Interface Gráfica e Banco de Dados em Java

Componentes GUI – Parte I

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

- Introduzir conceitos básicos sobre criação de GUIs em Java usando a API Swing;
- Apresentar componentes gráficos simples como rótulos, botões, listas e painéis;
- Mostrar como tratar eventos diversos nos componentes apresentados;
- Explicar sobre gerenciadores de *layout* e seu papel na disposição dos componentes.

Componentes GUI

- GUI = *Graphical User Interface*:
  - Janela (ou similar) que se apresenta ao usuário, permitindo que use o programa.
- Componentes GUI / *Widgets* (*Window Gadgets*);
  - Objetos individuais com os quais o usuário interage;
  - Também chamado de “controles”;
  - Compõem as GUIs.

Aprender sobre o funcionamento dos *widgets*

=

Aprender a construir interfaces gráficas

Referências

- API do Java SE:
  - http://java.sun.com/j2se/1.5.0/docs/api/index.html
- The Java Tutorial – Swing Trail:
  - http://java.sun.com/docs/books/tutorial/ui/index.html
- Bibliografia deste curso e slides.

Aprendendo na prática

- Temos a funcionalidade:

```
public class Calculadora {
   public int mdc(int x, int y) {
      while (x != y) {
         if (x > y) x = x – y;
         else {
            int z = x;
            x = y;
            y = z;
         }
      }
      return x;
   }
}
```

Aprendendo na prática

- Precisamos da interface com o usuário:

```
public class MainConsole {
   public static void main(String[] args) {
      Calculadora calc = new Calculadora();
      Scanner sc = new Scanner(System.in);

      System.out.println("Informe 2 números:");
      int x = sc.nextInt();
      int y = sc.nextInt();

      int mdc = calc.mdc(x, y);
      System.out.println("MDC = " + mdc);
   }
}
```

JOptionPane

- Interface mais simples;
- Utilização de caixas de diálogo;
- Métodos estáticos da classe `javax.swing.JOptionPane`;
  - `showConfirmDialog()`: sim, não, cancela, etc.;
  - `showInputDialog()`: entrada de dados;
  - `showMessageDialog()`: saída de dados;
  - `showOptionDialog()`: opções personalizadas;
  - Versões para "*internal frame*" (`showInternalInputDialog()`, etc.).

JOptionPane: obtendo dados

- Use `showInputDialog()` (`static`);
- Parâmetros possíveis:
  - O componente pai (`Component`);
  - A mensagem (`Object`);
  - O título (`String`);
  - O tipo de mensagem (uma das constantes);
  - Um ícone (`Icon`);
  - Valores possíveis (`Object[]`);
  - Valor inicial (`Object`).
- Retorno: `String`.

JOptionPane: tipos de mensagens

- Uma das constantes (`int`) definidas:
  - `ERROR_MESSAGE`: erro;
  - `INFORMATION_MESSAGE`: informação;
  - `WARNING_MESSAGE`: aviso;
  - `QUESTION_MESSAGE`: pergunta;
  - `PLAIN_MESSAGE`: mensagem simples.

JOptionPane: exibindo respostas

- Use `showMessageDialog()` (`static`);
- Parâmetros possíveis:
  - O componente pai (`Component`);
  - A mensagem (`Object`);
  - O título (`String`);
  - O tipo de mensagem (uma das constantes);
  - Um ícone (`Icon`).
- Retorno: `void`.

Incrementando o exemplo

- Vamos perguntar ao usuário se deseja calcular mais algum MDC;
- Algoritmo:

Enquanto a resposta for “sim”:

- ✔ Leia o primeiro número;
- ✔ Leia o segundo número;
- ✔ Calcule o MDC;
- ✔ Escreva o MDC;
- ✔ Pergunte se quer calcular mais algum número;
- ✔ Leia a resposta.

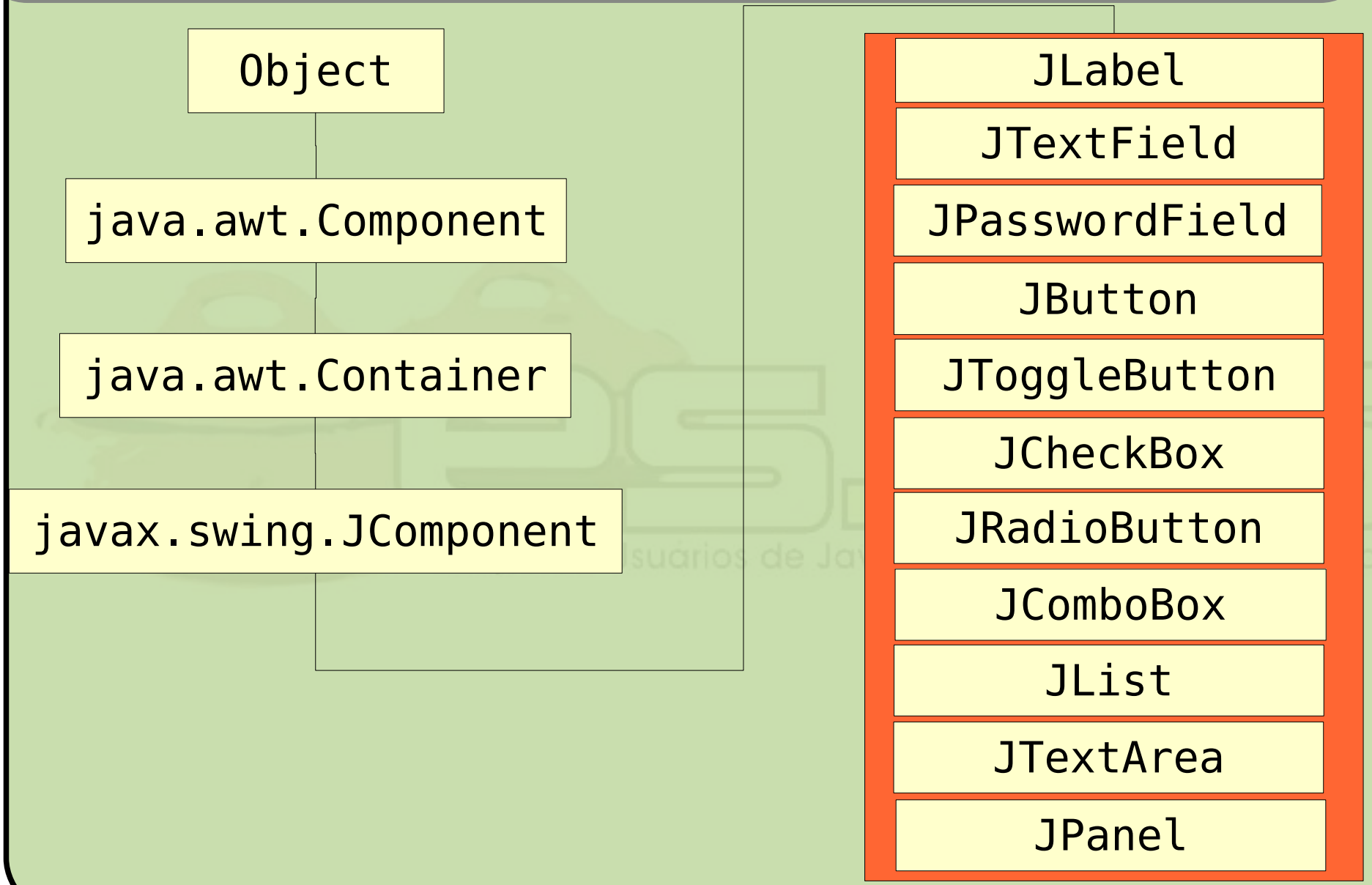
JOptionPane: fazendo perguntas

- Use `showConfirmDialog()` (`static`);
- Parâmetros possíveis:
  - O componente pai (`Component`);
  - A mensagem (`Object`);
  - O título (`String`);
  - O tipo de confirmação (uma das constantes);
  - O tipo de mensagem (uma das constantes);
  - Um ícone (`Icon`).
- Retorno: uma das constantes (`int`).

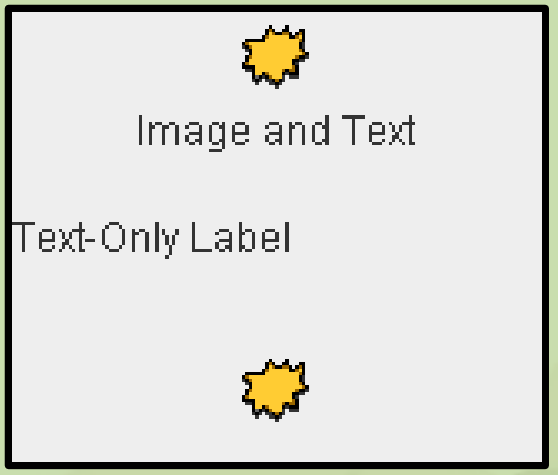
JOptionPane: mais constantes

- Constantes (`int`) de tipo de confirmação:
  - `DEFAULT_OPTION`: depende do SO;
  - `YES_NO_OPTION`: sim ou não;
  - `YES_NO_CANCEL_OPTION`: sim, não ou cancela;
  - `OK_CANCEL_OPTION`: OK ou cancela.
- Constantes (`int`) de resposta:
  - `YES_OPTION`: respondeu sim;
  - `NO_OPTION`: respondeu não;
  - `CANCEL_OPTION`: clicou em cancelar;
  - `OK_OPTION`: clicou em OK;
  - `CLOSED_OPTION`: fechou a janela.

Componentes Swing mais simples



Componentes Swing mais simples



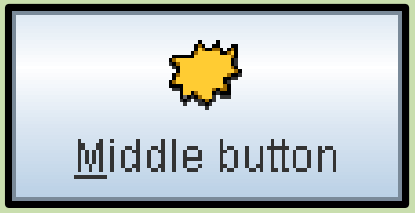
Rótulos exibem texto e/ou ícones e não podem ser editados.

`JLabel`

Campos de texto permitem que o usuário digite informações a serem recuperadas pelo programa. Podem também mostrar texto não-editável. Podem ser simples, de senha ou de múltiplas linhas.

`JTextField`
`JPasswordField`
`JTextArea`

Componentes Swing mais simples

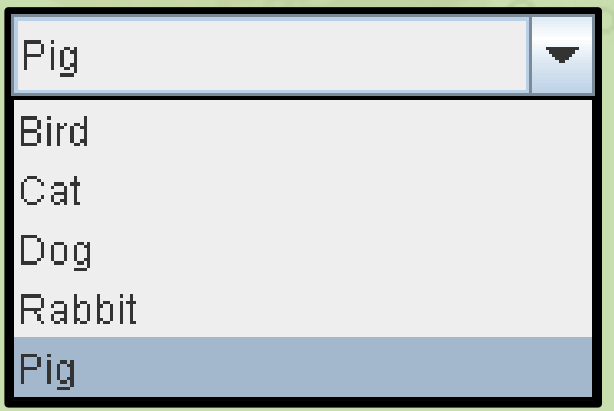


Botões podem receber estímulos do usuário (cliques) e ativar eventos.

`JButton / JToggleButton`

Caixas de marcação e botões de rádio especificam opções que podem ou não ser selecionadas (marcadas).

`JCheckBox / JRadioButton`



Caixas de combinação oferecem opções e permitem que o usuário selecione uma ou digite um valor qualquer.

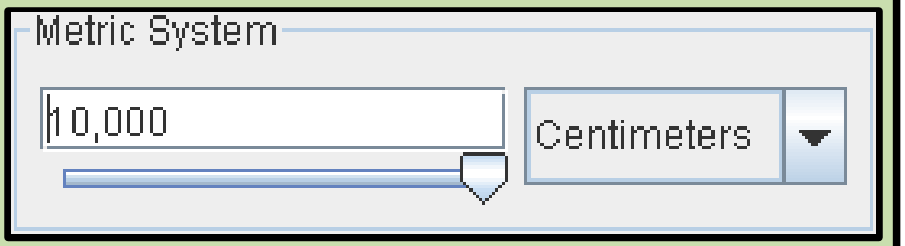
`JComboBox`

Componentes Swing mais simples

JList

Listas exibem opções e permitem que sejam selecionados vários itens.

Painéis fornecem uma área na qual componentes podem ser organizados. Também podem ser usados como área para desenho.



JPanel

Características comuns

- *Look & feel* plugável;
- Teclas de atalho;
- Tratamento de eventos;
- Dicas em balões (*tool tips*);
- Acessibilidade;
- Internacionalização e localização.

Janelas

- Componentes não flutuam na tela, são organizados dentro de janelas;
- O sistema operacional fornece a borda e a barra de título;
- Java oferece a parte interna, onde podemos dispor nossos componentes.

JFrame: minha primeira janela

- Crie uma instância de `javax.swing.JFrame`:
  - Você pode passar o título da janela no construtor.
- Use suas funções para configurá-la:
  - `setTitle(titulo)`: altera o título da janela;
  - `setDefaultCloseOperation(constante)`: indica o que fazer ao fechar a janela;
  - `setSize(largura, altura)`: redimensiona a janela (todas as medidas são em pixels);
  - `setVisible(true/false)`: exibe ou esconde a janela.

JFrame: operações de fechamento

- Uma das constantes (`int`) definidas:
  - `EXIT_ON_CLOSE`: termine o programa;
  - `DISPOSE_ON_CLOSE`: libera os recursos nativos utilizados pela janela, inutilizando-a. Se só há uma janela ativa, *dispose* encerra a aplicação;
  - `DO_NOTHING_ON_CLOSE`: não faça nada;
  - `HIDE_ON_CLOSE`: somente esconda a janela.
- A primeira constante é definida em `JFrame` e as demais em `javax.swing.WindowContants` (e herdadas por `JFrame`).

JFrame: mostrando algo na janela

- Janelas `JFrame` possuem um painel interno, onde podemos adicionar *widgets*;
- Podemos adicionar componentes GUI na janela usando o método `add(Component);`
- Teste-o, adicionando um `javax.swing.JLabel` criado pela expressão:

```
new JLabel("Hello World!").
```

JFrame como superclasse

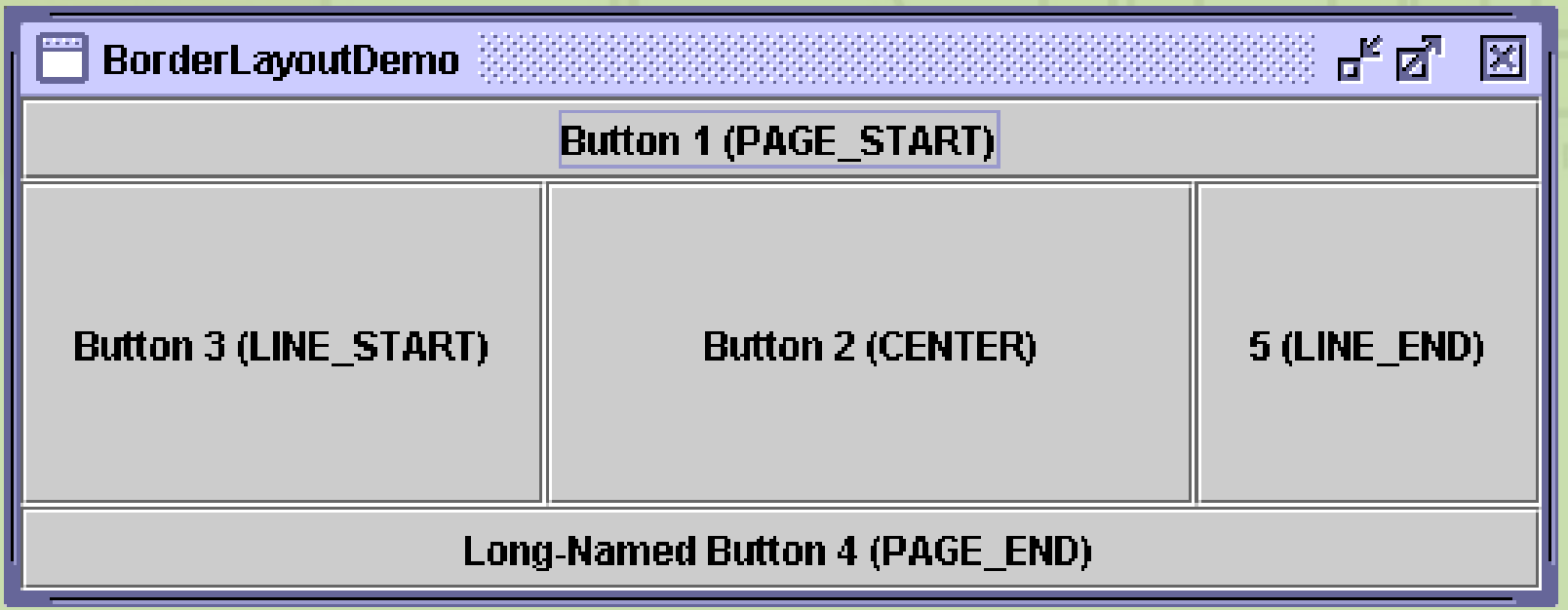
- Forma mais organizada e usada pela grande maioria das pessoas;
- Ao invés de criar toda a janela no método `main()`, criaremos uma nova classe;
- Esta classe deve herdar de `JFrame`;
- Crie um método privado `initComponents();`
- Chame este método em todos os construtores;
- Coloque o código relacionado à adição de *widgets* no método `initComponents()`.

Gerenciadores de *layout*

- Componentes em painéis podem ser gerenciados por um `java.awt.LayoutManager`;
- Se não for usado nenhum gerenciador, devem ser especificadas coordenadas (x, y);
- Vantagens de usá-los:
  - Exibição coerente independente da plataforma;
  - Redimensionamento da janela também expande/contrai os componentes GUI.
- O gerenciador de *layout* padrão é o `java.awt.BorderLayout`.

BorderLayout: centro e bordas

- Permite dispor componentes em cinco posições (constantes `int` da classe `BorderLayout`):
  - PAGE_START, PAGE_END: topo e base;
  - LINE_START, LINE_END: esquerda e direita;
  - CENTER: meio.



FlowLayout: fluxo de *widgets*

- Dispõe os componentes em seqüência;
- Quando acaba o espaço de uma linha, segue para a próxima.

Experimentando *layout managers*

- Experimente nenhum gerenciador (nulo), `FlowLayout` e `BorderLayout`:
  - Método `setLayout()` de `JFrame`, parâmetros `null`, `new FlowLayout()` e `new BorderLayout()`;
  - `BorderLayout` é o *default* e para usá-lo você pode simplesmente não especificar nenhum.
- Para configurar a posição do rótulo no `BorderLayout`, use o próprio método `add()`:

```
JLabel rotuloTitulo = new Jlabel("Título");
add(rotuloTitulo, BorderLayout.PAGE_START);
```

Gerenciadores de *layout*

- Para configurar a posição no *layout* `null`, use o método `setBounds(x, y, larg, alt)` da classe `JLabel` (herdado de `Component`):

```
JLabel rotuloTitulo = new Jlabel("Título");
add(rotuloTitulo);

// Mostra na posição (100, 50) com o
// tamanho 200 x 50
rotuloTitulo.setBounds(100, 50, 200, 50);
```

JLabel: trabalhando com rótulos

- Rótulos mostram textos e ícones;
- Já usamos um rótulo simples, vamos agora mostrar um com ícone:
  - Usaremos a interface `javax.swing.Icon` e a classe `javax.swing.ImageIcon`;
  - Obteremos a imagem no caminho de classes.

```
Icon iconeJar = new ImageIcon(getClass().get
Resource("/resources/images/iconJar.png"));

JLabel rotuloTitulo = new JLabel("Calculadora
Java", iconeJar, SwingConstants.LEFT);
```

JLabel: posição do texto e do ícone

- Você pode alterar a posição do texto em relação ao ícone:
  - `setHorizontalTextPosition(constante)`: `LEFT`, `CENTER` ou `RIGHT`;
  - `setVerticalTextPosition(constante)`: `TOP`, `CENTER` ou `BOTTOM`;
  - Constantes de `JLabel`, herdadas de `SwingConstants`.
- Configure o ícone para ficar acima do texto, ambos centralizados (ou seja, o texto deve estar centralizado e abaixo).

JLabel: novos rótulos, redimensionar

- Adicione dois novos rótulos para os campos da calculadora: “Primeiro” e “Segundo número”;
- Os rótulos são exibidos um do lado do outro. Como fazer para que fiquem na próxima linha?
- Redimensione os rótulos:
  - `setMinimumSize(dimensão)`: o mínimo;
  - `setPreferredSize(dimensão)`: o preferido;
  - `setMaximumSize(dimensão)`: o máximo;
  - dimensão é da classe `java.awt.Dimension` e recebe a largura e a altura em seu construtor.

JFrame: e se redimensionar?

- O que acontece se redimensionarmos a janela agora? Graças ao `FormLayout`, bagunça tudo.
- Podemos impedir o redimensionamento e ainda configurar outras características da janela:
  - `setResizable(true/false)`: se pode redimensionar (*default* é `true`);
  - `setUndecorated(true/false)`: se é apresentada sem barra e borda (*default* é `false`);
  - `setAlwaysOnTop(true/false)`: se fica por cima de todas as outras janelas (*default* é `false`);
  - `setBounds(x, y, largura, altura)`: posição na tela e tamanho.

JTextField: entrada de dados

- Campos de texto permitem entrada de dados;
- Crie instância de `javax.swing.JTextField`:
  - Pode especificar texto, número de colunas ou ambos no construtor.
- Crie três campos de texto e disponha-os ao lado de seus respectivos rótulos;
  - Para isso adicione-os à janela logo após seu rótulo e ajuste o tamanho de ambos;
  - Dica: alinhe os rótulos e os campos de texto à direita com `setHorizontalAlignment()` e não use uma altura muito maior que 20 pixels para os campos.

JTextField: características

- Não faz sentido permitir que o usuário altere o valor do campo que exibe o resultado;
- Podemos alterar esta e outras características de campos textos com os métodos:
  - `setEditable(true/false)`: se pode ser editado;
  - `setEnabled(true/false)`: se está ativo;
  - `setColumns(numero)`: altera o tamanho relativo à quantidade de colunas (depende da fonte escolhida);
  - `setFont(fonte)`: altera a fonte utilizada;
  - `setText(string)`: altera o conteúdo.

Eventos

- Uma interface não tem muita utilidade sem captura de eventos;
- Para capturar eventos, precisamos de classes ouvintes (*listeners*);
- Começaremos com o ouvinte mais comum: `java.awt.event.ActionListener`;
  - Interface que define um único método: `actionPerformed(ActionEvent e);`
  - Criamos classes que implementam a interface;
  - Registramos estas classes como ouvintes nos componentes GUI.

ActionListener no JTextField

- Crie uma classe interna membro que implemente a interface `ActionListener`;
- Em seu método `actionPerformed()`, exiba uma mensagem qualquer com `JOptionPane`;
- No método `initComponents()`:
  - Crie uma instância de sua classe interna;
  - Registre-a como ouvinte nos dois primeiros campos de texto usando o método `addActionListener()` da classe `JTextField`.
- Teste sua GUI!

ActionListener no JTextField

- Vamos fazer algo de útil: calcular e exibir o MDC;
  - No método `actionPerformed()`, vamos obter os dados dos dois campos, calcular e exibir o resultado;
  - Precisamos transformar os campos textos em membros da janela para acessá-los;
  - Pode ser `private`, pois a classe interna tem acesso aos membros da classe externa, mesmo privativos;
  - Use métodos `getText()` e `setText()` da classe `JTextField`, fazendo as conversões `String` - `int`.

Novo exercício

- Crie uma janela para autenticação (*login*) do usuário;
- Ao preencher os campos e pressionar ENTER, verifique se é um dos usuários válidos:
  - Login: `fulano`, senha: `ful35`;
  - Login: `beltrano`, senha: `tran40`;
  - Login: `admin`, senha: `sa1000`.
- Em seguida, substitua o `JTextField` da senha por um `JPasswordField` e veja a diferença.

JPasswordField

- Funciona como um `JTextField`;
- Diferenças:
  - Oculta o conteúdo do campo, exibindo "caracteres eco" (geralmente asteriscos);
  - `getText()` é *deprecated*, use `getPassword()`, que retorna `char[]` ao invés de `String`.

```
// Para comparar a senha:
private boolean igual(char[] s1, char[] s2) {
   boolean igual = (s1.length == s2.length);
   for (int i = 0; igual && (i < s1.length); i++)
      igual = s1[i] == s2[i];
   return igual;
}
```

ActionListener: identificando a fonte

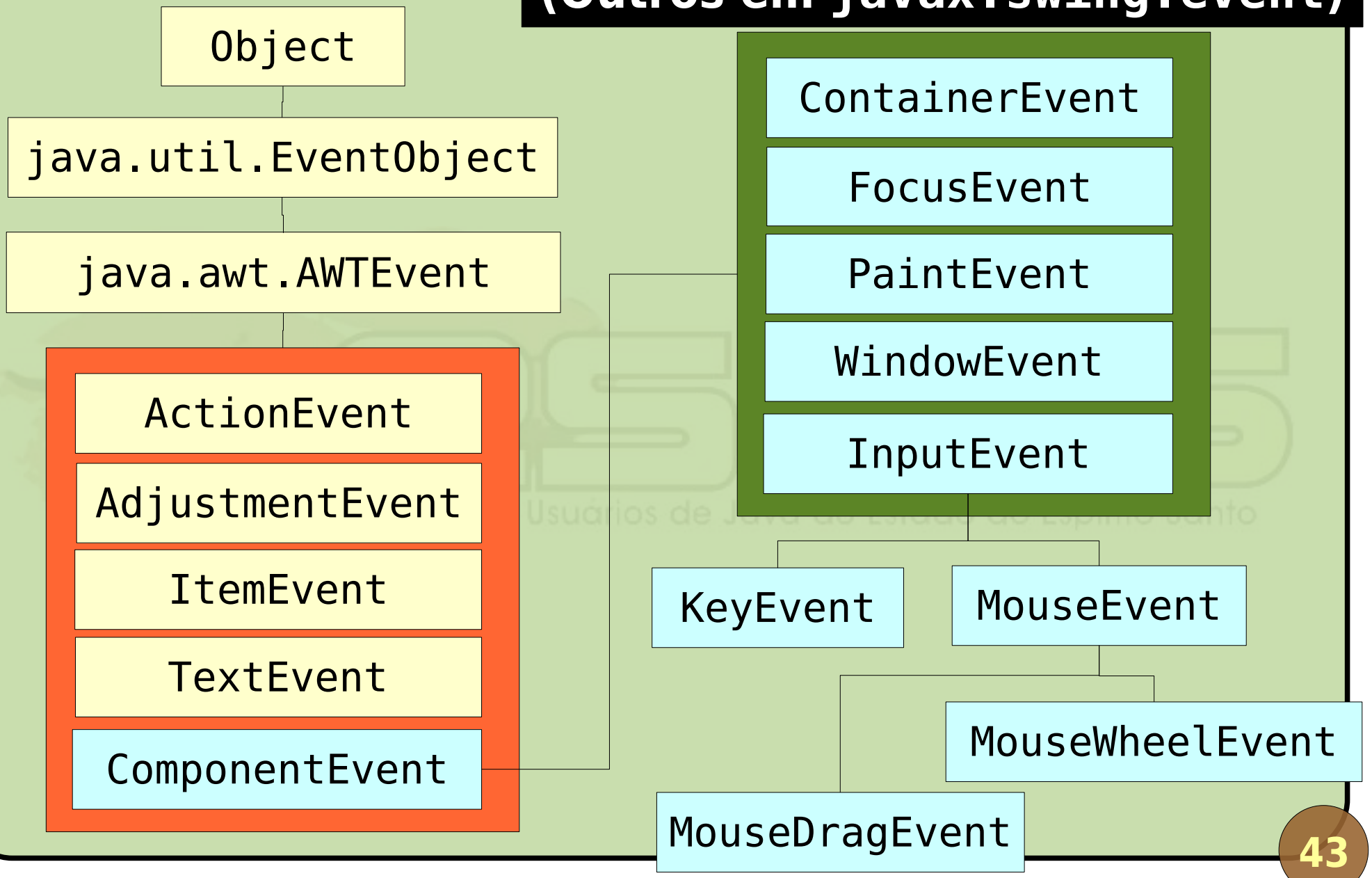
- Registramos o mesmo ouvinte para dois campos texto. Como identificar quem acionou o ouvinte?
- O método `actionPerformed()` recebe como parâmetro um objeto `ActionEvent`:
  - `ActionEvent.getSource()`: retorna o componente que acionou o evento;
  - `ActionEvent.getWhen()`: retorna o timestamp (`long`) do momento do acionamento.
- Exercite: se o usuário acionar a autenticação pelo campo de *login*, pergunte se ele não esqueceu da senha (use `JOptionPane`).

Sistema de delegação de evento

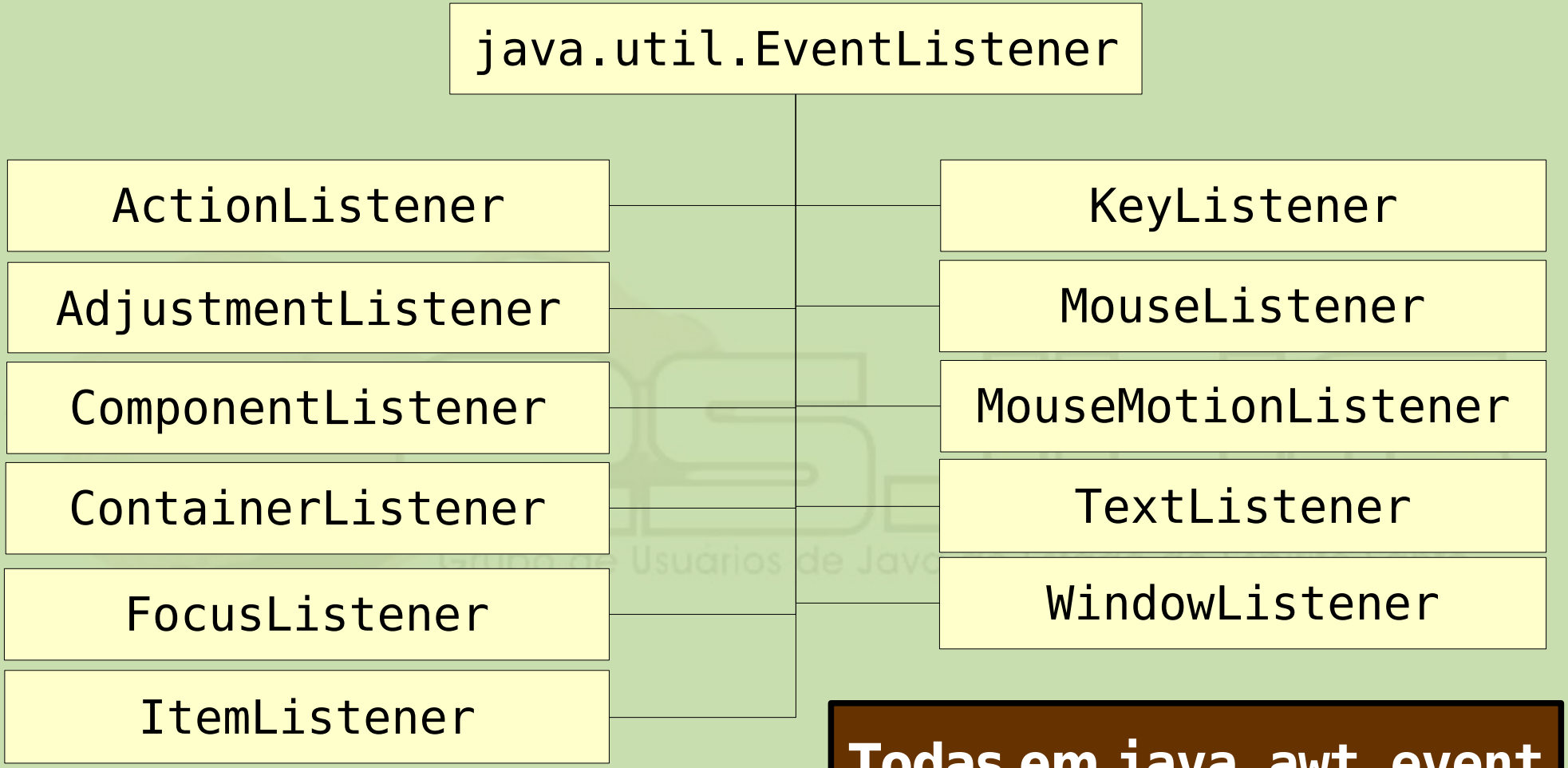
- Participantes:
  - Componentes GUI (*widgets*);
  - Ouvintes;
  - Eventos.
- Ouvintes registram-se junto aos componentes GUI (podemos registrar quantos quisermos);
- Quanto um evento ocorre, o sistema:
  - Cria um objeto que representa o evento;
  - Despacha o evento a todos os ouvintes registrados naquele componente para aquele tipo de evento.

Alguns eventos

java.awt
java.awt.event
(Outros em javax.swing.event)



Alguns ouvintes



Todas em `java.awt.event`
Há muitos outros...

JButton: adicionando botões

- Botões disparam eventos quando clicados;
- Crie instâncias de `javax.swing.JButton`;
  - Pode especificar rótulo e ícone no construtor.
- Crie um botão para sair do programa e outro pra efetuar o cálculo do MDC:
  - Funcionam como nos outros componentes:

`setEnabled()`, `setFont()`, `setIcon()`, `setHorizontalAlignment()`, `setHorizontalTextPosition()`, `setVerticalTextPosition()`, `setPreferredSize()`, `setVisible()`, `setText()`

JButton: também sofre ActionEvent

- Registre o ouvinte da calculadora no botão ao invés de nos campos de texto;
- Registre o botão como padrão da janela: `getRootPane().setDefaultButton();`
- Registre um ouvinte para o botão de sair usando uma classe interna anônima:

```
botaoSair.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      dispose();
   }
});
```

JButton: *rollover* e JToggleButton

- É possível configurar um ícone que aparece quando o cursor do *mouse* está acima do botão:
  - `setRolloverIcon(ícone)`.
- Existem também um botão que serve para marcar/desmarcar uma opção:
  - Adicione um botão “travar”, trocando `JButton` por `javax.swing.JToggleButton`;
  - Só efetue o cálculo se o botão estiver destravado (método `isSelected()`);
  - Se quiser ir além, registre um ouvinte para o botão “travar” que troque seu nome e ícone.

JCheckBox: opções sim/não

- Caixas de marcação funcionam como *toggle buttons*, marcando/desmarcando opções;
- Crie instâncias de `javax.swing.JCheckBox`;
  - Pode especificar rótulo, ícone e se está selecionada.
- Crie *checkboxes* para “negrito” e “itálico”:
  - Determine a fonte do campo resultado (`setFont()`);
  - Mude a fonte quando as caixas forem marcadas (use a interface `ItemListener` e `addItemListener()`).

```
Font f = textoMdc.getFont();
textoMdc.setFont(fonte.deriveFont(Font.BOLD +
Font.ITALIC));
```

Font: brincando com fontes

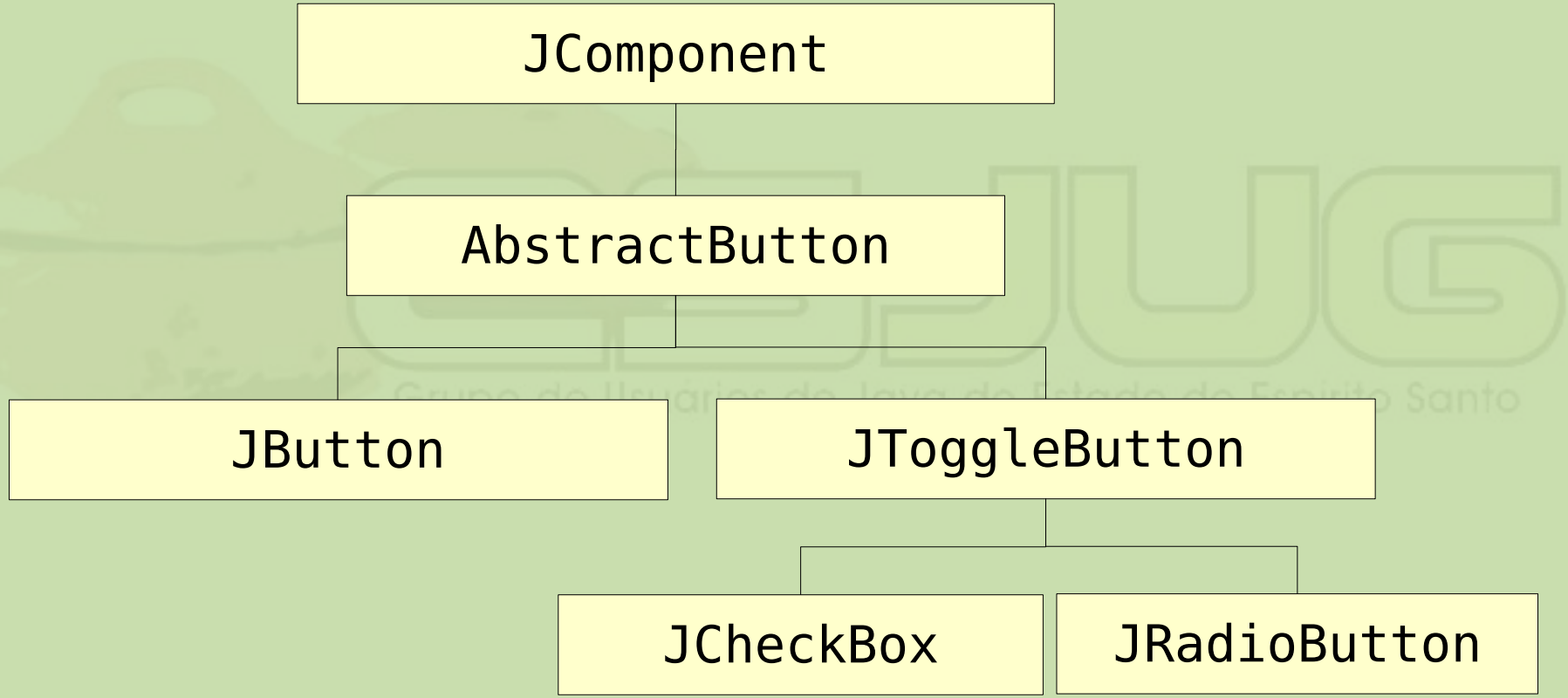
- A classe `java.awt.Font` representa fontes;
- Construtor especifica nome, estilo e tamanho;
- Constantes representam estilo, como `PLAIN`, `BOLD` e `ITALIC`;
- `Font.createFont()` cria uma fonte a partir de um arquivo no disco rígido;
- `deriveFont()` cria uma derivação de uma fonte (ex.: Courier normal para Courier negrito).

JRadioButton: opções exclusivas

- Botões de rádio funcionam como *checkboxes*, mas só um deles pode estar selecionado;
- Instâncias de `javax.swing.JRadioButton`;
  - Pode especificar rótulo, ícone e se está selecionada.
- Troque as *checkboxes* por *radio buttons* "normal", "negrito", "itálico" e "negrito itálico":
  - Dica: coloque a fonte como parâmetro do ouvinte inicializado no construtor e crie 4 objetos ouvintes.
- Botões de rádio precisam ser agrupados:
  - Use a classe `javax.swing.ButtonGroup` e seu método `add()`.

Hierarquia dos botões Swing

- A classe ButtonGroup funciona com qualquer instância indireta de AbstractButton.



JComboBox: lista *drop down*

- Uma caixa de combinação permite selecionar um item da lista ou informar um novo;
- Crie uma nova janela usando `BorderLayout`:
  - Um rótulo vazio no centro para mostrar imagens;
  - Uma *combo* no rodapé para selecionar a imagem.
- Crie instância de `javax.swing.JComboBox`:
  - Elementos da lista: vetor de strings no construtor;
  - `setMaximumRowCount(num)`: máximo de linhas;
  - `setEditable(true/false)`: se pode editar;
  - `getSelectedItem()`: retorna item selecionado.

JList: lista de seleção simples

- Uma lista é como uma *combo box* não-editável aberta, exibindo vários itens por vez;
- Incremente a GUI anterior colocando uma lista no cabeçalho com várias cores para escolher;
  - Instância de `javax.swing.JList`;
  - Elementos da lista: vetor de strings no construtor;
  - `setVisibleRowCount(num)`: linhas exibidas;
  - `setSelectionMode(ListSelectionModel.SINGLE_SELECTION)`: só pode selecionar um por vez;
  - Ao adicionar na janela, decore-o com barras de rolagem: `new JScrollPane(minhaLista)`.

JList: tratamento de eventos

- Use `ListSelectionListener`;
- Métodos da lista para obter o elemento:
  - `getSelectedIndex()`: obtém o índice do elemento selecionado (`-1` para nenhum elemento);
  - `getSelectedValue()`: retorna o elemento selecionado;
- Ao escolher uma cor, pinte o fundo da janela:
  - Constantes da classe `java.awt.Color`;
  - Para trocar a cor de fundo de um `JFrame`: `getContentPane().setBackgroundColor()`.

Modos de seleção de listas

- Constantes da classe `ListSelectionModel`:
  - `SINGLE_SELECTION`: só um elemento;
  - `SINGLE_INTERVAL_SELECTION`: só um intervalo (vários elementos contíguos);
  - `MULTIPLE_INTERVAL_SELECTION`: múltiplos intervalos (vários elementos espalhados).
- Você pode criar sua própria política de seleção extendendo `ListSelectionModel`.

JList: lista de seleção múltipla

- Use `MULTIPLE_INTERVAL_SELECTION` para deixar o usuário escolher várias cores;
- Misture as cores:
  - Obtenha os valores de RGB com `getRed()`, `getGreen()` e `getBlue()`;
  - Some todos e tire a média aritmética;
  - Crie uma nova cor especificando no construtor seus valores para R, G e B;
  - Determine a cor de fundo usando a nova cor.
- Métodos de `JList`: `getSelectedIndices()` e `getSelectedValues()`.

JTextArea: campo de texto multi-linha

- Áreas de texto permitem que escrevamos várias linhas de texto;
- Instâncias de `javax.swing.JTextArea`;
- Crie uma janela com um campo texto no cabeçalho e uma área de texto no centro;
  - Ao digitar no campo e pressionar ENTER, adicione a linha à área de texto e limpe o campo;
  - Pode especificar linhas e colunas no construtor;
  - Adicione-a à janela decorada por um painel de rolagem, assim como a lista.

JTextArea e JScrollPane: opções

- JTextArea:
  - setLineWrap(true/false): quebras de linha;
  - setWrapStyleWord(true/false): quebra na palavra.
- JScrollPane:
  - setHorizontalScrollBarPolicy(constante): política de exibição das barras de rolagem;
- Constantes de JScrollPane:
  - HORIZONTAL_SCROLLBAR_ALWAYS;
  - HORIZONTAL_SCROLLBAR_AS_NEEDED;
  - etc.

Eventos de *mouse*

MouseListener

- mousePressed()
- mouseClicked()
- mouseReleased()
- mouseEntered()
- mouseExited()

MouseMotionListener

- mouseDragged()
- mouseMoved()

MouseWheelListener

- mouseWheelMoved()

São capturados por qualquer Component

MouseInputListener

Eventos de *mouse*

- Objeto `MouseEvent` encapsula informações sobre o eventos de *mouse* e movimento:
  - `getButton()`: botão clicado (direito, esquerdo, ...);
  - `getClickCount()`: quantos cliques;
  - `getPoint()`, `getX()` e `getY()`: posição clicada;
  - etc.
- Objeto `MouseWheelEvent` faz o mesmo:
  - `getScrollAmount()`: quantidade rolada;
  - `getScrollType()`: tipo de rolagem;
  - etc.

Experimentar eventos de *mouse*

- Crie uma janela usando `BorderLayout` com um rótulo no centro e outro no rodapé;
- Trate eventos de *mouse* no rótulo do centro:
  - Pressionar, clicar e soltar: indicar no rodapé em que coordenada ocorreu o clique;
  - Ao entrar: pintar o fundo do rótulo de verde (use `setOpaque(true)` para ficar opaco);
  - Ao sair: pintar o fundo do rótulo de vermelho;
  - Ao mover ou arrastar: indicar a posição no rodapé;
  - Se usar uma classe só como ouvinte de *mouse* e movimento, lembre-se de adicioná-la duas vezes.

Classes adaptadoras (`java.awt.event`)

- Classes que implementam interfaces `Listener`:
  - Provêem implementações vazias dos métodos;
  - Vantagem: não escrever métodos vazios nos ouvintes que criarmos;
  - Desvantagem: se for escrita a assinatura errada do método, o compilador não indicará erros.

ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

Novo experimento

- Similar ao anterior, porém usando um adaptador e registrando o ouvinte diretamente no `JFrame`;
- Mude a cor de fundo da janela:
  - Clique, botão esquerdo: verde;
  - Duplo-clique, botão esquerdo: azul;
  - Clique, botão direito: ciano;
  - Clique, botão do meio: magenta.
- Para mudar a cor do `JFrame`, mude a cor do seu "*content pane*" (`getContentPane()`).

Como determinar o botão clicado

- Na classe `MouseEvent`:
  - `isMetaDown()` retorna `true` quando o botão clicado é o botão direito;
  - `isAltDown()` returna `true` quando o botão clicado é o botão do meio;
  - `getButton()` retorna um `int`, sendo:
    - `MouseEvent.BUTTON1`: botão esquerdo;
    - `MouseEvent.BUTTON2`: botão do meio;
    - `MouseEvent.BUTTON3`: botão direito.

JPanel: aprendendo sobre desenho

- Painéis possuem dois objetivos principais:
  - Organizar componentes utilizando gerenciadores de *layout* diversos;
  - Desenhar formas geométricas na tela com Java2D.
- Mostraremos primeiro um pouco de Java2D;
- Aprenderemos como Swing desenha componentes;
- Para criar: instância de `javax.swing.JPanel`;
  - Experimente criar um painel, determinar sua cor de fundo e colocá-lo numa posição e tamanho arbitrários numa janela com *layout manager* nulo.

JPanel: o método paintComponent()

- Quando um `JComponent` precisa ser exibido, Java chama o seu método `paintComponent()`;
- Este método pode ser sobrescrito para desenhar o componente de forma diferente;
- Crie um painel que armazena numa lista todos os pontos que o usuário clicar:
  - Crie uma classe que estende `JPanel`;
  - Crie o método `initComponents()`, como sempre;
  - Neste método, adicione uma classe interna anônima que estende `MouseAdapter`;
  - Trate o evento de clique no painel.

JPanel: o método paintComponent()

- Chame o método `repaint()` ao final do tratamento do clique (para redesenhar o painel);
- Sobrescreva `paintComponent()` deste painel:

```
protected void paintComponent(Graphics g) {
   super.paintComponent(g);   // Essencial!
   for (Point p : pontos)
      g.fillOval(p.x, p.y, 4, 4);
}
```

- Coloque este painel como painel de conteúdo da janela: `setContentPane()`.
- Execute e experimente!

Desenhando com JPanel

- O que acontece se fizermos no evento de arraste de *mouse* o que fizemos com o de clique?
- E se trocarmos os parâmetros “`4, 4`” do método `fillOval()` por números maiores?
- Experimente outros métodos de desenho:
  - `fillArc();`
  - `fillPolygon();`
  - `fillRect();`
  - `fillRoundRect()`.

Eventos de teclado

- A interface KeyListener define os eventos:
  - `keyPressed()`: tecla pressionada;
  - `keyReleased()`: tecla liberada;
  - `keyTyped()`: tecla digitada (pressionada / liberada).
- A classe `KeyEvent` traz características da tecla:
  - `getKeyChar()`: caractere associado à tecla;
  - `getKeyCode()`: código da tecla (`KeyEvent` possui constantes que representam códigos válidos);
  - `isActionKey()`: se é uma tecla de ação;
  - `getModifiers()`: teclas modificadoras (ex.: shift).

Limpando o desenho

- Adicione código de tratamento de teclado ao painel de desenho;
- Ao pressionar L, limpe o desenho:
  - `getKeyCode()` deve retornar `VK_L`.
- Em seguida, altere para Ctrl+L:
  - `getModifiers() & CTRL_MASK` deve ser igual a `CTRL_MASK`.
- Observação: trate tecla pressionada ou liberada. No evento de tecla digitada o código da tecla é sempre 0 (zero).

JPanel: organizando componentes

- Podemos usar painéis para organizar componentes;
- Para isso, precisaremos conhecer mais os gerenciadores de *layout*;
- Ao criar um `JPanel`, podemos especificar o gerenciador;
- O método `setLayout(gerenciador)` permite alterar o gerenciador;
- O gerenciador padrão é o `BorderLayout`.

Gerenciadores de *layout* revisitados

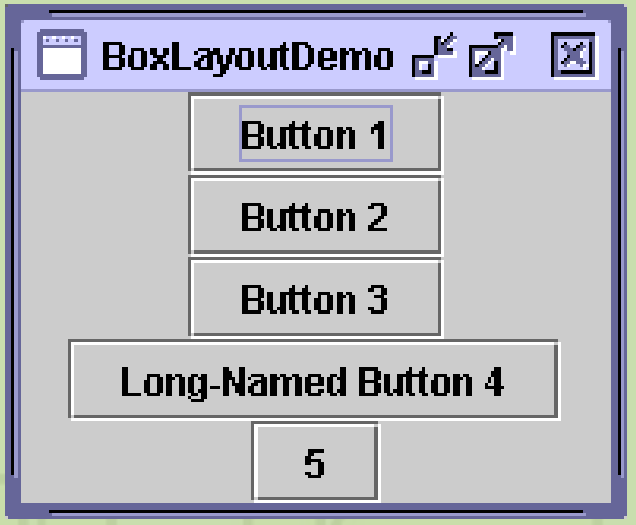
- Já sabemos o que são gerenciadores de *layout* e como funcionam;
- JFC/Swing traz 7 gerenciadores (além do nulo):
  - `java.awt.BorderLayout;` ✔
  - `javax.swing.BoxLayout;`
  - `java.awt.CardLayout;`
  - `java.awt.FlowLayout;` ✔
  - `java.awt.GridBagLayout;`
  - `java.awt.GridLayout;`
  - `javax.swing.SpringLayout.`

FlowLayout e BorderLayout

- Mais características do `FlowLayout`:
  - Pode determinar alinhamento e espaçamento no construtor;
  - `setAlignment(constante)`: muda alinhamento;
  - `setHGap()` e `setVGap()`: mudam espaçamento;
  - `layoutContainer(container)`: refaz o *layout*.
- Mais características do `BorderLayout`:
  - Pode especificar espaçamento no construtor ou com os métodos `setHGap()` e `setVGap()`;
  - `layoutContainer(container)`: refaz o *layout*.

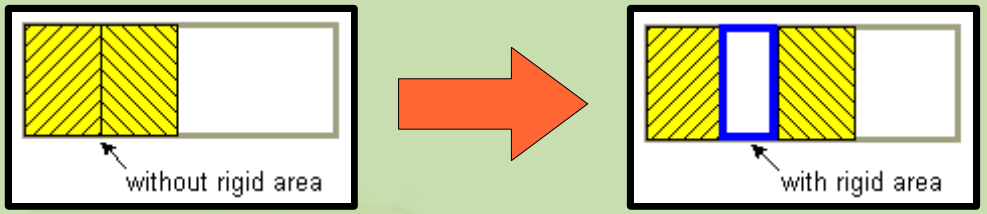
BoxLayout: linha ou coluna

- Dispõe os elementos numa mesma linha ou numa mesma coluna;
- No construtor, especificar o painel que será gerenciado e a direção do *layout*:
  - `PAGE_AXIS`: eixo Y, componentes na vertical;
  - `LINE_AXIS`: eixo X, componentes na horizontal.

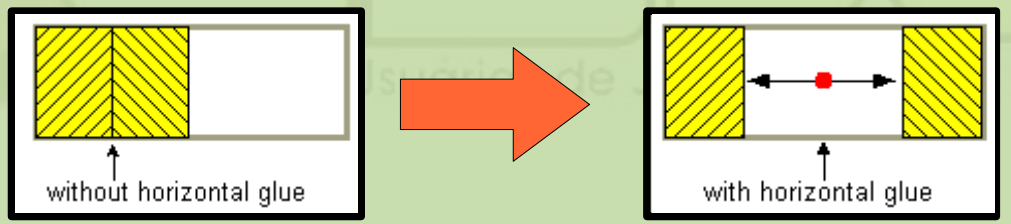


BoxLayout: linha ou coluna

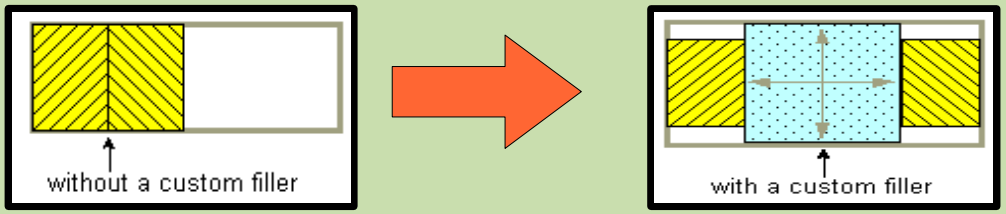
- Recursos invisíveis:
  - `Box.createRigidArea();`



  - `Box.createHorizontalGlue();`
  - `Box.createVerticalGlue();`

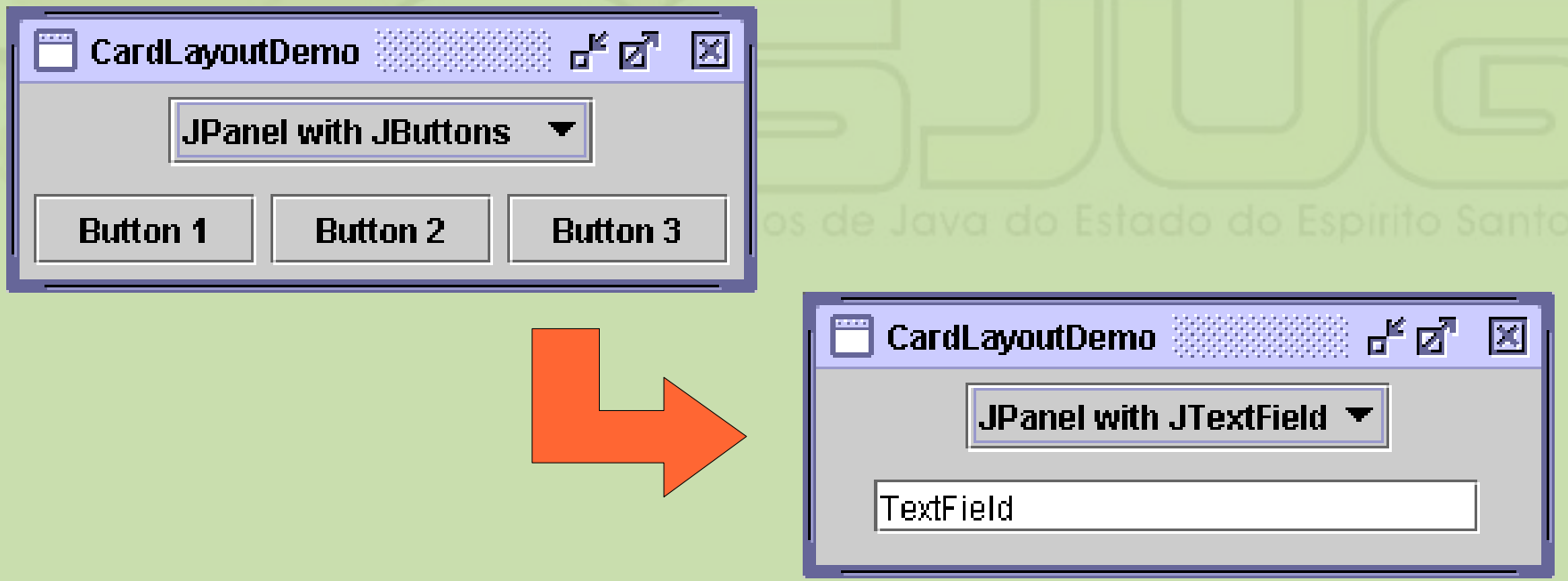


  - `new Box.Filler().`



CardLayout: troca de componentes

- Área que contém componentes diferentes em tempos diferentes;
- Alterna entre grupos de componentes, como em um painel de abas.

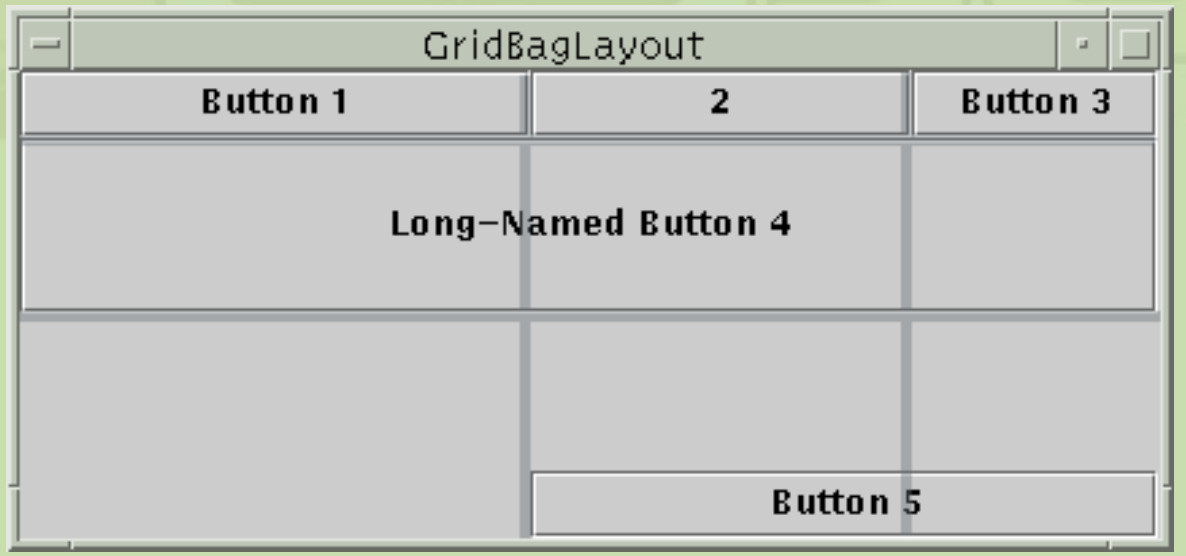


GridLayout: componentes em grade

- 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], ...).

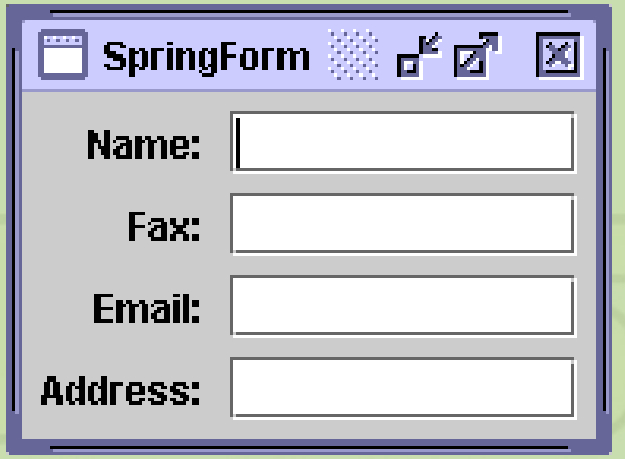
GridBagLayout: grade sofisticada

- 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.



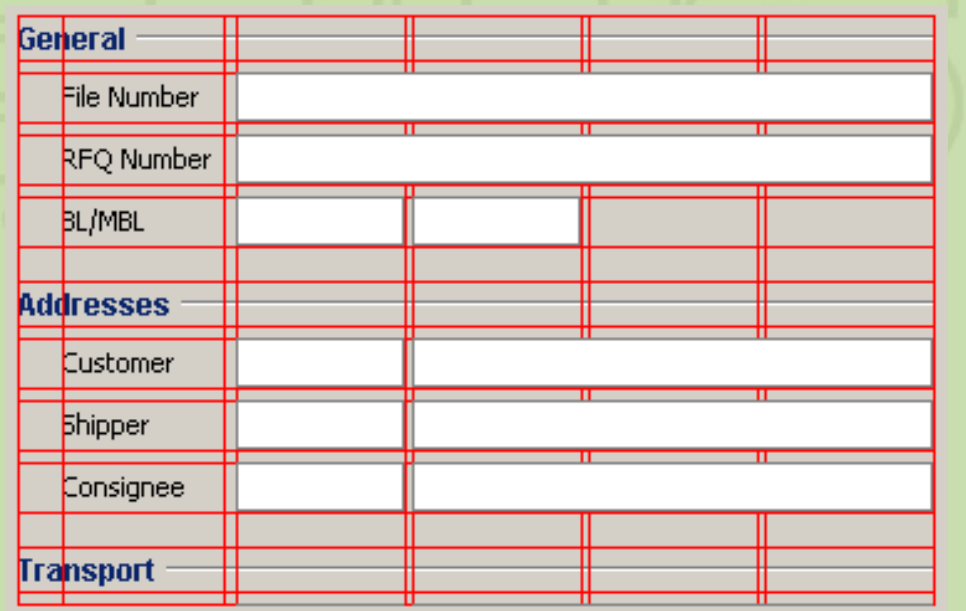
SpringLayout: para uso em IDEs

- Permite especificar relacionamentos entre as margens de cada componente GUI;
- Mais recente, construído para uso em IDEs;
- Também muito complexo para usar manualmente.



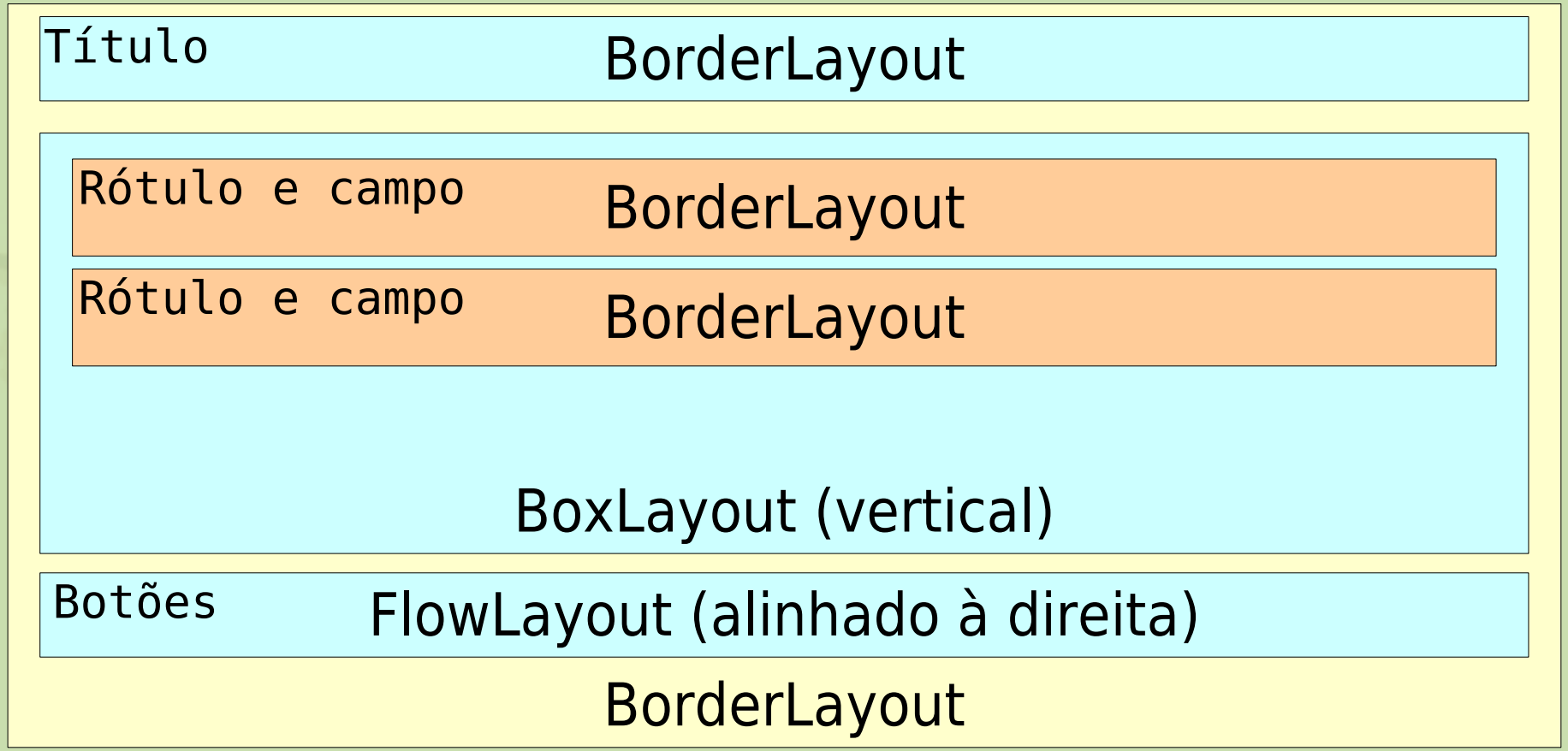
Outros gerenciadores

- NetBeans 5.0 traz um novo gerenciador:
  - `GroupLayout`;
  - Fará parte do Java SE na versão 6.0.
- O projeto JGoodies tem um outro gerenciador:
  - `FormLayout`.
- Qualquer um pode criar um gerenciador.



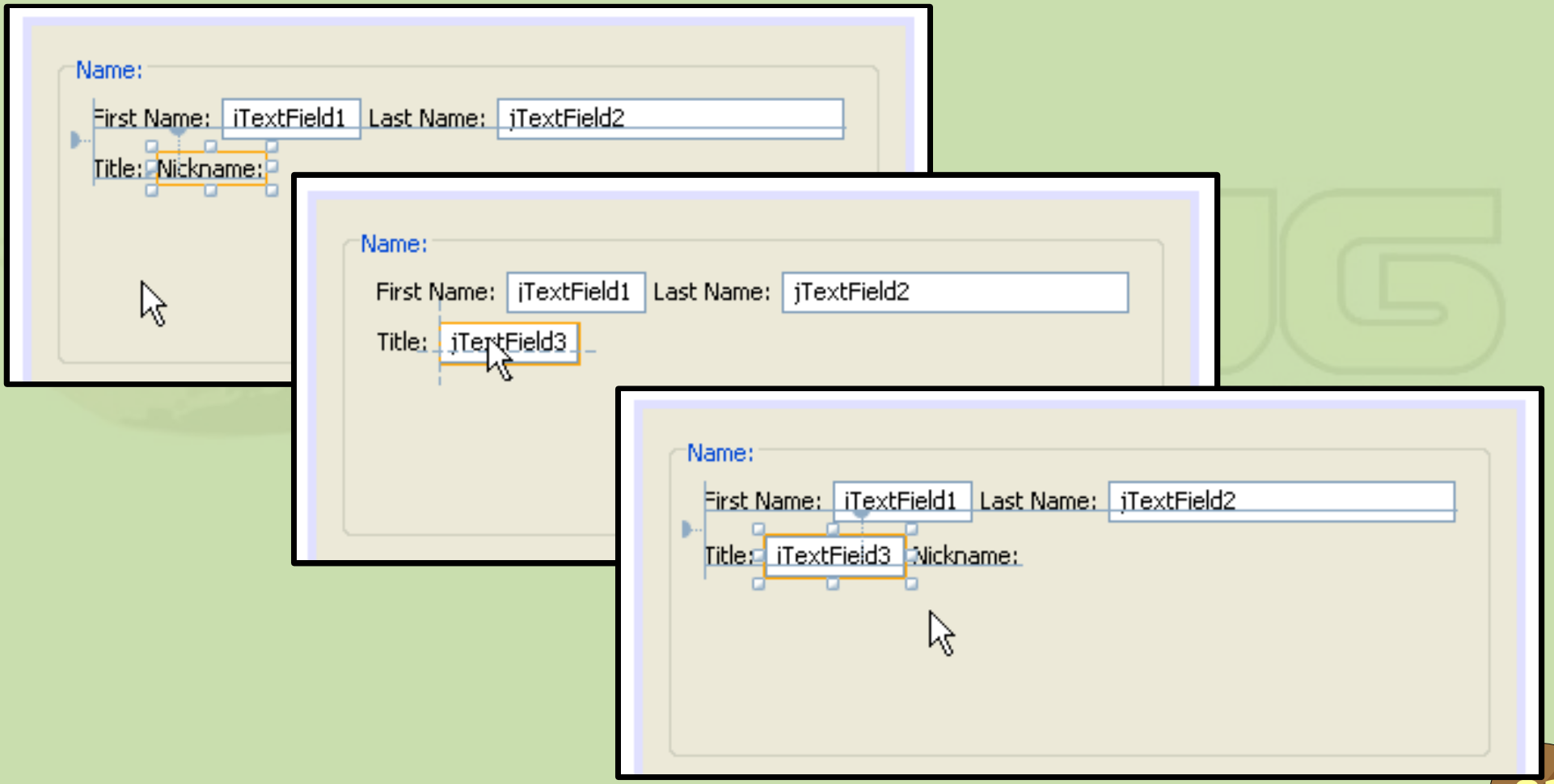
Misturando gerenciadores

- Seria possível construir nossa calculadora de forma que possamos redimensionar a janela?



Usando IDEs

- Vamos exercitar o uso de *GUI Builders*;
- Usaremos o NetBeans 5.0 com o editor Matisse.



Conclusões

- Vimos nesta parte do curso:
  - Conceitos básicos sobre criação de GUIs em Java usando a API Swing;
  - Componentes gráficos simples como rótulos, campos de texto, botões, listas e painéis;
  - O sistema de tratamento de eventos e alguns eventos comuns aos componentes apresentados;
  - Gerenciadores de *layout* e seu papel na disposição dos componentes;
  - Uso da IDE NetBeans 5.0 para criação de interfaces gráficas de forma visual.