Interface Gráfica e Banco de Dados em Java

Componentes GUI – Parte II

Licença para uso e distribuição

Este material está disponível para uso não-comercial e pode ser derivado e/ou distribuído, desde que utilizando uma licença equivalente.

Atribuição-Uso Não-Comercial-Compartilhamento pela mesma licença, versão 2.5

http://creativecommons.org/licenses/by-nc-sa/2.5/deed.pt

Você pode copiar, distribuir, exibir e executar a obra, além de criar obras derivadas, sob as seguintes condições: (a) você deve dar crédito ao autor original, da forma especificada pelo autor ou licenciante; (b) você não pode utilizar esta obra com finalidades comerciais; (c) Se você alterar, transformar, ou criar outra obra com base nesta, você somente poderá distribuir a obra resultante sob uma licença idêntica a esta.

Objetivos desta parte

- Apresentar componentes gráficos mais complexos, como:
  - Controles deslizantes e barras de progresso;
  - Menus, barras de ferramenta e menus *pop-up*;
  - Painel de abas, divisórias e bordas de painéis;
  - Áreas de edição de texto;
  - Janelas internas.
- Explicar como funciona o sistema de aparência e comportamento (*look & feel*) plugável;
- Entender o funcionamento do GridBagLayout.

JavaBeans

- Padrão definido pela Sun:
  - *Serializable* com construtor *default*;
  - Propriedades possuem `getXyz()` e `setXyz()` padronizados;
- Principal propósito: construtores de interface gráfica;
- A partir de agora, faremos referências a propriedades JavaBeans, e não a métodos de componentes.

JSlider: componente deslizante

- Determina um valor em um intervalo fechado;
- Instância de `javax.swing.JSlider`:
  - `majorTickSpacing`: espaçamento do traço maior;
  - `minorTickSpacing`: espaçamento do traço menor;
  - `minimum` e `maximum`: definem o intervalo;
  - `orientation`: `VERTICAL` ou `HORIZONTAL`;
  - `paintLabels`: mostra legenda;
  - `paintTicks`: mostra os traços;
  - `snapToTicks`: só pode escolher valores marcados por traços.

JSlider: detectando mudanças

- `ChangeListener` – método `stateChanged();`
- Propriedade `value` de `JSlider`: valor atual;
- Experimente:
  - Crie uma janela com dois painéis:
  - Painel central: construa uma classe que herda de `JPanel` e desenha um círculo, dado o diâmetro;
  - Painel no rodapé: um `JSlider` de 0 a 400, com marcas maiores a cada 100 e menores a cada 50;
  - Ao mudar o valor do *slider*, redesenhar o círculo com o novo diâmetro.

JSlider: legenda personalizada

- Podemos mudar a legenda do *slider*:
  - Crie uma `Hashtable` (similar a um `HashMap`);
  - Adicione pares número x componente. Ex.:

```
Hashtable ht = new Hashtable();

// Troca a legenda 100 por “Pequeno”:
ht.put(100, new JLabel("Pequeno"));
```

  - Adicione ao *slider* com `setLabelTable()`.

A tabela mapeia números a componentes GUI. Podemos então substituir os números por qualquer *widget*. Como poderíamos fazer para colocar imagens no rótulo do *slider*?

JFrame: mais sobre janelas

- Já vimos bastante sobre `JFrame`:
  - Criação, título e exibição;
  - Operação de fechamento e liberação de recursos;
  - Configurações diversas: *resizable*, *alwaysOnTop*, etc.
- Para encerrar o assunto, discutiremos:
  - Eventos de janelas;
  - O painel de conteúdo;
  - Menus e barras de ferramentas.

JFrame: eventos de janela

- Três ouvintes específicos de janela:
- `WindowListener`:
  - `windowActivated();`
  - `windowClosed();`
  - `windowClosing();`
  - `windowDeactivated();`
  - `windowDeiconified();`
  - `windowIconified();`
  - `windowOpened()`.

JFrame: eventos de janela

- `WindowFocusListener`:
  - `windowGainedFocus();`
  - `windowLostFocus()`.
- `WindowStateListener`:
  - `windowStateChanged()`.
- Todos os métodos recebem uma instância de `WindowEvent`;
- O adaptador `WindowAdapter` implementa todas estas interfaces.

JFrame: eventos de janela

- Teste os eventos de janela!
- Crie uma janela que contenha um rótulo que diga:
  - “Bom dia!” quando a janela é aberta;
  - “Olá de novo!” quando a janela for restaurada;
  - “ZZZ...” quando a janela estiver inativa;
  - “Opa!” quando a janela for reativada.
- Além disso, use `JOptionPane` para dizer:
  - “Até logo!” quando a janela for minimizada;
  - “Adeus!” quando a janela for fechada.

JFrame: o painel de conteúdo

- JFrames possuem a propriedade contentPane: painel exibido entre as bordas da janela;
- Até agora, criamos subclasses de JFrame;
- A partir de agora, criaremos classes que estendem JPanel e as colocaremos como painel de conteúdo de um JFrame;
  - Justificativa: nosso painel pode ser usado em outros tipos de janela (ex.: janelas internas);
- Quando o JFrame precisar de configurações extensas (ex.: janela com menus), usaremos as duas abordagens.

JFrame: o painel de conteúdo

```
public class PanelJSlider extends JPanel {

   /* ... */

   public static void main(String args[]) {
      JFrame janela = new JFrame("Teste");
      janela.setSize(600, 600);
      janela.setDefaultCloseOperation(
        JFrame.DISPOSE_ON_CLOSE);
      janela.setContentPane(
        new PanelJSlider());
      janela.setVisible(true);
   }
}
```

JMenuBar: adicionando menus

- Várias classes compõem um menu:
  - `JMenuBar`: a barra de menus;
  - `JMenu`: um menu;
  - `JMenuItem`: um item dentro de um menu;
    - `JCheckBoxMenuItem`: item com *checkbox*;
    - `JRadioButtonMenuItem`: item com *radio button*;
  - `JSeparator`: um separador de menus / itens.
- Todas do pacote `javax.swing`.

JMenuBar: propriedades

- `JMenu`, `JMenuItem` e subclasses:
  - `text`: texto apresentado pelo menu;
  - `mnemonic`: letra que ativa o menu pelo teclado;
  - `icon`: ícone do menu;
  - `toolTipText`: dica de ferramenta (aparece quando o *mouse* paira sobre o menu / item).
- Adiciona-se itens e separadores à `JMenus` pelo método `add();`
  - A ordem que são adicionados indica a ordem que aparecerão.

Dicas de ferramentas

- Vimos que menus possuem uma propriedade `toolTipText`;
- *Tool tip* é um texto que aparece quando o *mouse* paira sobre o componente;
- Todos os componentes possuem esta mesma propriedade, que pode ser configurada.

JPopupMenu: menu *pop-up*

- Um menu *pop-up* aparece quando clicamos com o botão direito em um componente;
- Funciona como um menu normal;
- Deve ser exibido usando o método `show()` – parâmetros:
  - Componente que ativou o *pop-up*;
  - Posição X e Y onde o menu deve aparecer (relativo ao componente que ativou).

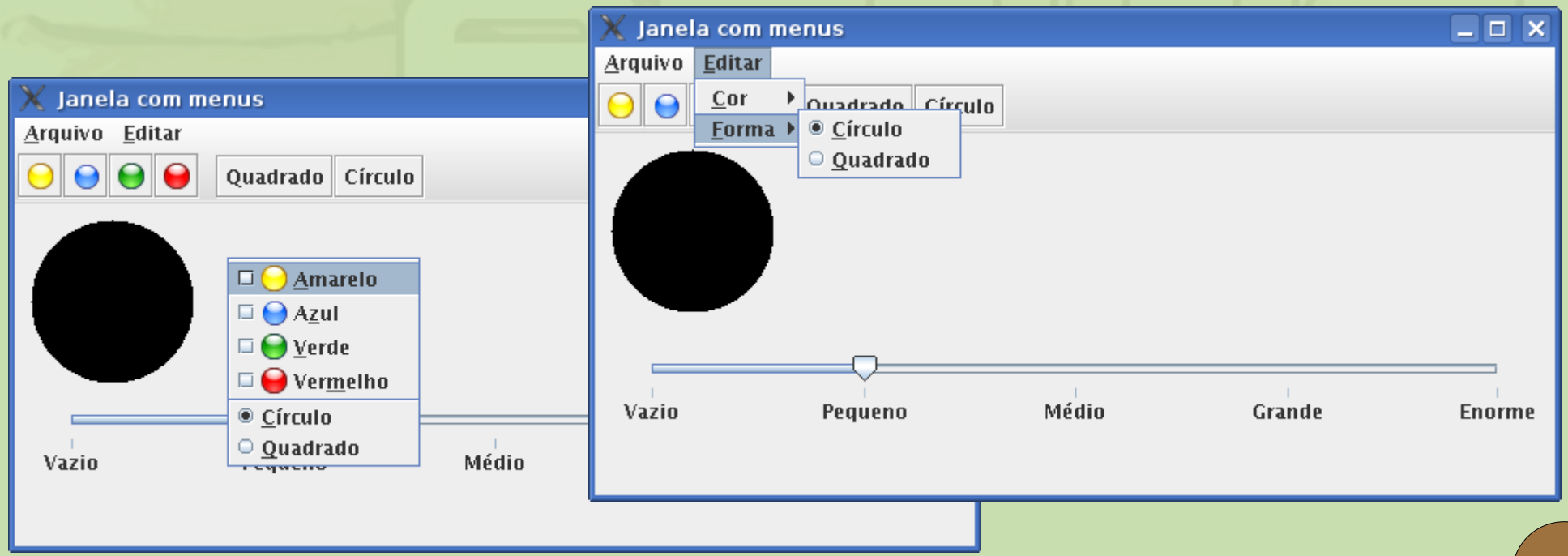
ATENÇÃO: menus não podem compartilhar itens (adicionar o mesmo item a dois menus).

JToolBar: barra de ferramentas

- Instância de `javax.swing.JToolBar`:
  - `floatable`: se a barra pode ou não trocar de lugar e ser colocada como flutuante;
  - Aceita qualquer *widget*;
  - Para adicionar separadores, use `JToolBar.Separator` ao invés de `JSeparator`.

Menus e barras de ferramentas

- Experimente:
  - Crie uma janela e coloque o painel do `JSlider` criado anteriormente;
  - Adicione menus (normal e *pop-up*) e uma barra de ferramentas: menus básicos, troca de cor e de forma.



Look & feel

- O *look & feel* (aparência & comportamento, L&F ou LAF) define como serão exibidos os *widgets*;
- Componentes Swing possuem LAF plugável, isto é, pode ser alterado em tempo de execução;
- LAFs do Java SE:

L&F	Classe	SO
Metal	**javax.swing.plaf.metal.MetalLookAndFeel**	Todos
CDE/Motif	**com.sun.java.swing.plaf.motif.MotifLookAndFeel**	Todos
GTK+	**com.sun.java.swing.plaf.gtk.GTKLookAndFeel**	Linux
Windows	**com.sun.java.swing.plaf.windows.WindowsLookAndFeel**	Windows

- Outros podem ser baixados da Internet.

Mudando o LAF

```
// import javax.swing.SwingUtilities;
// import javax.swing.UIManager;

// Obtém os LAFs instalados:
UIManager.LookAndFeelInfo[] lafs;
lafs = UIManager.getInstalledLookAndFeels();

// Obtém informações sobre o segundo LAF:
String classe = lafs[1].getClassName();
String nome = lafs[1].getName();

// Altera para este LAF:
UIManager.setLookAndFeel(classe);
SwingUtilities.updateComponentTreeUI(this);
```

Janelas MDI

- MDI = *Multiple Document Interface* = Interface de Múltiplos Documentos;
- Classes `JDesktopPane` (*container*) e `JInternalFrame` (janela) – `javax.swing`;
- Permitem a criação de janelas internas:
  - Não aparecem na barra de tarefas;
  - Podem ser maximizadas ou minimizadas dentro da área da janela MDI.

JDesktopPane & JInternalFrame

- `JDesktopPane`:
  - Gerencia as janelas internas;
  - Pode ser colocado no painel de conteúdo da janela ou ser o próprio painel de conteúdo;
  - Método `add()` adiciona janelas internas.
- `JInternalFrame`:
  - Muito similar ao `JFrame`;
  - Logo no construtor, pode ser especificado: título, se pode redimensionar, se pode fechar, se pode maximizar e se pode minimizar;
  - `setFrameIcon()`: atribui um ícone à janela.

JDesktopPane & JInternalFrame

- Experimente:
  - Crie um `JFrame` que tenha um painel para disposição de janelas internas;
  - Adicione um menu “Nova janela” que cria e exibe uma nova janela interna;
  - Coloque o painel do teste do `JSlider` nas janelas criadas;
  - Coloque um ícone na janela.

JInternalFrame: reposicionando

- Por padrão, janelas internas abrem todas na posição (0, 0);
- Podemos mudar sua posição com o método `reshape()` – parâmetros:
  - Coordenada X;
  - Coordenada Y;
  - Largura;
  - Altura.

JInternalFrame: selecionando

- Algumas vezes a nova janela interna não é selecionada e trazida para frente por padrão;
- Você pode forçar esta seleção com `setSelected(true);`
- Este método pode lançar uma `PropertyVetoException`, caso a atual janela selecionada se recuse a ceder a vez.

JDesktopPane: manipulando janelas

- Em algumas situações, você pode querer manipular todas as janelas internas;
  - Ex.: colocá-las lado a lado ou em cascata.
- Para obtê-las, use o método `getAllFrames()` de `JDesktopPane`;
- Experimente:
  - Adicione um menu “Janela” e um item “Em cascata” e disponha todas as janelas internas em cascata.
- Você precisará saber o estado de uma janela: use `isIcon()` e `setMaximum()`.

JInternalFrame: eventos

- Janelas internas podem ser monitoradas por `InternalFrameListener` – igual a `JFrames`;
- Alternativamente, podemos utilizar um `VetoableChangeListener`:
  - `PropertyChangeEvent` como parâmetro;
  - Propriedades `propertyName` e `newValue` indicam o que está sendo solicitado;
  - Ex.: `propertyName` = “closed”, `newValue` = TRUE.
  - Se o ouvinte lançar uma exceção, a propriedade não muda de valor (é realizado um veto).
- Pode ser feito com `internalFrameClosing()`.

JOptionPane para janelas internas

- JOptionPane possui versões para janelas internas:
  - `showInternalConfirmDialog();`
  - `showInternalInputDialog();`
  - `showInternalMessageDialog();`
  - `showInternalOptionDialog()`.
- Justificativa: diálogos internos gastam menos recursos do SO do que diálogos normais.

JProgressBar: medindo progresso

- Barras de progresso medem o andamento de uma tarefa específica;
- Instância de `javax.swing.JProgressBar`:
  - `minimum`: valor mínimo;
  - `maximum`: valor máximo;
  - `value`: valor atual da barra;

  } Igual ao `JSlider`!
  - `stringPainted`: se o percentual é impresso em cima da barra de progresso;
  - `orientation`: `VERTICAL` ou `HORIZONTAL`;
  - `string`: o que está impresso em cima da barra.

Threads 101

- Barras de progresso monitoram atividades que ocupam o processador;
- A atualização da barra de progresso também ocupa o processador;
- Precisamos processar estas duas atividades em paralelo. Para isso utilizamos *threads*;
- *Threads* ou “linhas de execução” são implementadas por `java.lang.Thread`.

Threads 101

```
public class AtividadeSimulada extends Thread {
  private int minimo;
  private int maximo;
  private int valor;

  public AtividadeSimulada(int minimo, int maximo) {
    this.minimo = minimo;
    this.maximo = maximo;
  }

  public void run() {
    valor = minimo;
    while (valor++ < maximo) {
      try { sleep(100); }
      catch (InterruptedException e) { return; }
    }
  }
```

Threads 101

```
   public int getValor() {
       return valor;
   }

   public static void main(String[] args) {
     AtividadeSimulada ativ;
     ativ = new AtividadeSimulada(0, 10);
     ativ.start();
   }
}
```


- O método a sobrescrever é `run();`
- O método a chamar é `start();`
- Para interromper: `interrupt();`
- Para ver se está ativa: `isAlive()`.

`Timer`: monitoramento periódico

- Temos a barra de progresso, temos a atividade paralela. Falta o elo entre os dois: um *timer*;
- Recebe um intervalo e um `ActionListener`, ativando o evento de tempos em tempos;
- Criaremos um *timer* para monitorar a atividade e atualizar a barra de progresso;
- Instância de `javax.swing.Timer`.

JProgressBar com *timer* e *thread*

```
// A atividade ainda não foi iniciada. Inicia.
if (atividade == null) {
  atividade = new AtividadeSimulada(0, 400);
  barraProgresso.setValue(atividade.getValor());

  // O timer monitorará a atividade a cada 0,5 seg.
  timer = new Timer(500, new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      // Verifica se já acabou ou foi interrompida.
      if ((atividade == null) ||
                              (! atividade.isAlive())) {
        timer.stop();
        atividade = null;
        botaoIniciar.setText("Iniciar Simulação");
      }
      else {
```

JProgressBar com *timer* e *thread*

```
            barraProgresso.setValue(atividade.getValor());
            areaTexto.append(atividade.getValor() + "\n");
        }
      }
    });

    atividade.start();
    timer.start();
    botaoIniciar.setText("Parar Simulação");
}

// A atividade já foi iniciada. Interrompe.
else {
    atividade.interrupt();
    atividade = null;
    botaoIniciar.setText("Iniciar Simulação");
}
```

Organizando o visual

- Algumas ferramentas que temos para organizar o visual de nossas janelas e painéis:
  - Painéis com bordas;
  - Divisórias;
  - Painéis com guias.

Bordas dos painéis

- Painéis podem ser usados para organizar o *layout* – isto nós já vimos;
- Podemos ir além e decorá-los com bordas:
  - Classe `javax.swing.BorderFactory` cria bordas;
  - Pacote `javax.swing.border` contém bordas.

JSplitPane: divisórias

- Divide um painel/janela em duas áreas;
- Instância de `javax.swing.JSplitPane`:
  - `dividerLocation`: distância da margem;
  - `dividerSize`: tamanho;
  - `orientation`: `HORIZONTAL_SPLIT` ou `VERTICAL_SPLIT`;
  - `continuousLayout`: atualizações contínuas;
  - `oneTouchExpandable`: presença de botões de expansão;
  - `resizeWeight`: como redimensionar os painéis quando o painel de divisória é redimensionado.

JSplitPane: experimente...

- Crie um painel com uma divisória no meio;
- À esquerda, coloque o painel do `JSlider`;
- À direita, coloque um texto de ajuda:

Manipule o componente deslizante na parte inferior da janela para mudar o tamanho da figura desenhada acima dele.

  - Use uma `JTextArea`, mudando sua cor de fundo para `Panel.background`;
  - Coloque a área de texto em um `JScrollPane`, configurando sua borda como “(No border)”.

JTabbedPane: múltiplas guias

- Mostra guias (abas, *tabs*) que permitem múltiplos painéis na mesma janela;
- Instância de `javax.swing.JTabbedPane`:
  - `addTab()` adiciona uma guia:
    - Título, ícone, componente, tooltip.
  - `insertTab()` insere em uma dada posição:
    - Título, ícone, componente, tooltip, índice.
  - `removeTabAt(índice)` remove uma guia;
  - `setSelectedIndex(índice)` seleciona uma guia;
  - `tabLayoutPolicy`: *wrap* ou *scroll*;
  - `tabPlacement`: `TOP`, `LEFT`, `RIGHT`, `BOTTOM`.

JTabbedPane: detectando mudanças

- `ChangeListener` – método `stateChanged();`
- Propriedades `selectedIndex` e `tabCount`;
- Experimente:
  - Crie um painel com 6 guias, colocando em cada uma uma imagem diferente;
  - Experimente diferentes políticas de *layout* e posicionamentos;
  - Adicione um ouvinte que exiba a mensagem “Esta é a última imagem” sempre que a última imagem for exibida.

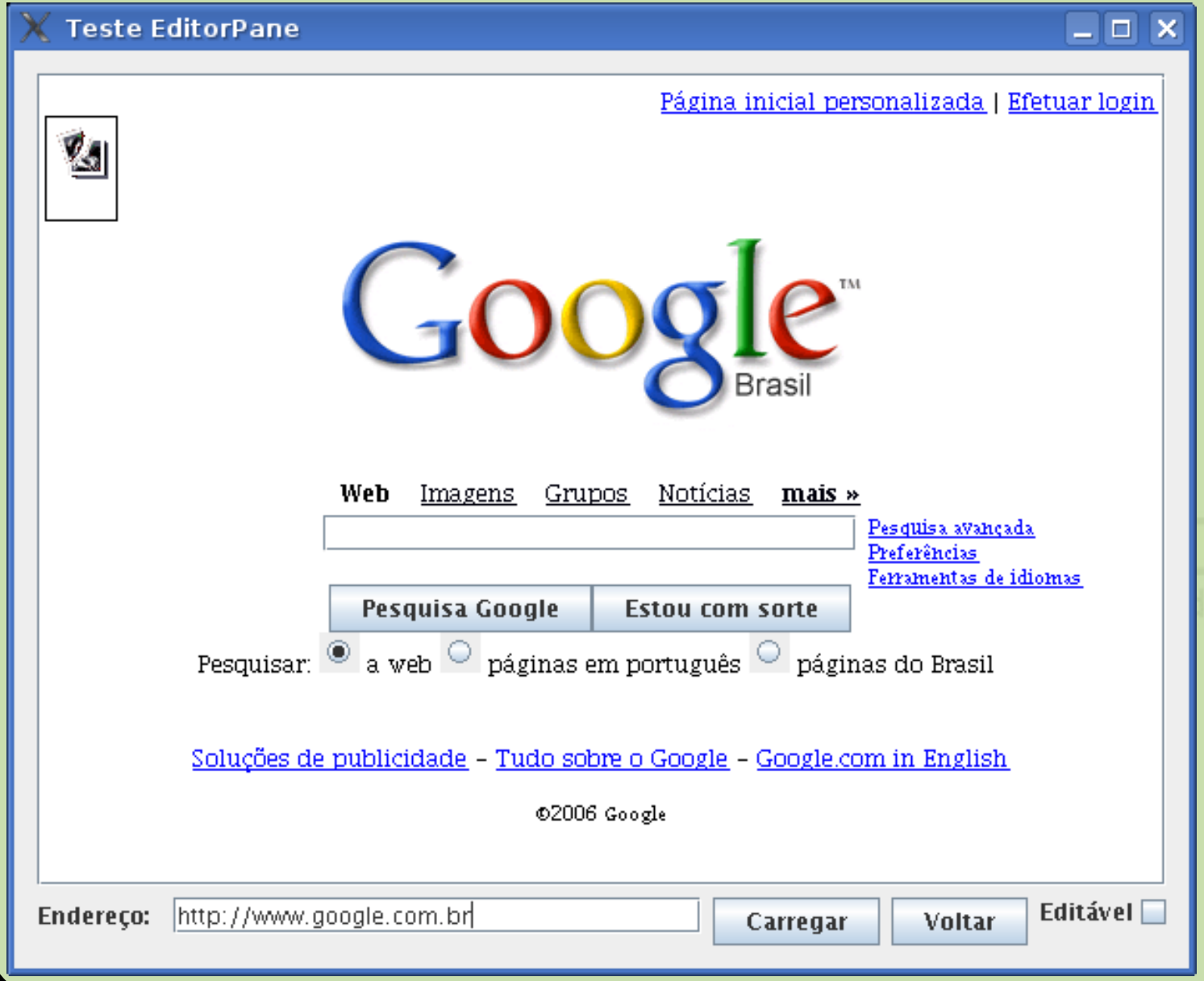
JEditorPane: texto “rico”

- O painel de edição permite exibir *rich text* em RTF ou HTML;
  - A exibição de texto é razoável.
- Instância de `javax.swing.JEditorPane`:
  - `editable`: se é editável;
  - `text`: conteúdo (texto);
  - `setPage(url)`: carrega uma página dada a URL (pode lançar `java.io.IOException`).
- Assim como `JTextArea`, deve ser usado dentro de um `JScrollPane`.

JEditorPane: detectando hiperlinks

- `HyperlinkListener` – método `hyperlinkUpdate();`
- Recebe uma instância de `HyperlinkEvent`, a qual fornece um método `getEventType()`:
  - `HyperlinkEvent.EventType.ACTIVATED;`
  - `HyperlinkEvent.EventType.ENDERED;`
  - `HyperlinkEvent.EventType.EXITED.`

JEditorPane: experimente!



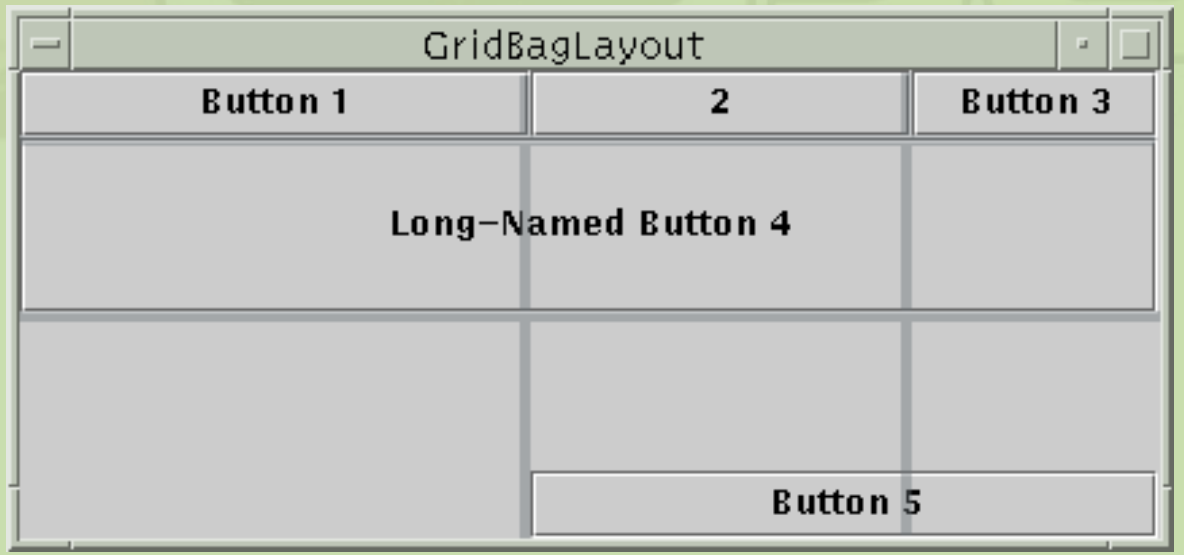
Construa um navegador da Internet!

Relembrando GridLayout

- Divide o painel em células, como uma tabela;
- Cada célula contém até um componente;
- Construtor recebe número de linhas e colunas e, opcionalmente, espaçamentos vert. e horiz.;
- Componentes são dispostos na seqüência de leitura (ex.: [1, 1], [1, 2], [2, 1], [2, 2], ...).

Relembrando GridBagLayout

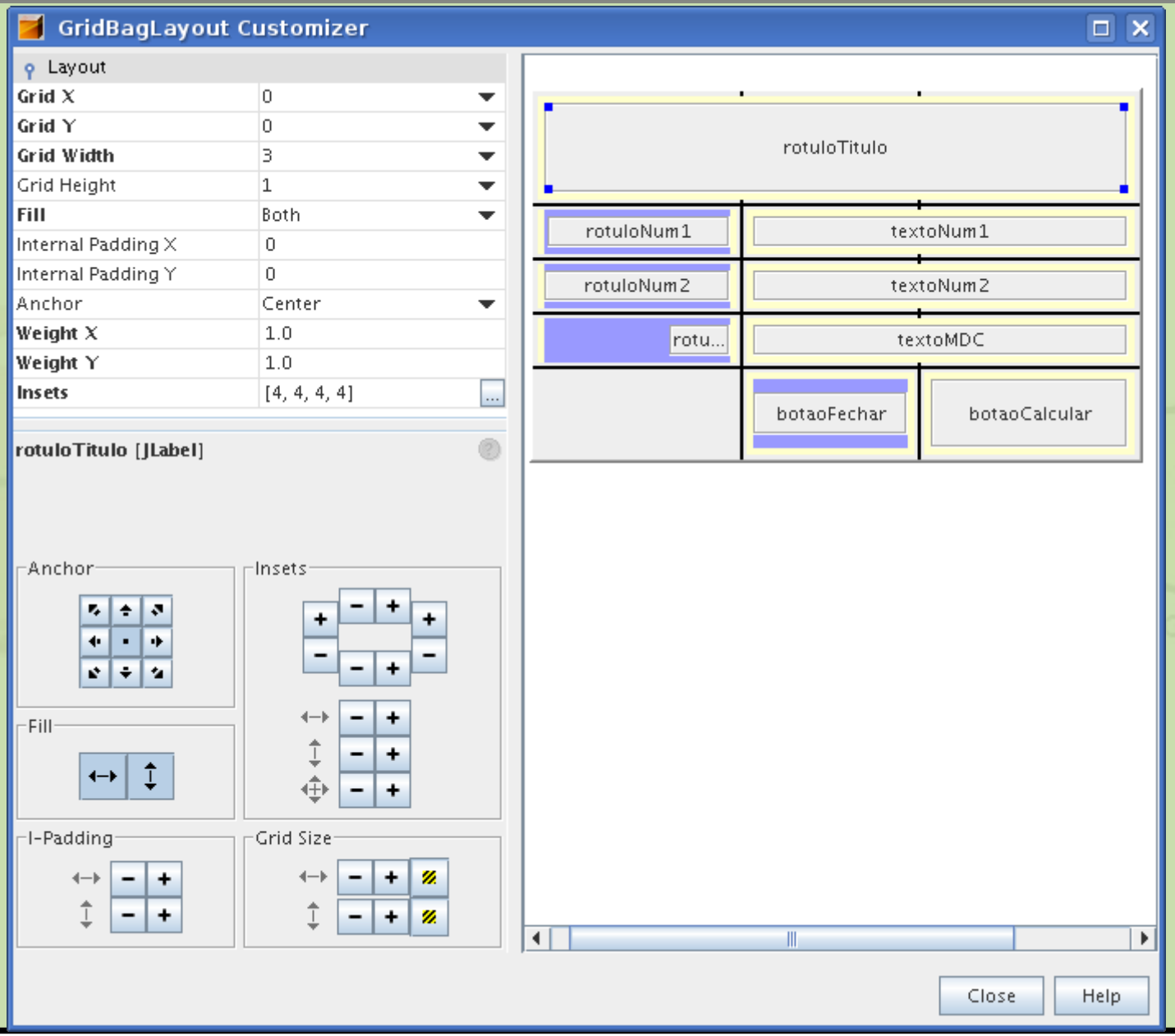
- Similar ao `GridLayout`, porém:
  - Colunas e linhas podem ter tamanhos diferentes;
  - Componentes podem ocupar mais de uma célula;
  - Uso de `GridBagConstraints` para configuração;
  - Muito difícil de usar manualmente, requer IDE.



Entendendo o GridBagLayout

- O que determina a posição de um objeto são restrições impostas ao *GridBag*;
  - Instância de `java.awt.GridBagConstraints`;
  - Passada como 2º parâmetro do método `add()`.
- Propriedades:
  - `gridx` e `gridy`: linha e coluna para posicionamento;
  - `gridwidth` e `gridheight`: largura e altura (em número de células ocupadas no *grid*);
  - `weightx` e `weighty`: peso na distribuição de espaço;
  - `anchor`: posição do componente no espaço do *grid*;
  - `fill`: se o componente deve ocupar todo o espaço.

Usando uma IDE



Conclusões

- Vimos nesta parte do curso:
  - Controles deslizantes e barras de progresso;
  - Menus, barras de ferramenta e menus *pop-up*;
  - Painel de abas, divisórias e bordas de painéis;
  - Áreas de edição de texto;
  - Janelas internas;
  - Sistema de aparência e comportamento plugável;
  - O gerenciador de *layout* `GridBagLayout`.