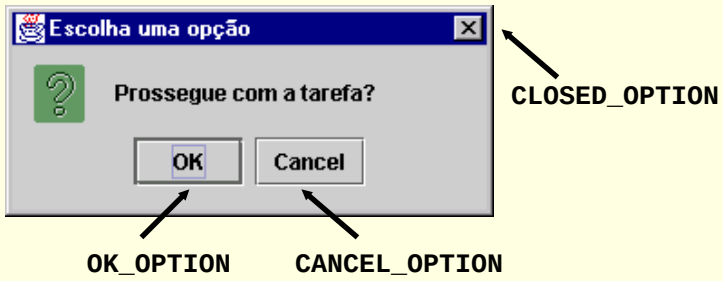
(Java L&F)

Método **showConfirmDialog**

```
int showConfirmDialog(Component parentComponent,
                Object message);
int showConfirmDialog(Component parentComponent,
                Object message,
                      String title,
                      int optionType);
int showConfirmDialog(Component parentComponent,
                Object message,
                      String title,
                      int optionType
                      int messageType,
                      Icon icon);
```

Exemplo de **ConfirmDialog**

```
int resp = JOptionPane.showConfirmDialog(janela,
                                "Prossegue com a tarefa?",
                                "Escolha uma opção",
                                JOptionPane.OK_CANCEL_OPTION);
```



InputDialog

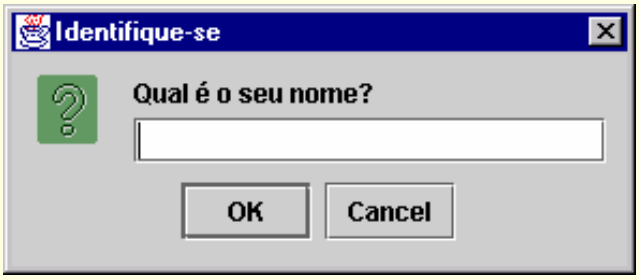
- Exibe uma mensagem e obtém um valor de entrada do usuário
  - campo de texto editável
  - *combo box*

Método **showInputDialog**

```
String showInputDialog(Component parentComponent,
                       Object message);
String showInputDialog(Component parentComponent,
                       Object message,
                              String title,
                              int messageType);
Object showInputDialog(Component parentComponent,
                       Object message,
                              String title,
                              int messageType
                              Icon icon,
                              Object[] selectionValues,
                              Object defaultSelection);
```


Exemplo de **InputDialog**

```
String nome = JOptionPane.showInputDialog(janela,
                                "Qual é o seu nome?",
                                "Identifique-se",
                                JOptionPane.QUESTION_MESSAGE);
```



OptionDialog

- Exibe uma mensagem (ou objeto) e obtém uma opção escolhida pelo usuário
- O número de botões e seus textos são configuráveis
- A opção *default* é configurável

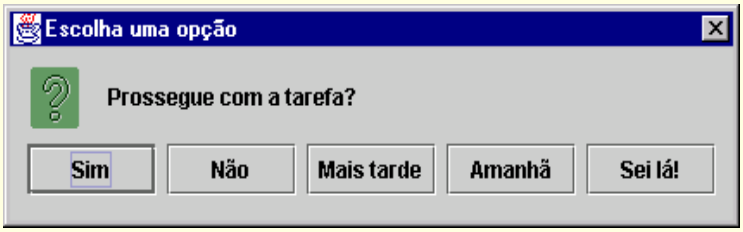
Método **showOptionDialog**

```
int showOptionDialog(Component parentComponent,
                     Object message,
                            String title,
                            int optionType
                            int messageType,
                            Icon icon,
                            Object[] options,
                            Object initialValue);
```

Exemplo de **OptionDialog**

```
Object[] opções = {"Sim", "Não", "Mais Tarde",
                   "Amanhã", "Sei lá!"};
int resp = JOptionPane.showOptionDialog(janela,
                            "Prossegue com a tarefa?",
                            "Escolha uma opção",
                            JOptionPane.DEFAULT_OPTION,
                            JOptionPane.QUESTION_MESSAGE,
                            null,
                            opções,
                            opções[0]);
```


Exemplo de **OptionDialog**



Classe **JFileChooser**

- É comum uma aplicação abrir e salvar arquivos
- A classe **JFileChooser** implementa uma caixa de diálogo que permite que o usuário navegue pelo sistema de arquivos
  - semelhante às usadas por aplicações “nativas”

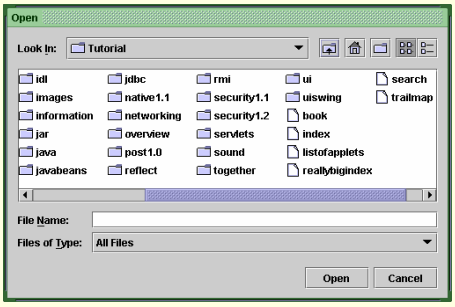
Métodos de **JFileChooser**

```
void setCurrentDirectory(File dir)
void setSelectedFile(File file)
void setMultiSelectionEnabled(boolean b)
void setFileFilter(FileFilter filter)

int showOpenDialog(Component parent)
int showSaveDialog(Component parent)
int showDialog(Component parent,
         String approveButtonText)

File getSelectedFile()
File[] getSelectedFiles()
```

Exemplo de **JFileChooser**

```
JFileChooser chooser = new JFileChooser();
chooser.setCurrentDirectory(new File("c:\\Tutorial"));
int res = chooser.showOpenDialog(janela);
if (res == JFileChooser.APPROVE_OPTION) {
    File file = chooser.getSelectedFile();
    System.out.println(file.getName());
}
```

FileChooserDemo.java



Classe **FileFilter**

- Utilizada para a implementação de filtros que permitem restringir os tipos de arquivos exibidos em um diálogo de seleção
  - FileFilter é uma classe abstrata
- A subclasse de **FileFilter** deverá implementar os métodos:

```
boolean accept(File file)
String getDescription()
```

Exemplo de **FileFilter**

```
import javax.swing.filechooser.FileFilter;

public class GifFilter extends FileFilter
{
  public boolean accept(File f) {
    return f.getName().toLowerCase().endsWith(".gif")
           || f.isDirectory();
  }
  public String getDescription() {
    return "Arquivos GIF";
  }
}
```

Exercícios – Questão 26

POO-Java

Conclusões

Conclusões

- Componentes essenciais de uma aplicação Swing:
  - Contêiners são janelas ou painéis que contêm componentes.
  - Layouts especificam como arranjar componentes em um contêiner.
  - Componentes: são os controles da interface gráfica com o usuário.
  - Ouvintes (listeners) são conectados a componentes e contém o código que é executado quando o componente é usado.
    - É desse modo que uma ação do usuário sobre um componente é conectada a um método Java.

Conclusões

- Alguns IDEs têm eles próprios facilidades de construção da interface gráfica (editores de formulários)
  - e.g. NetBeans - www.netbeans.org
- Também há ferramentas específicas para a criação de GUIs em Java. Exemplos são:
  - XUI - http://xui.sourceforge.net/
  - UICompiler - http://uic.sourceforge.net/