

Interface Gráfica Swing

Leonardo Gresta Paulino Murta
leomurta@ic.uff.br

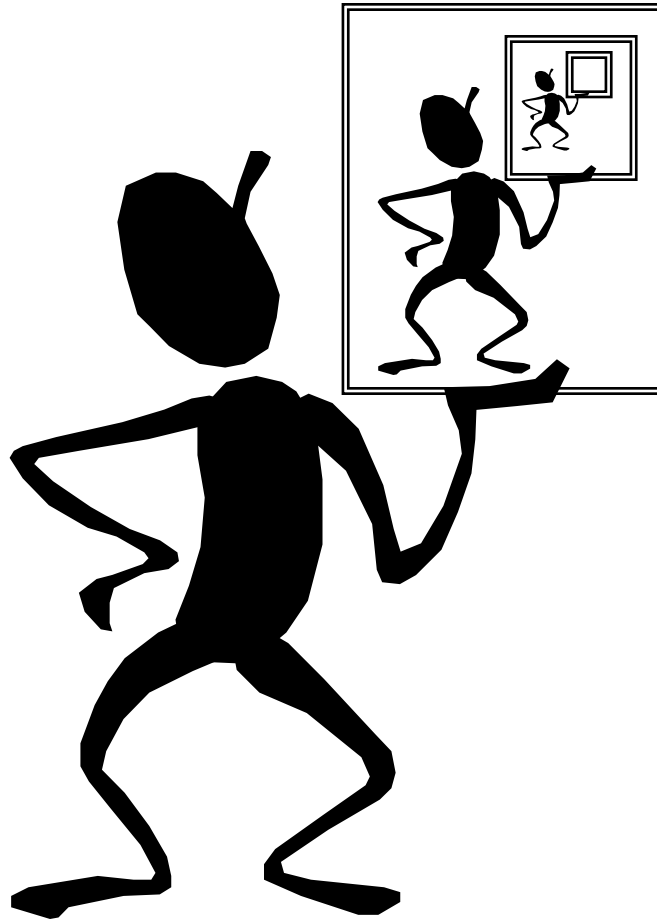
Aula de hoje

- Criação de interface gráfica via biblioteca Swing
 - Containers
 - Componentes
 - Menu
 - Layout
 - Bordas
 - Eventos

Pacotes do Swing

- As classes do Swing estão distribuídas por diversos pacotes. Os principais pacotes são:
 - `javax.swing.*`
 - `javax.swing.event.*`
- Algumas classes dos pacotes antigas da AWT também são utilizados pelo Swing:
 - `import java.awt.*`
 - `import java.awt.event.*`

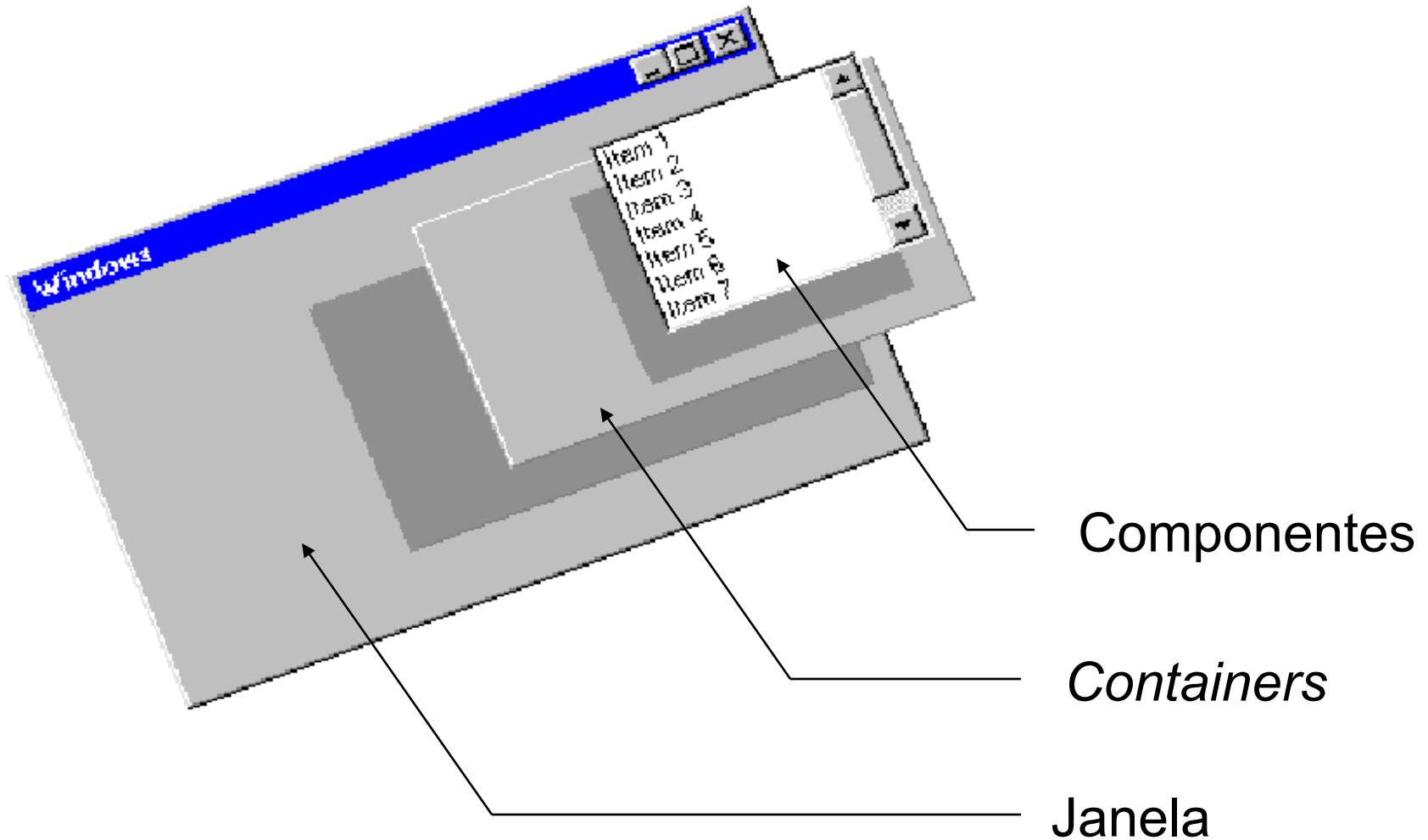
Componentes e Containers



Componentes e Containers

- A interface com o usuário é orientada a janelas
 - Os elementos de interface com o usuário são classificados como janelas, *containers* e componentes
 - As janelas são a base da interface com o usuário, contendo os demais elementos
 - Os *containers* são grupos de componentes, apresentados em uma região de tela definida
 - Os componentes são os principais elementos de interface, sendo utilizados diretamente pelo usuário

Componentes e Containers



Componentes e Containers

- A janela (JFrame) é o *container* de mais alto nível
 - A janela existe para prover espaço para apresentação dos componentes Swing
- O painel (JPanel) é um *container* intermediário
 - Os painéis existem para controlar o posicionamento dos componentes
- Componentes atômicos, como botões (JButton) e linhas de edição (JTextField), realizam a interação com o usuário propriamente dita

Janelas - *JFrame*

- A classe *JFrame* representa uma janela Swing
 - Seu construtor pode receber uma string com o título da janela
 - Toda janela possui um painel invisível (*ContentPane*)
 - Os componentes da janela são inseridos neste painel
 - A janela pode conter uma barra de menu

```
JFrame frame = new JFrame("Alo, Mundo");  
frame.getContentPane().add (new JButton ("Teste"));
```


Painéis Intermediários - *JPanel*

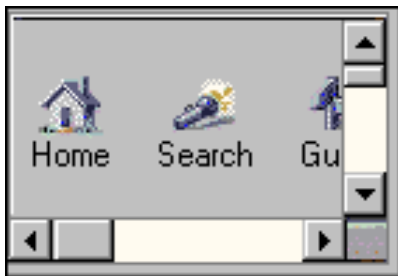
- A classe `JPanel` representa um painel simples
 - `JPanel` é o *container* intermediário mais simples
 - O painel pode ser inserido em uma janela ou outro painel
 - O método *add()* insere o painel na janela

```
JFrame frame;  
JPanel painel;
```

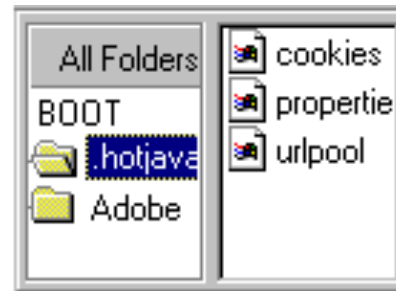
```
frame = new JFrame("Alo, Mundo");  
painel = new JPanel ();  
frame.getContentPane().add (painel);
```

Outros Painéis Intermediários

- Swing possui outros painéis intermediários:



JScrollPane



JSplitPane



JTabbedPane



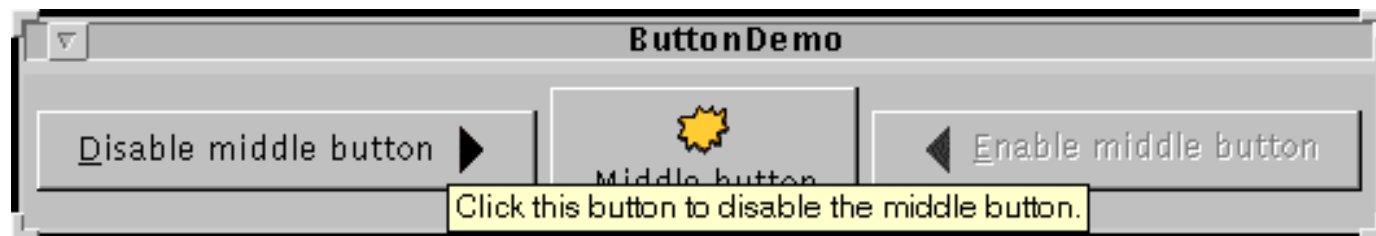
JToolBar

Componentes

- Os componentes Swing herdam da classe *JComponent*
- Esta classe oferece recursos para o desenvolvedor de aplicações com interface gráfica
 - Textos de ajuda
 - Bordas
 - Tratamento de eventos
 - Outros métodos de suporte

Texto de Ajuda

- Os componentes Swing suportam textos de ajuda
 - O método *setToolTipText()* indica o texto de ajuda de um componente
 - Este método recebe uma String com o texto de ajuda como um parâmetro
 - O texto de ajuda é apresentado quando o usuário para o cursor do mouse sobre o componente



Métodos de Suporte

- Visualização e acesso ao componente
 - O método *setEnabled()* recebe um booleano que permite que o desenvolvedor habilite ou inabilite um componente
 - O método *isEnabled()* determina se um componente está habilitado ou inabilitado, retornando um booleano
 - O método *setVisible()* recebe um booleano que permite que o desenvolvedor apresente ou esconda um componente
 - O método *isVisible()* determina se um componente está visível, retornando um booleano

Métodos de Suporte

- Fonte e cursor do componente
 - O método *setFont()* altera a fonte de caracteres de um componente para a fonte recebida como parâmetro
 - O método *getFont()* retorna a fonte de caracteres sendo utilizada por um componente
 - O método *setCursor()* altera o cursor de mouse utilizado por um componente para o cursor recebido como parâmetro
 - O método *getCursor()* retorna o cursor de mouse sendo utilizado por um componente

Métodos de Suporte

- Tamanho e posição do componente:
 - O método *getWidth()* retorna a largura do componente em número de pontos
 - O método *getHeight()* retorna a altura do componente em número de pontos
 - O método *getSize()* retorna as dimensões do componente em um objeto da classe *Dimension*
 - O método *getBounds()* retorna a área ocupada por um componente em um objeto da classe *Rectangle*

Componentes - Exemplo

- Exemplo de trecho de código:

```
JButton botao = new JButton ();
```

```
botao.setToolTipText ("Texto de ajuda do botão");
```

```
botao.setEnabled (true);
```

```
boolean visivel = botao.isVisible();
```


Botões

- Swing possui diversos tipos de botões
 - JButton: botão simples
 - JCheckBox: caixa de seleção
 - JRadioButton: caixa de seleção com múltiplas opções
- Os botões possuem versões utilizadas em menus
 - JMenuItem: item de menu
 - JCheckBoxMenuItem: caixa de seleção para menus
 - JRadioButtonMenuItem: caixa de seleção múltipla para menus

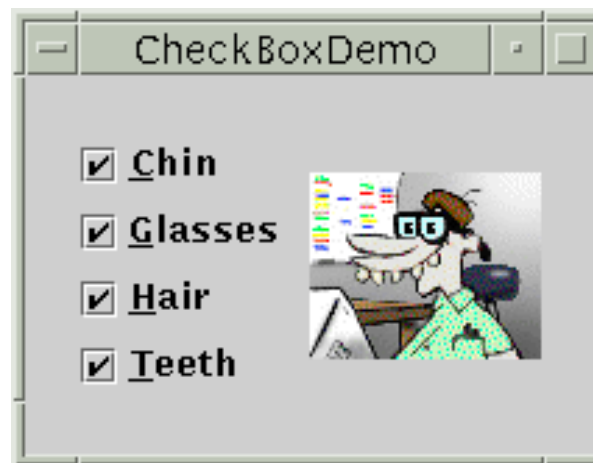
Botões

- Características dos botões Swing
 - Apresentam texto e imagens
 - Podem possuir um mnemônico utilizado como *hotkey*
 - Controlam o posicionamento do título do botão

```
ImageIcon icone = new ImageIcon("images/right.gif");  
JButton b1 = new JButton ("Teste", icone);  
b1.setVerticalTextPosition (AbstractButton.CENTER);  
b1.setHorizontalTextPosition (AbstractButton.LEFT);  
b1.setMnemonic(KeyEvent.VK_D);
```

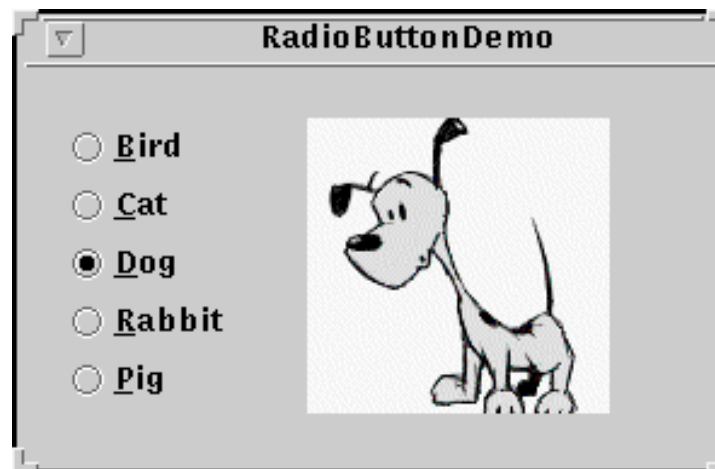
Caixas de Seleção - JCheckBox

- Possuem as características de um botão genérico
 - O método *getSelected()* determina se o check box foi selecionado pelo usuário
 - O método *setSelected()* seleciona ou deseleciona o check box, de acordo com seu parâmetro booleano



Caixas de Seleção - JRadioButton

- Radio buttons são criados em grupos
 - Somente um botão de um grupo pode estar selecionado em um determinado instante
 - A classe *ButtonGroup* permite a criação de grupos de botões



Caixas de Seleção - JRadioButton

- Exemplo de trecho de código:

```
JRadioButton birdButton = new JRadioButton("Bird");  
JRadioButton catButton = new JRadioButton("Cat");  
JRadioButton dogButton = new JRadioButton("Dog");
```

```
ButtonGroup group = new ButtonGroup();  
group.add(birdButton);  
group.add(catButton);  
group.add(dogButton);
```

```
birdButton.setSelected(true);
```

Combo Boxes

- Combo boxes permitem a seleção de um item de uma lista, ocupando um espaço delimitado da janela
 - Apenas um item está selecionado em cada instante
 - A combo box pode ser inicializada com um array ou com um conjunto de Strings



Combo Boxes

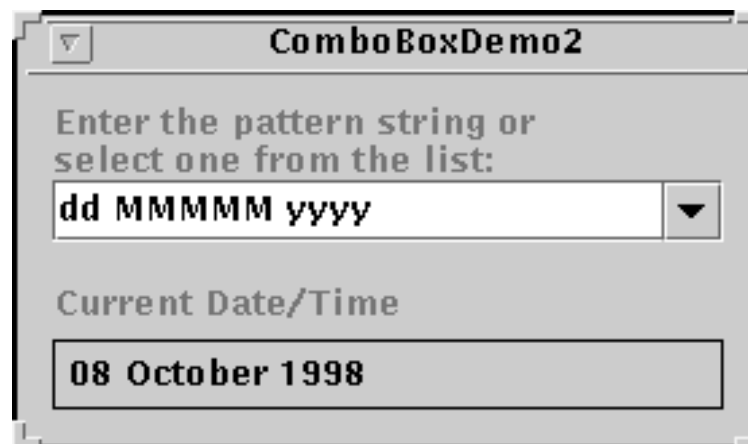
- Acesso aos dados
 - O método *setSelectedIndex()* altera o item selecionado, indicando o índice do novo item na lista de opções
 - O método *getSelectedIndex()* retorna o índice do item selecionado na lista de opções

```
String[] pets = { "Bird", "Cat", "Dog", "Rabbit", "Pig" };
```

```
JComboBox petList = new JComboBox (pets);  
petList.setSelectedIndex(4);
```

Combo Boxes

- Combo boxes podem ser editáveis
 - O método *setEditable()* altera o modo de edição de uma combo box de acordo com seu parâmetro booleano
 - O método *getSelectedItem()* retorna o item selecionado na combo box ou o texto digitado pelo usuário



Combo Boxes

- Modelos
 - Uma combo box pode ser inicializada com um modelo
 - O modelo permite a atualização dos itens da combo
 - O programa pode inserir e remover itens durante a execução
 - O método *setModel()* altera o modelo de uma combo box
- *DefaultComboModel*
 - O modelo é representado pela interface *ComboModel*
 - A classe *DefaultComboModel* implementa a interface
 - A classe possui métodos para inserção e remoção de itens

Combo Boxes

- Alguns métodos da *DefaultComboBoxModel*
 - O método *insertItemAt()* insere o item recebido como parâmetro em uma determinada posição da combo
 - O método *addItem()* insere um novo item ao fim da combo
 - O método *removeAllItems()* remove todos os itens da combo
 - O método *removeItem()* remove da combo o item recebido como parâmetro
 - O método *getItemCount()* retorna o número de itens da combo

Combo Boxes

- Exemplo de trecho de código:

```
DefaultComboModel modelo = new DefaultComboModel ();  
modelo.addElement ("Item 1");  
modelo.addElement ("Item 2");  
modelo.addElement ("Item 3");  
modelo.addElement ("Item 4");
```

```
JComboBox combo = new JComboBox (modelo);  
modelo.addElement ("Item 5");
```

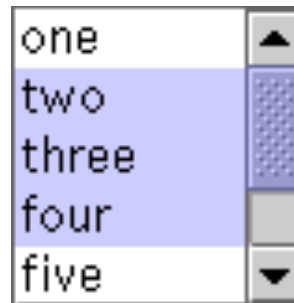
Listas

- Listas apresentam diversos itens ao mesmo tempo
 - Como na combo box, a lista pode ser inicializada com um vetor, uma lista de strings ou um modelo (*DefaultListModel*)



Listas

- Listas permitem diversos modos de seleção de itens
 - O método *setSelectionMode()* altera o modo de seleção dos itens de uma lista
 - SINGLE_SELECTION
 - SINGLE_INTERVAL_SELECTION
 - MULTIPLE_INTERVAL_SELECTION



Listas

- Listas devem ser inseridas em painéis de rolamento

```
String[] pessoas = { "Eu", "Você", "Ele", "Ela"};
```

```
JList lista = new JList (pessoas);
```

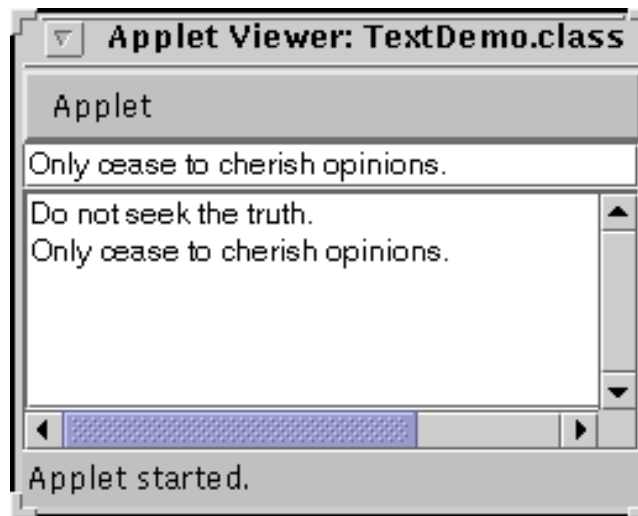
```
lista.setSelectionMode (ListSelectionMode.SINGLE_SELECTION);
```

```
JScrollPane scroll= new JScrollPane (lista);
```

```
painel.add(scroll);
```

Linhas/Áreas de Texto

- Permitem a edição de textos
 - O construtor da linha de edição indica seu tamanho desejado
 - O construtor não limita o tamanho do texto editado
 - O método *setText()* altera o texto em edição na linha
 - O método *getText()* retorna o texto editado pelo usuário



Linhas/Áreas de Texto

- Exemplo de trecho de código

```
JTextField linha = new JTextField (20);  
linha.setText ("Alo, Mundo !");  
linha.selectAll ();
```

*Linha de texto
simples*

```
JTextArea area = new JTextArea (5, 20);  
area.setText ("Testando 1, 2, 3");
```

*Área de texto
com múltiplas
linhas*

```
JScrollPane scroll= new JScrollPane (area);  
painel.add(scroll);
```

*Áreas de texto
são inseridas
em painéis*

Linhas/Áreas de Texto

- Métodos de suporte
 - O método *setEditable()* indica se a linha de texto é editável ou *read-only*, de acordo com seu parâmetro booleano
 - O método *isEditable()* determina se a linha de texto é editável ou *read-only*, retornando um booleano
 - O método *setHorizontalAlignment()* indica como o texto é alinhado na linha de edição
 - `TextField.LEFT`
 - `TextField.CENTER`
 - `TextField.RIGHT`

Rótulos

- Rótulos permitem a apresentação de textos e imagens
 - Os métodos *getText()* e *setText()* permitem consultar e alterar o texto de um rótulo
 - Os métodos *getIcon()* e *setIcon()* permitem consultar e alterar a imagem apresentada no rótulo



Rótulos

- Exemplo de trecho de código:

```
ImageIcon icon = new ImageIcon ("boom.gif");
```

```
JLabel label1 = new JLabel ("Image and Text", icon, JLabel.CENTER);  
label1.setVerticalTextPosition (JLabel.BOTTOM);  
label1.setHorizontalTextPosition (JLabel.CENTER);  
painel.add (label1);
```

*Alinhamento do
texto*



```
label2 = new JLabel ("Text-Only Label");  
painel.add (label2);
```

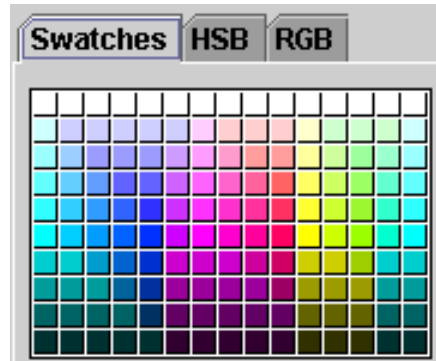
```
label3 = new JLabel (icon);  
painel.add (label3);
```

Outros Componentes

- Swing oferece uma série de outros componentes



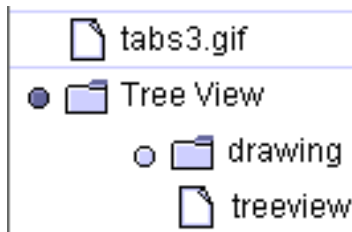
JSlider



JColorChooser

First Na...	Last Name
Mark	Andrews
Tom	Ball
Alan	Chung
Jeff	Dinkins

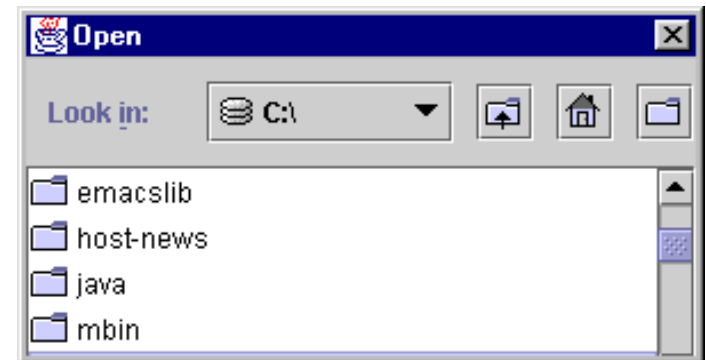
JTable



JTree

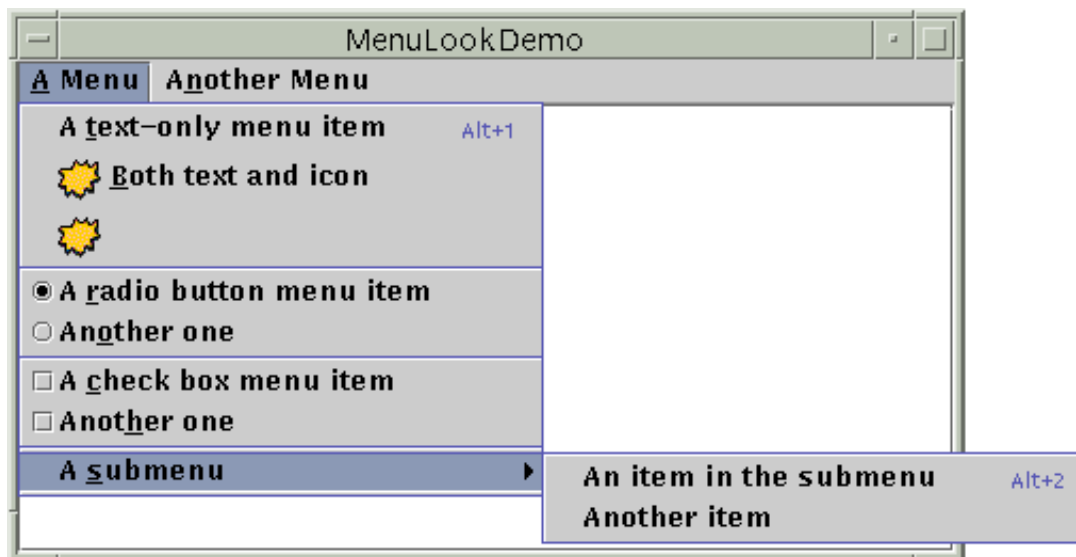


JProgressBar



JFileChooser

Menus



Menus em Barra

- Exemplo de trecho de código

```
JMenuBar menuBar = new JMenuBar();  
frame.setJMenuBar(menuBar);
```

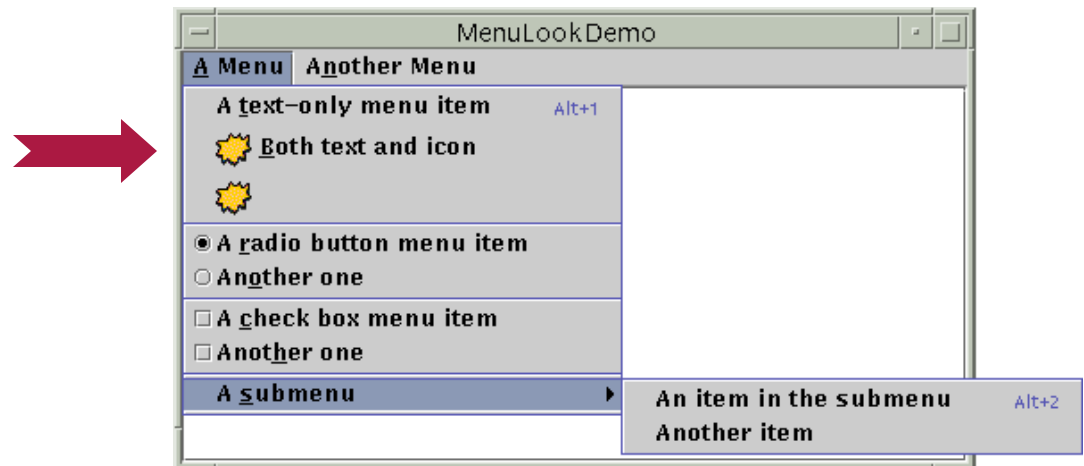
```
JMenu menu = new JMenu ("Menu");  
menuBar.add (menu);
```

```
JMenuItem menuItem = new JMenuItem ("Item de Menu", KeyEvent.VK_T);  
menu.add(menuItem);
```

Itens de Menu

- Podem ser inicializados de diversas formas

```
menu.add(new JMenuItem (titulo, letra_chave));
menu.add(new JMenuItem (titulo, icone));
menu.add(new JMenuItem (icone));
menu.addSeparator ();
```

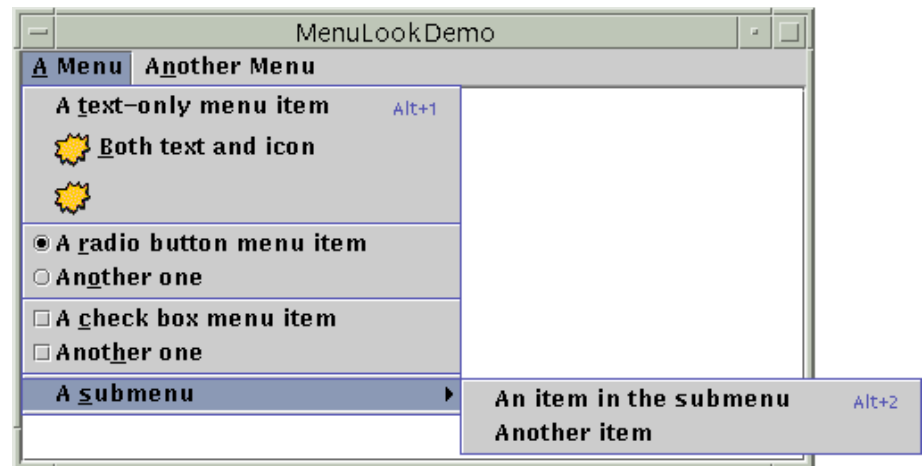


Itens de Menu - Radio Buttons

```
ButtonGroup group = new ButtonGroup();
```

```
rbMenuItem = new JRadioButtonMenuItem("A Radio button menu item");
rbMenuItem.setSelected(true);
menu.add (rbMenuItem);
group.add (rbMenuItem);
```

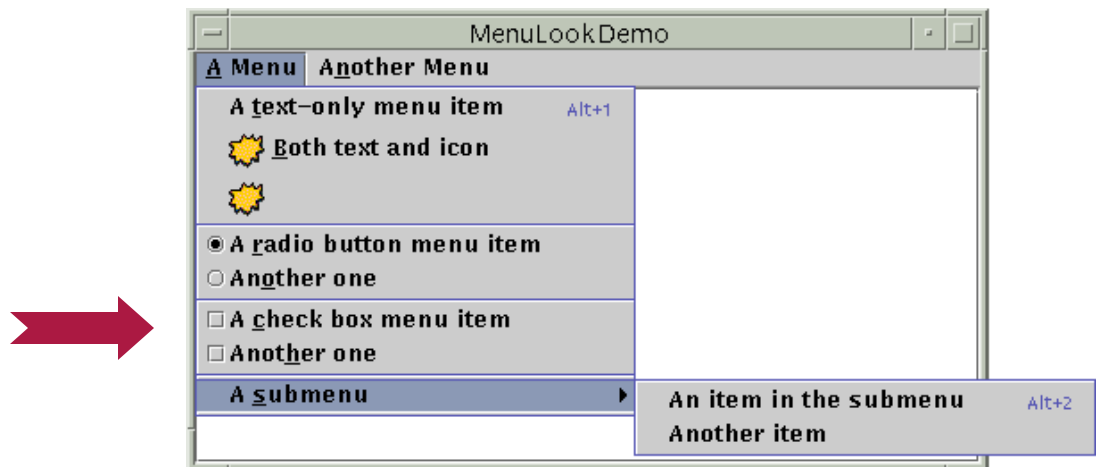
```
rbMenuItem = new JRadioButtonMenuItem ("Another One");
menu.add(rbMenuItem);
group.add(rbMenuItem);
```



Itens de Menu - Check Boxes

```
cbMenuItem = new JCheckBoxMenuItem ("A check box menu item");
cbMenuItem.setMnemonic (KeyEvent.VK_C);
menu.add (cbMenuItem);
```

```
cbMenuItem = new JCheckBoxMenuItem ("Another one");
cbMenuItem.setMnemonic (KeyEvent.VK_H);
menu.add (cbMenuItem);
```

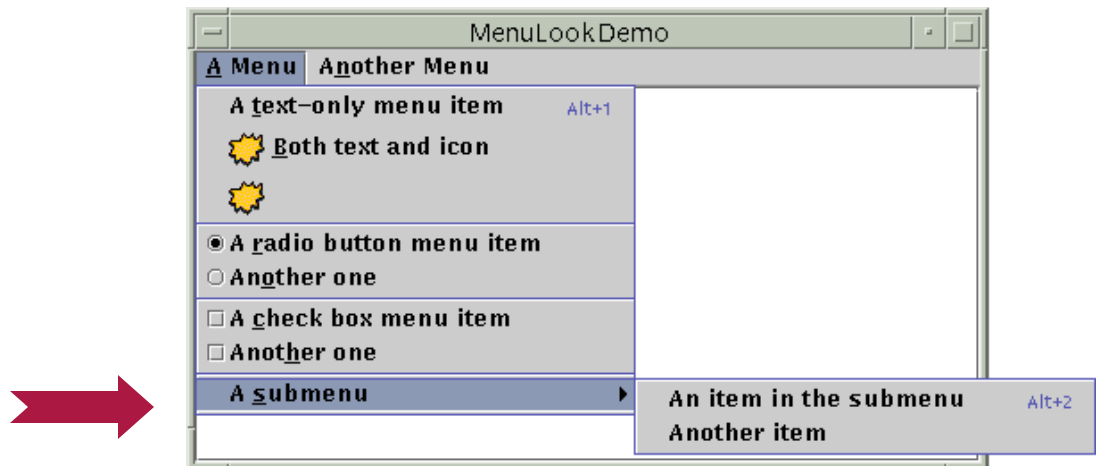


Itens de Menu - Submenus

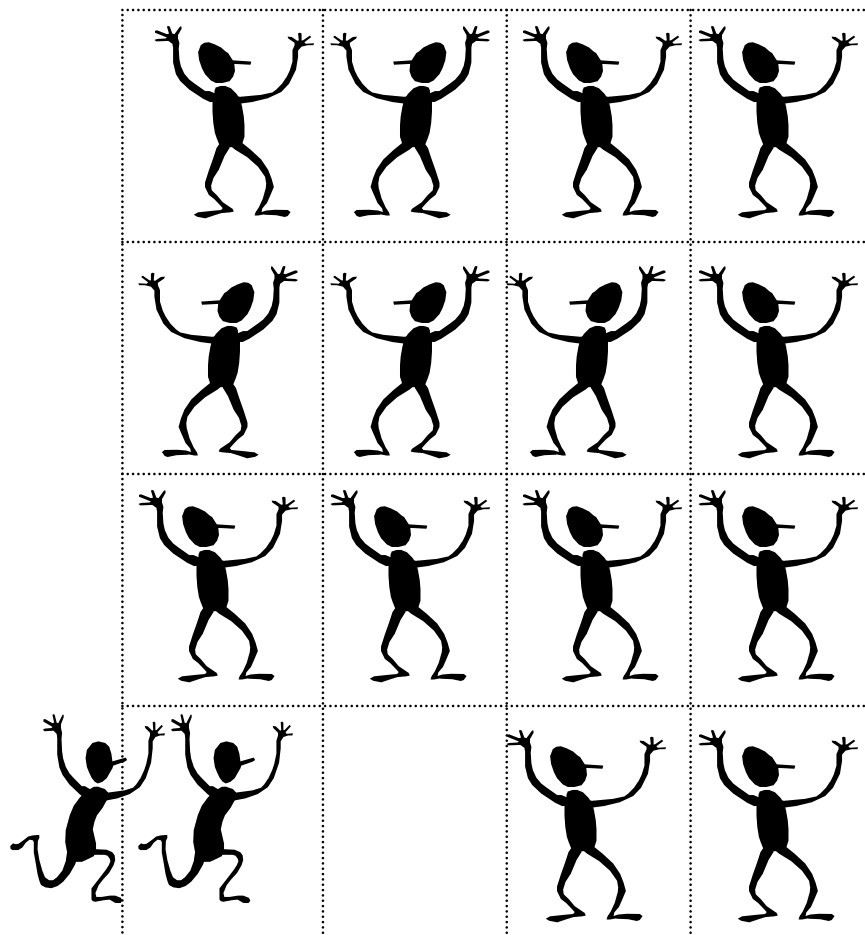
```
submenu = new JMenu ("A submenu");
menu.add (submenu);
```

```
menuItem = new JMenuItem ("An item in the submenu");
submenu.add (menuItem);
```

```
menuItem = new JMenuItem ("Another item");
submenu.add (menuItem);
```



Layout



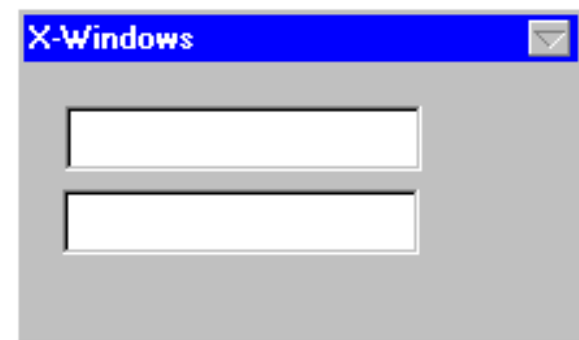
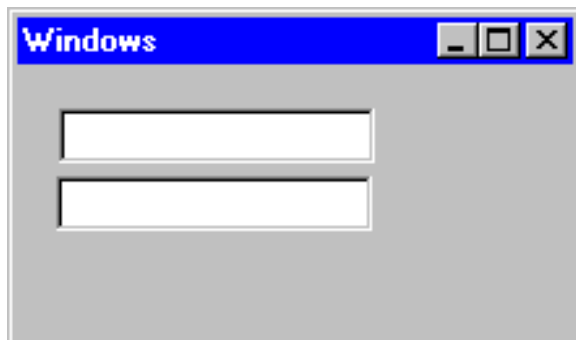
Posicionamento de Componentes

- A independência de plataforma gera alguns problemas
 - Alguns controles ocupam áreas de tela diferentes em plataformas distintas
 - Exemplo: uma linha de edição de texto ocupa uma área de tela menor no Windows que no X-Windows
 - Problema: como posicionar os controles na janela levando em conta o tamanho diferente em cada plataforma ?



Layout Managers

- Solução: controladores de posicionamento
 - Cada *container* indica seu controlador de posicionamento
 - Os controles são inseridos no *container* sem posição
 - O controlador de posicionamento ajusta a posição e tamanho dos controles, de acordo com as características da plataforma



Layout Managers

- O controle de posicionamento determina o tamanho e a posição dos componentes de um painel
 - Todos os *containers* possuem um *layout manager*
 - O *layout manager* é um objeto responsável por posicionar e determinar o tamanho dos componentes do *container*
 - Os componentes podem indicar seus tamanhos e posições desejadas, mas o *layout manager* decide sobre suas dimensões finais em cada *container*

Layout Managers

- Sempre que o desenvolvedor utilizar o método *add()* de um *container*, o *layout manager* deve ser considerado
 - Alguns *layout managers* exigem a especificação de um segundo parâmetro no método *add()*
 - O segundo parâmetro traz informações sobre a posição do componente inserido
 - Por exemplo, o *layout manager BorderLayout* exige que o desenvolvedor especifique a posição relativa de seus componentes.

Indicando o *layout manager* de um painel

- O método *setLayout()* de um painel permite alterar seu *layout manager*, que é passado como parâmetro

```
JPanel painel = new JPanel();  
painel.setLayout (new BorderLayout ());
```


Tamanho desejado pelos componentes

- Um componente pode indicar suas dimensões para seus *layout managers*
 - O método *setMinimumSize()* indica o tamanho mínimo do componente
 - O método *setMaximumSize()* indica o tamanho máximo do componente
 - O método *setPreferredSize()* indica o tamanho desejado pelo componente
 - O desenvolvedor pode chamar estes métodos após criar seus componentes

Layout Managers - *FlowLayout*

- Componentes são organizados em linhas
 - Cada componente ocupa o seu tamanho desejado
 - Se o espaço horizontal não for suficiente para todos os componentes, o *layout manager* utiliza diversas linhas
 - Dentro de uma linha, o desenvolvedor pode especificar o alinhamento dos componentes



Layout Manager - *FlowLayout*

- Exemplo de trecho de código:

```
Container painel = frame.getContentPane();
```

```
painel.setLayout (new FlowLayout ());  
painel.add (new JButton ("Button 1"));  
painel.add (new JButton ("Button 2"));  
painel.add (new JButton ("Button 3"));
```

Layout Managers - *BorderLayout*

- Componentes são organizados nas bordas do *container*
 - Quando um componente é inserido, o método *add()* deve receber sua posição relativa no container



Layout Manager - *BorderLayout*

- Exemplo de trecho de código

```
Container painel = frame.getContentPane();
```

```
painel.setLayout (new BorderLayout ());  
painel.add (new JButton ("Button 1"), BorderLayout.NORTH);  
painel.add (new JButton ("Button 2"), BorderLayout.CENTER);  
painel.add (new JButton ("Button 3"), BorderLayout.WEST);  
painel.add (new JButton ("Button 4"), BorderLayout.SOUTH);  
painel.add (new JButton ("Button 5"), BorderLayout.EAST);
```

Layout Managers - *GridLayout*

- Componentes são organizados em uma grade
 - Cada componente ocupa uma célula da grade
 - Todas as células possuem o mesmo tamanho
 - O tamanho das células é determinado pelo espaço ocupado pelo *container*



Layout Manager - *GridLayout*

- Exemplo de trecho de código:

```
Container painel = frame.getContentPane();
```

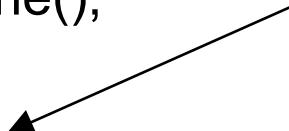
```
painel.setLayout(new GridLayout(0,2));
```

```
painel.add(new JButton("Button 1"));
```

```
painel.add(new JButton("Button 2"));
```

```
painel.add(new JButton("Button 3"));
```

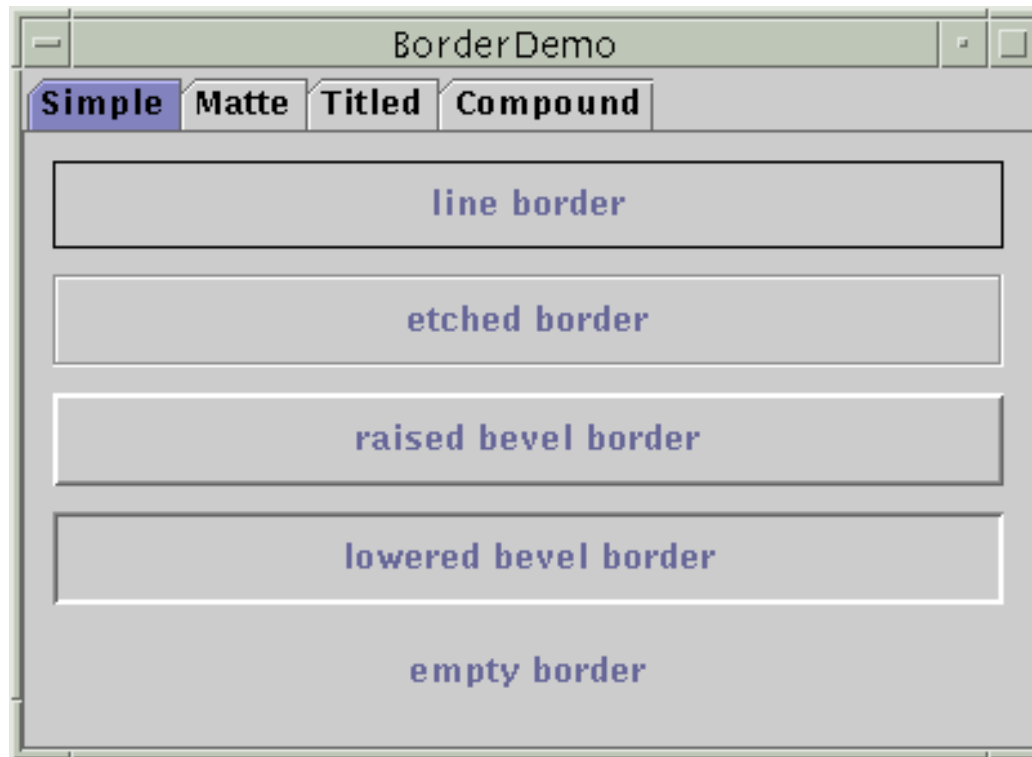
*2 colunas e
quantas linhas
forem necessárias*



Como o posicionamento é realizado ?

- Depois de inserir os componentes de uma janela, o desenvolvedor deve chamar o método *pack()*
 - O *layout manager* calcula o tamanho desejado da janela, somando o tamanho de suas bordas ao tamanho preferido dos componentes inseridos diretamente na janela.

Bordas



Bordas

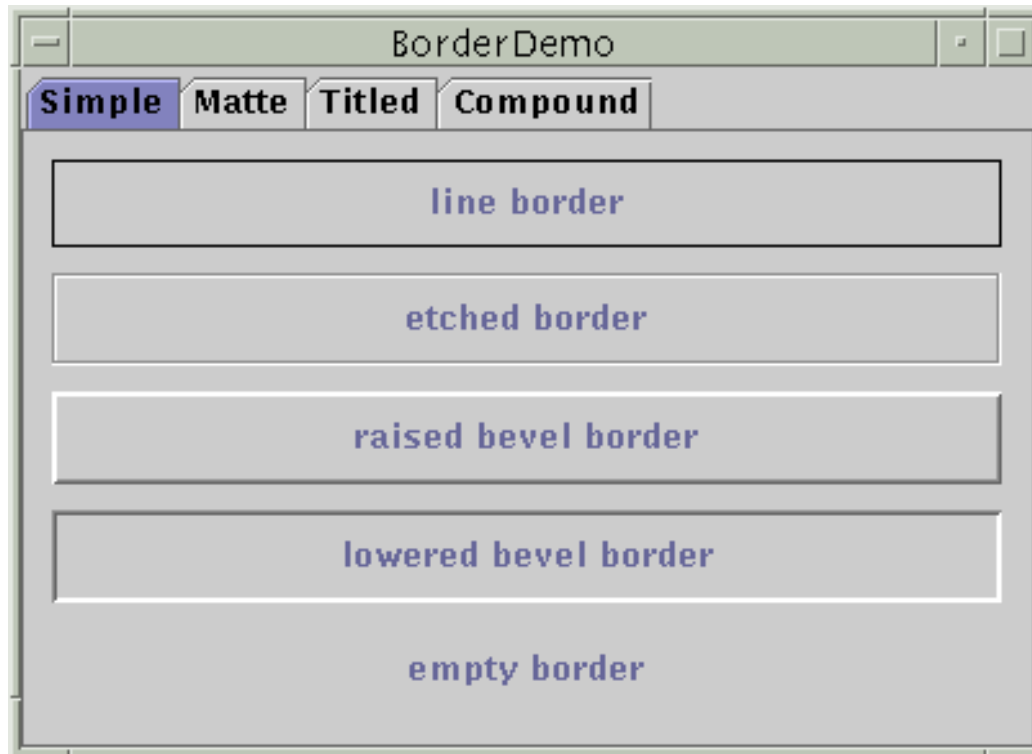
- O método *setBorder()* indica a borda do componente
 - Swing define uma fábrica que deve ser utilizada para criar as bordas dos componentes

```
JPanel panel = new JPanel();  
panel.setBorder(BorderFactory.createLineBorder (Color.black));
```



Bordas Simples

- Representam os tipos mais simples de bordas



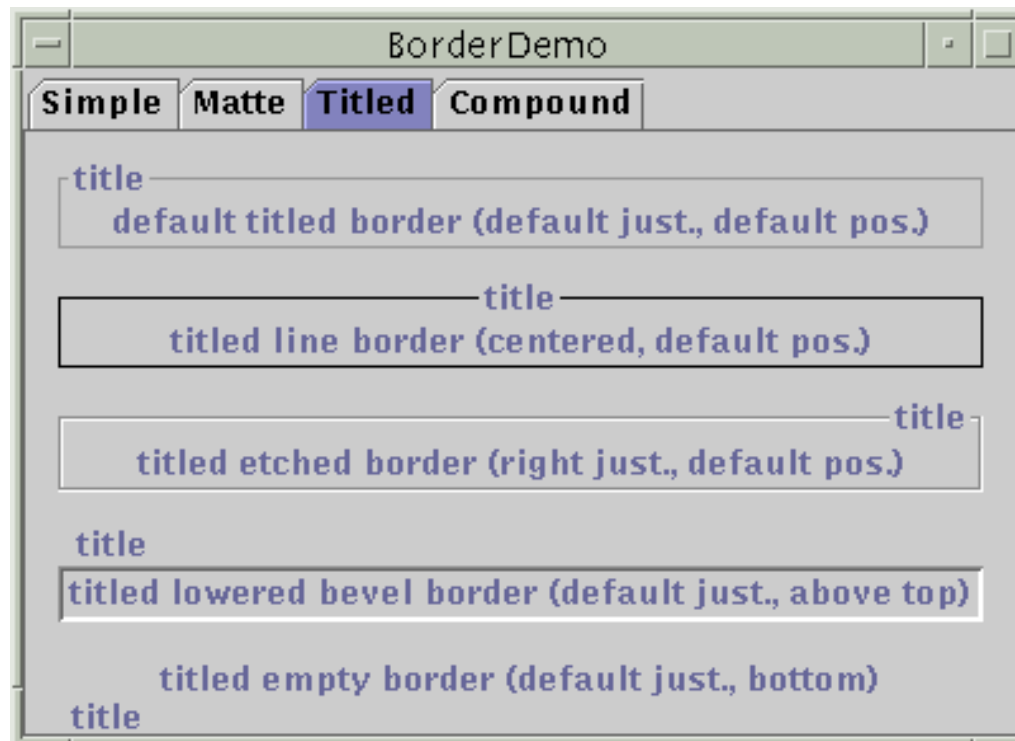
Bordas Simples

- Exemplo de criação das bordas a partir da fábrica

```
comp1.setBorder (BorderFactory.createLineBorder(Color.black);  
comp2.setBorder (BorderFactory.createRaisedBevelBorder());  
comp3.setBorder (BorderFactory.createEtchedBorder());  
comp4.setBorder (BorderFactory.createLoweredBevelBorder());  
comp5.setBorder (BorderFactory.createEmptyBorder());
```

Bordas com Títulos

- Bordas com título são bordas simples que apresentam títulos. Se nenhuma borda for especificada junto ao título, o *look & feel* utiliza uma borda *default*.



Bordas com Títulos

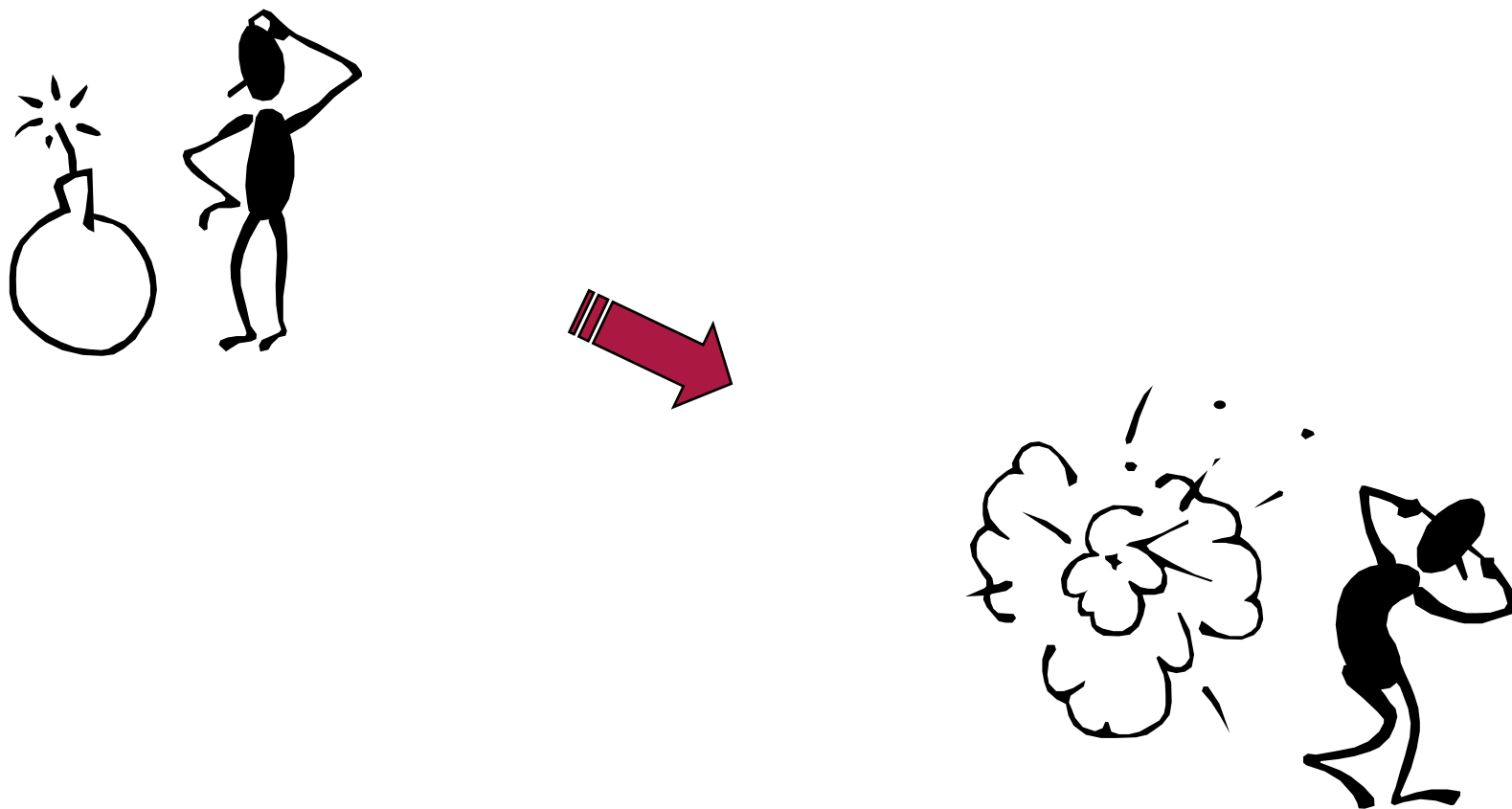
- Exemplo de criação de bordas de título

```
comp9.setBorder (BorderFactory.createTitledBorder("teste"));
```

```
...
```

```
Border linha = BorderFactory.createLineBorder (Color.black);  
TitledBorder titulo = BorderFactory.createTitledBorder (linha, "teste");  
titulo.setTitleJustification (TitledBorder.CENTER);  
titulo.setTitlePosition (TitledBorder. BOTTOM);  
comp10.setBorder (titulo);
```

Tratamento de Eventos



Eventos

- Controles geram eventos
 - Um evento indica a ocorrência de uma atuação do usuário sobre o controle ou alguma mudança interna que o afete
 - Exemplos: clique de um botão, movimento do mouse, alteração do conteúdo de uma linha de edição, ...
- Resposta a eventos
 - O desenvolvedor deve programar sua aplicação para responder aos eventos gerados sobre seus componentes
 - Qualquer objeto pode responder a eventos

Respondendo a Eventos

- Um objeto pode se registrar para “escutar” eventos
 - O objeto deve implementar uma interface específica
 - O objeto deve se registrar no componente
 - Diversos objetos podem ser registrados para o mesmo evento
 - Um mesmo objeto pode tratar diversos eventos distintos
- Cada evento é representado por um objeto
 - O objeto possui informações sobre o evento
 - O objeto identifica o componente gerador do evento

Principais Eventos e suas Interfaces

Ação do usuário	Interface
Clicar um botão	<i>ActionListener</i>
Pressionar Enter em uma linha de texto	<i>ActionListener</i>
Selecionar um item de menu	<i>ActionListener</i>
Fechar uma janela	<i>WindowListener</i>
Pressionar um botão do mouse	<i>MouseListener</i>
Mover o mouse	<i>MouseMotionListener</i>
Componente sendo apresentado	<i>ComponentListener</i>
Componente ganha o foco do teclado	<i>FocusListener</i>
Seleção de um item em uma lista	<i>ListSelectionListener</i>

Tratamento de Eventos

- Exemplo de trecho de código

```
...
JButton botao = new JButton ("Botao Qualquer");
botao.addActionListener (new Tratador ());
...

class Tratador implements ActionListener
{
    public void actionPerformed (ActionEvent e)
    {
        System.out.println("Houve um clique no botao");
    }
}
```

Eventos de Ação

- Interface *ActionListener*
 - Gerados por botões, menus e Enter em linhas de edição
 - Registrada através do método *addActionListener()*
 - A interface possui um único método, *actionPerformed()*
 - O método recebe um objeto da classe *ActionEvent*

Eventos de Teclado

- Interface *KeyListener*

- Evento de pressionamento e liberação de teclas
- Eventos são gerados pelo componente com o foco
- Registrada no componente pelo método *addKeyListener*
- Abaixo, são apresentados os métodos da interface

```
public void keyTyped (KeyEvent e);
```

```
public void keyPressed (KeyEvent e);
```

```
public void keyReleased (KeyEvent e);
```

Eventos de Teclado

- Classe *KeyEvent*
 - Informações adicionais dos eventos de teclado
 - O método *getKeyChar()* retorna o caractere da tecla associada com o evento
 - O método *getKeyCode()* retorna o código (Unicode) da tecla associada com o evento
 - A classe *KeyEvent* define diversas constantes de código referentes às teclas associadas com o evento
 - As constantes são iniciadas por VK (VK_A, VK_ESCAPE)

Eventos de Mouse

- Interface *MouseListener*
 - Eventos de clique e liberação dos botões do mouse
 - Eventos de entrada e saída do mouse de um componente
 - Registrada no componente através de *addMouseListener()*
 - Abaixo, são apresentados os métodos da interface

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

Eventos de Mouse

- Classe *MouseEvent*
 - Representa os dados complementares do evento de mouse
 - O método *getClickCount()* retorna o número de cliques consecutivos realizados pelo usuário
 - Os métodos *getX()*, *getY()* e *getPoint()* retornam a posição do mouse, relativa ao componente afetado pelo evento
 - O método *getComponent()* retorna o componente afetado pelo evento

Eventos de Mouse

- Interface *MouseMotionListener*
 - Eventos de movimento e arrastamento do mouse
 - Registrada através do método *addMouseMotionListener()*
 - Abaixo, são apresentados os métodos da interface
 - Os métodos utilizam objetos da classe *MouseEvent*

```
public void mouseMoved (MouseEvent e);  
public void mouseDragged (MouseEvent e);
```

Eventos de Seleção de Item

- Interface *ItemListener*
 - Utilizado por check boxes e combos boxes
 - Registrada através do método *addItemListener()*
 - A interface possui o método *itemStateChanged()*
 - O método recebe um objeto da classe *ItemEvent*
 - A classe *ItemEvent* possui diversos métodos
 - *getItem()* que retorna o item selecionado ou deselecionado
 - *getStateChange()* determina o tipo do evento, que pode ser *ItemEvent.SELECTED* ou *ItemEvent.DESELECTED*

Eventos de Janela

- Interface *WindowListener*
 - Eventos de abertura, fechamento, ativação, desativação, minimização e restauração de janelas ou diálogos
 - Recebem um objeto da classe *WindowEvent*, que possui o método *getWindow()*, que retorna a janela do evento

```
void windowOpened (WindowEvent);  
void windowClosing (WindowEvent);  
void windowClosed (WindowEvent);  
void windowIconified (WindowEvent);  
void windowDeiconified (WindowEvent);  
void windowActivated (WindowEvent);  
void windowDeactivated (WindowEvent);
```

Exercício

- Faça uma agenda telefônica que permita adicionar, remover e atualizar contatos usando os conceitos vistos em aula

The image shows a Java Swing window titled "Agenda". On the left side, there is a list of names: João, José, Maria, Paulo, and Pedro. The name "Maria" is currently selected and highlighted. To the right of this list, there are three text input fields. The first is labeled "Nome:" and contains the text "Maria". The second is labeled "Telefone:" and contains the text "(21) 98765-4321". The third is labeled "Detalhes:" and contains the text "Esposa do Pedro.". At the bottom of the window, there are three buttons: "Adiciona", "Remove", and "Atualiza".

Interface Gráfica Swing

Leonardo Gresta Paulino Murta
leomurta@ic.uff.br