

Interface Gráfica com Swing

SCC0604 - Programação Orientada a Objetos

Prof. Fernando V. Paulovich

<http://www.icmc.usp.br/~paulovic>

paulovic@icmc.usp.br

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

25 de julho de 2010



Sumário

- 1 Conceitos Introdutórios
- 2 Gerenciamento de Layout
- 3 Composição de Telas
- 4 Componentes Gráficos
- 5 Gerenciamento de Layout Sofisticado
- 6 Criando Menus
- 7 Caixas de Diálogo

Sumário

- 1 Conceitos Introdutórios
- 2 Gerenciamento de Layout
- 3 Composição de Telas
- 4 Componentes Gráficos
- 5 Gerenciamento de Layout Sofisticado
- 6 Criando Menus
- 7 Caixas de Diálogo

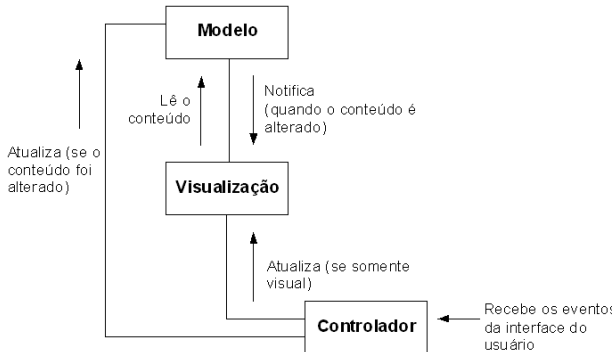
Introdução

- No capítulo anterior foi mostrado como é feito o tratamento de eventos no swing
- Agora partiremos para criar interfaces gráficas com alguns elementos oferecidos pelo swing

O Padrão de Projeto Modelo-Visão-Controlador (MVC)

- O Swing segue a essência do padrão de projeto MVC
 - Modelo - armazena um conteúdo
 - Visão - exibe o conteúdo
 - Controlador - processa a entrada de dados do usuário

O Padrão de Projeto Modelo-Visão-Controlador (MVC)



Sumário

- 1 Conceitos Introdutórios
- 2 Gerenciamento de Layout**
- 3 Composição de Telas
- 4 Componentes Gráficos
- 5 Gerenciamento de Layout Sofisticado
- 6 Criando Menus
- 7 Caixas de Diálogo

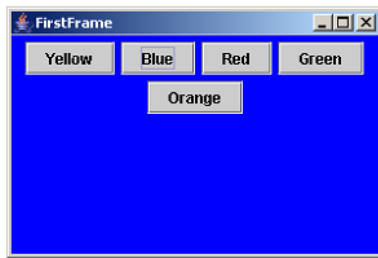
Uma Introdução ao Gerenciamento de Layout

- Antes de prosseguirmos com os componentes individuais Swing, é preciso entender como organizar esses componentes dentro de um quadro (ou painel)

Uma Introdução ao Gerenciamento de Layout

- O que aconteceria se novos botões fossem adicionados ao exemplo dado no capítulo passado (o que pinta o fundo de acordo com o botão clicado)?

Uma Introdução ao Gerenciamento de Layout



Uma Introdução ao Gerenciamento de Layout

- É possível notar que os botões são centralizados em uma linha, e quando não há mais espaço em uma linha, uma nova é iniciada
- Isso acontece porque quando elementos são inseridos em algum contêiner (painel), a distribuição dos elementos nesse contêiner é gerenciado automaticamente pelo Java por meio de um gerenciador de layout

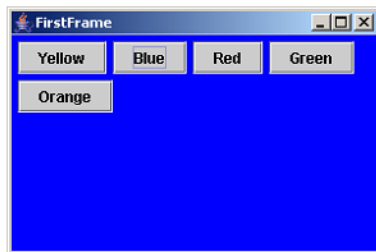
Uma Introdução ao Gerenciamento de Layout

- Nesse caso, o gerenciador se chama **Flow Layout**, que é o gerenciador padrão de um painel
- Assim, quando o contêiner é redimensionado, o gerenciador reorganiza automaticamente seus elementos
- Para se configurar um layout, o comando **setLayout()** deve ser utilizado

Uma Introdução ao Gerenciamento de Layout

```
1 public class MyPanel extends JPanel {  
2  
3     private JButton yellowButton = new JButton("Yellow");  
4     ...  
5  
6     public MyPanel() {  
7         ...  
8         this.setLayout(new FlowLayout(FlowLayout.LEFT));  
9     }  
10  
11     class OuvinteBotao implements ActionListener {  
12         ...  
13     }  
14 }
```

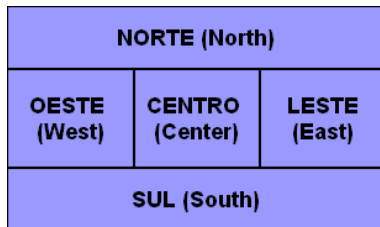
Uma Introdução ao Gerenciamento de Layout



Gerenciador BorderLayout

- Um dos mais gerenciadores de layout mais interessante é o gerenciador de borda, **BorderLayout**
- Esse é o gerenciador padrão do painel de conteúdo dos quadros (Frames)
- Esse gerenciador permite escolher onde se colocar o(s) componente(s): Norte, Sul, Centro, Leste e Oeste
- Os componentes de borda são colocados primeiro, e o espaço remanescente é ocupado pelo centro

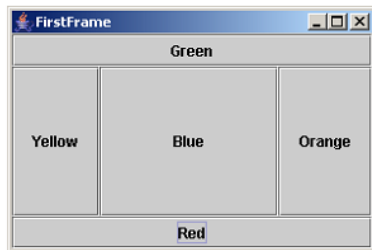
Layout de Borda



Layout de Borda

```
1 public class MyPanel extends JPanel {  
2  
3     private JButton yellowButton = new JButton("Yellow");  
4     ...  
5  
6     public MyPanel() {  
7         this.setLayout(new BorderLayout());  
8         this.add(yellowButton, "West");  
9         this.add(blueButton, "Center");  
10        this.add(redButton, "South");  
11        this.add(greenButton, "North");  
12        this.add(orangeButton, "East");  
13        ...  
14    }  
15  
16    ...  
17 }
```

Resultado



Sumário

- 1 Conceitos Introdutórios
- 2 Gerenciamento de Layout
- 3 Composição de Telas**
- 4 Componentes Gráficos
- 5 Gerenciamento de Layout Sofisticado
- 6 Criando Menus
- 7 Caixas de Diálogo

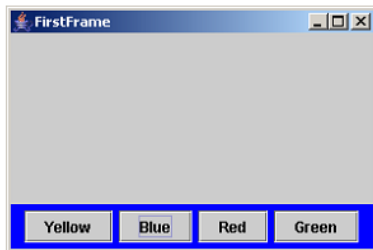
Painéis

- Para se obter uma melhor estruturação da tela, painéis podem ser aninhados
- Pode-se colocar painéis (panel) dentro de quadros como se fossem elementos (usando algum layout), e depois colocar dentro desses painéis outros elementos (usando outro layout)

Painéis

```
1 public class FirstFrame extends JFrame {  
2  
3     public FirstFrame() {  
4         setTitle("FirstFrame");  
5         setSize(300, 200);  
6         this.setDefaultCloseOperation(EXIT_ON_CLOSE);  
7  
8         Container contentPane = getContentPane();  
9         contentPane.setLayout(new BorderLayout());  
10        contentPane.add(new MyPanel(), "South");  
11    }  
12 }
```

Resultado



Sumário

- 1 Conceitos Introdutórios
- 2 Gerenciamento de Layout
- 3 Composição de Telas
- 4 Componentes Gráficos**
- 5 Gerenciamento de Layout Sofisticado
- 6 Criando Menus
- 7 Caixas de Diálogo

Entradas de Texto

- Na linguagem Java são usados dois componentes para obter entrada de texto: campo de texto (**JTextField**) e áreas de texto (**JTextArea**)
- Assim como um botão, uma entrada de texto deve ser simplesmente adicionada a um contêiner (painel); Existem vários métodos para se manipular esses componentes, mas os principais são
 - void setText(String t)
 - String getText()
 - void setEditable(boolean b)

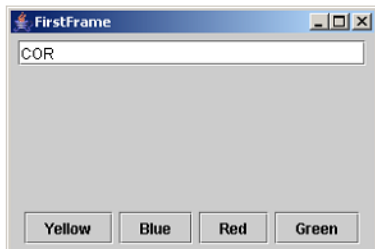
Código

```
1 public class FieldPanel extends JPanel {  
2  
3     private JTextField textField = new JTextField("COR",25);  
4  
5     public FieldPanel() {  
6         this.add(textField);  
7     }  
8 }
```

Código

```
1 public class FirstFrame extends JFrame {  
2  
3     public FirstFrame() {  
4         setTitle("FirstFrame");  
5         setSize(300, 200);  
6         this.setDefaultCloseOperation(EXIT_ON_CLOSE);  
7  
8         Container contentPane = getContentPane();  
9         contentPane.setLayout(new BorderLayout());  
10        contentPane.add(new FieldPanel(),"North");  
11        contentPane.add(new MyPanel(),"South");  
12    }  
13 }
```

Resultado



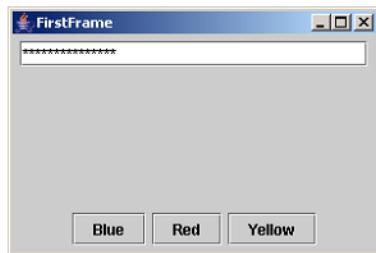
Campos de Senha

- Para manipular senhas, existe uma classe especial denominada **JPasswordField**
- Nessa classe podemos encontrar os métodos
 - void setEchoChar(char eco)
 - char[] getPassword()

Código

```
1 public class FieldPanel extends JPanel {  
2  
3     private JPasswordField textField = new JPasswordField("COR",25);  
4  
5     public FieldPanel() {  
6         this.add(textField);  
7     }  
8 }
```

Resultado



Áreas de Texto

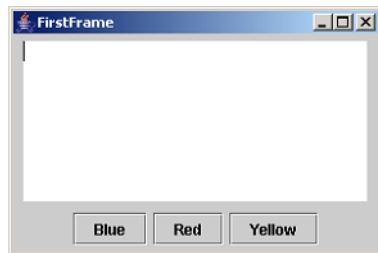
- Algumas vezes é necessário coletar dados de entrada de usuário com mais de uma linha de comprimento, nesses casos **JTextArea** deve ser empregado
- No construtor do **JTextArea** se define o número de linhas e colunas da área

```
1 JTextArea textArea = new JTextArea(8,40);
```

Código

```
1 public class FieldPanel extends JPanel {  
2  
3     private JTextArea textField = new JTextArea(8,25);  
4  
5     public FieldPanel() {  
6         this.add(textArea);  
7     }  
8 }
```


Resultado



Áreas de Texto

- Se houver mais texto que a área de texto consegue exibir, então o texto restante será cortado. Pode-se evitar cortar as linhas mais longas ativando a mudança de linha automática

```
1 textArea.setLineWrap(true);
```

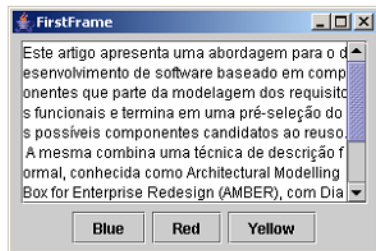
Áreas de Texto

- No Swing, uma área de texto não possui barras de rolagem. Se você quiser, terá que inserir a área de texto dentro de um painel de rolagem (**JScrollPane**), depois inserir o painel de rolagem dentro do painel

Código

```
1 public class FieldPanel extends JPanel {  
2     private JTextArea textArea = new JTextArea(8,25);  
3     private JScrollPane scrollPane = new JScrollPane(textArea);  
4  
5     public FieldPanel() {  
6         this.add(scrollPane);  
7         textArea.setLineWrap(true);  
8     }  
9 }
```

Resultado



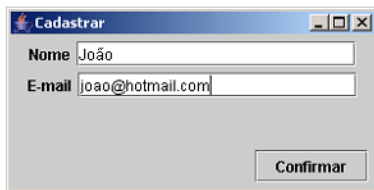
Rótulos e Como Rotular Componentes

- Os rótulos são componentes que contêm uma linha de texto simples. A classe que implementa os rótulos é chamada de **JLabel**
- Essa classe tem construtores que permitem especificar o texto ou o ícone inicial e, opcionalmente, o alinhamento do conteúdo

```
1 JLabel label = new JLabel("Texto", JLabel.LEFT);
```

- Essa classe apresenta os seguintes métodos
 - void setText(String texto)
 - void setIcon(Icon icone)

Exercício: Gerar a tela abaixo



The image shows a Java Swing window titled "Cadastrar". It contains two text input fields. The first field is labeled "Nome" and contains the text "João". The second field is labeled "E-mail" and contains the text "joao@hotmail.com". At the bottom right of the window is a button labeled "Confirmar".

Como Fazer Escolhas: Caixas de Seleção

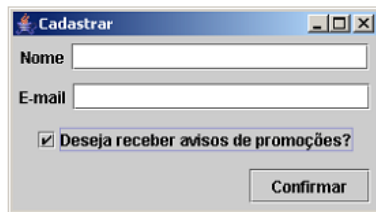
- Se for necessário apenas uma entrada do tipo “sim” ou “não”, use um componente chamado caixa de seleção (**JCheckBox**)
- As caixas de seleção precisam de um rótulo ao lado para identificar sua finalidade. Esse texto é passado direto no seu construtor

```
1 JCheckBox checkBox = new JCheckBox("Texto");
```


Como Fazer Escolhas: Caixas de Seleção

- Usa-se o método **setSelected()** para marcar ou desmarcar uma caixa de seleção
- Quando o usuário clica em uma caixa de seleção isso dispara um **ActionEvent**. Dessa forma, esse evento pode ser tratado dentro de um **actionPerformed()**
- Para saber se uma caixa de seleção está selecionada, use o método **isSelected()**

Resultado



A screenshot of a Java Swing window titled "Cadastrar". The window has a standard title bar with minimize, maximize, and close buttons. Inside the window, there are two text input fields: one labeled "Nome" and another labeled "E-mail". Below these fields is a checkbox that is checked, with the text "Deseja receber avisos de promoções?". At the bottom right of the window is a button labeled "Confirmar".

Como Fazer Escolhas: Botões de Rádio

- Se necessário, dar uma olhada na página 368 do livro “Core Java 2 Volume 1 - Fundamentos”

Bordas

- O *Swing* propicia um conjunto de bordas para se agrupar elementos em comum
- Uma borda é implementada na classe **Border**
- Para se usar essa classe o pacote **`javax.swing.border.*`** deve ser importado

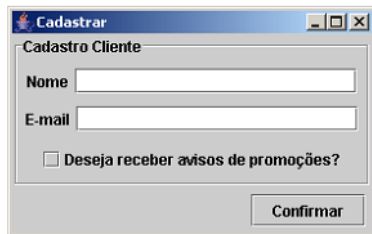
Bordas

- Existem vários tipos de bordas, mas para todas o seguinte procedimento deve ser seguido
 - Chame o método **BorderFactory()** para criar uma borda (existem vários tipos)
 - Se quiser adicione um título a borda usando o método **BorderFactory.createTitledBorder()**
 - Adicione a borda ao componente usando o método **setBorder()**

Exemplo Borda

```
1 public class FirstFrame extends JFrame {  
2     public FirstFrame(){  
3         ...  
4         //Criar o Panel dos Campos  
5         Border borda = BorderFactory.createEtchedBorder();  
6         Border campos = BorderFactory.createTitledBorder(borda, "Cadastro ←  
            Cliente");  
7         panelCampos.setBorder(campos);  
8         ...  
9     }  
10  
11     ...  
12 }
```

Resultado



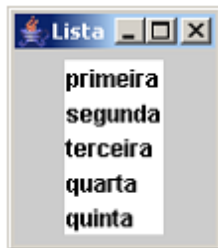
The image shows a Java Swing window titled "Cadastrar" (Register). The window has a standard title bar with minimize, maximize, and close buttons. Inside the window, there is a section titled "Cadastro Cliente" (Client Registration). This section contains two text input fields: one labeled "Nome" (Name) and another labeled "E-mail". Below these fields is a checkbox labeled "Deseja receber avisos de promoções?" (Do you want to receive promotion notices?). At the bottom right of the window, there is a button labeled "Confirmar" (Confirm).

Lista de Elementos

- Em Java, uma lista é formada usando-se a classe **JList**
- Para elaborar uma lista simples (**String**), primeiro crie um vetor de strings e depois passe esse vetor para o construtor do **JList**

```
1 String[] words = {"primeira", "segunda", "terceira", "quarta"};  
2 JList wordList= new JList(words);
```


Lista de Elementos



Lista de Elementos

- Por padrão, uma **JList** não oferece uma barra de rolagem se houver mais itens na lista que o número de linhas visíveis; nesse caso, usa-se um objeto **JScrollPane** para fornecer capacidade de rolagem
- Por sua vez, é essa barra de rolagem que deve ser adicionada ao quadro (ou painel) requerido

```
1 JScrollPane scrollList = new JScrollPane(wordList);  
2 ...  
3 painel.add(scrollList);
```

Lista de Elementos

- A priori, o componente lista exibe oito itens
- Use o método **setVisibleRowCount()** para mudar esse valor

```
1 wordList.setVisibleRowCount(4);
```

Resultado



Lista de Elementos

- Em uma lista é possível se restringir o usuário a um modo de seleção mais limitada com o método **setSelectionMode()**

```
1 wordList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION)
2 wordList.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION↵
  )
```

Lista de Elementos

- O tratamento de eventos de uma lista não é tão simples quanto de um botão, pois ao invés de monitorar eventos de ação é necessário monitorar eventos de seleção de lista
- Dessa forma, a classe ouvinte deve implementar a interface **ListSelectionListener**, provendo o método **valueChanged()**

Código

```
1 public class Ouvinte implements ListSelectionListener {  
2     ...  
3  
4     public void valueChanged(ListSelectionEvent e) {  
5         ...  
6     }  
7 }
```

Lista de Elementos

- Quando algum elemento de uma lista é selecionado, o evento **ListSelectionEvent** é gerado
- A interface usada para tratamento de eventos de lista é a **ListSelectionListener**
- Toda classe que implemente essa interface deve prover o método
 - `public void valueChanged(ListSelectionEvent evt);`
- Para se adicionar um ouvinte aos eventos de uma lista usa-se o comando **addListSelectionListener()**

Lista de Elementos

- Quando um item é selecionado, vários eventos são gerados pela lista de seleção
- Para saber quais foram os itens selecionados, o método **getSelectedValues()** deve ser usado
- Esse método retorna um array de objetos contendo todos os itens selecionados

Código

```
1 public class Ouvinte implements ListSelectionListener {
2     ...
3     public class Ouvinte implements ListSelectionListener {
4
5         public void valueChanged(ListSelectionEvent e) {
6             JList source = (JList)e.getSource();
7             Object[] values = source.getSelectedValues();
8
9             for(int i=0; i < values.length; i++) {
10                 System.out.println(values[i]);
11             }
12         }
13     }
14 }
```

Lista de Elementos

- Quando um elemento de uma lista é selecionado, dois eventos mais importantes ocorrem: a seleção de um novo elemento; e o cancelamento da seleção do elemento anterior
- Para saber se o evento então é a seleção final, o método **getValueIsAdjusting()** deve ser usado

Código

```
1 public class Ouvinte implements ListSelectionListener {
2     ...
3     public class Ouvinte implements ListSelectionListener {
4
5         public void valueChanged(ListSelectionEvent e) {
6             JList source = (JList)e.getSource();
7             Object[] values = source.getSelectedValues();
8
9             if(!e.getValueIsAdjusting()) {
10                 for(int i=0; i < values.length; i++) {
11                     System.out.println(values[i]);
12                 }
13             }
14         }
15     }
16 }
```

Lista de Elementos

- A classe **JList** não oferece métodos para manipular elementos em uma lista; na verdade a **JList** é somente a parte gráfica de uma lista, a mesma não sabe nada sobre os dados
- Para controlar os dados que estão presentes em uma lista, uma classe deve ser criada que implemente a interface **ListModel**

Lista de Elementos

- Como esse modelo é difícil de ser aplicado, para nós o que interessa é a adição e remoção de elementos de uma lista, podemos usar um modelo particular, o **DefaultListModel**, e associá-lo com a lista

```
1 DefaultListModel modelo = new DefaultListModel();
2 JList wordList= new JList(modelo);
3 modelo.addElement("primeiro");
4 modelo.addElement("segundo");
5 modelo.addElement("terceiro");
6 modelo.removeElement("primeiro");
```

Outros Elementos Gráficos

- Para maiores informações sobre outros elementos gráficos consulte o livro “Core Java 2 Volume I - Fundamentos”, páginas 391-412

Sumário

- 1 Conceitos Introdutórios
- 2 Gerenciamento de Layout
- 3 Composição de Telas
- 4 Componentes Gráficos
- 5 Gerenciamento de Layout Sofisticado**
- 6 Criando Menus
- 7 Caixas de Diálogo

Gerenciamento de Layout Sofisticado

- Embora os gerenciadores de layout apresentados até agora serem importantes, os mesmos não são suficientes para aplicações mais complexas
- Aqui será apresentado outros gerenciadores padrão de layout, e como os mesmos permitem maior controle sobre a aparência de um aplicativo

Layout de Grade

- O layout de grade (**GridLayout**) é útil para organizar os componentes em uma grade, de forma parecida com as linhas e colunas de uma planilha. Contudo, todas as linhas e colunas da grade têm tamanho idêntico

Layout de Grade

- No construtor do layout de grade é especificado quantas linhas e colunas se quer

```
1 painel.setLayout(new GridLayout(5,4));
```

- É possível também se especificar os espaçamentos vertical e horizontal desejados

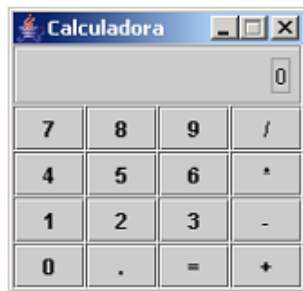
```
1 painel.setLayout(new GridLayout(5,4,3,3));
```

Layout de Grade

- Adicionam-se componentes começando com a primeira entrada da primeira linha, depois a segunda entrada da primeira linha e assim sucessivamente

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

Layout de Grade



Gerenciador GridBagLayout

- O gerenciador de layout **GriBagLayout** é a base para todos os outros gerenciadores de layout do *Swing*
- Nesse gerenciador, as linhas e as colunas podem ter tamanhos variáveis, e podem-se unir elementos adjacentes para abrir espaço para componentes maiores
- Além disso, os componentes não precisam preencher toda área da célula
- Problema: a utilização desse gerenciador pode envolver muita complexidade

Configurando o GridBagLayout

- O primeiro passo é definir que o contêiner usará esse layout

```
1 painelTexto.setLayout(new GridBagLayout());
```

- O segundo passo para se usar esse layout é criar uma variável que irá cuidar da diagramação desse contêiner

```
1 GridBagConstraints restricoes = new GridBagConstraints();
```

Configurando o GridBagLayout

- Após isso, os valores dessa variável de diagramação devem ser configurados para cada elemento

```
1  restricoes.weigthx=100;  
2  restricoes.weigthy=100;  
3  ...
```

- Por fim, adicione os componentes usando essas restrições

```
1  painelTexto.add(componente, restricoes);
```


Configurando o GridBagLayout

- As configurações podem ser
 - Os valores **gridx** e **gridy** especificam a posição (coluna e linha) do canto superior esquerdo do componente a ser adicionado
 - Os valores **gridwidth** e **gridheight** determinam quantas colunas e quantas linhas o componente ocupa

Configurando o GridBagLayout

- Possível configuração para um elemento que está na linha 1 e coluna 0 e ocupa 1 linha e 2 colunas

```
1  restricoes.gridx = 0; //posicao coluna
2  restricoes.gridy = 1; //posicao linha
3  restricoes.gridwidth = 2; //quantas colunas ocupa
4  restricoes.gridheight = 1; //quantas linhas ocupa
```

Configurando o GridBagLayout

- Os campos **weightx** e **weighty** especificam que proporção do espaço de folga deve ser reservada para cada área se o contêiner exceder seu tamanho preferencial
- Esses campos sempre precisam ser definidos para cada área de um **GridBagLayout**
- Se o peso for 0 (zero), então essa área nunca vai ser ampliada ou reduzida além do seu tamanho inicial nessa direção

Configurando o GridBagLayout

```
1 restricoes.weightx = 0; //ampliacao na horizontal  
2 restricoes.weighty = 0; //ampliacao na vertical
```

The screenshot shows a Java Swing window titled "Cadastrar Cliente". Inside the window, there is a section titled "Dados Cliente" which contains a form with the following fields:

- Login
- Nome
- Rua
- Complemento
- Bairro
- Cidade
- Estado
- Cep

A "Confirma" button is located at the bottom right of the window.

Configurando o GridBagLayout

```
1  restricoes.weightx = 100; //ampliacao na horizontal  
2  restricoes.weighty = 100; //ampliacao na vertical
```

The screenshot shows a Java Swing window titled "Cadastrar Cliente". Inside the window, there is a section titled "Dados Cliente" containing several text input fields. The fields are arranged in a grid-like fashion. The labels and their corresponding fields are: "Login" (one field), "Nome" (one field), "Rua" (one field), "Complemento" (one field), "Bairro" (one field), "Cidade" (one field), "Estado" (one field), and "Cep" (one field). At the bottom right of the window, there is a button labeled "Confirma".

Configurando o GridBagLayout

- Se não se quiser que um componente se estenda e preencha toda área, é necessário especificar o campo **fill** com
 - GridBagConstraints.NONE
 - GridBagConstraints.HORIZONTAL
 - GridBagConstraints.VERTICAL
 - GridBagConstraints.BOTH

Configurando o GridBagLayout

- Se o componente não preencher toda área, pode-se especificar onde, na área, deseja-se colocar o mesmo. Para isso usa-se o campo `anchor`
- Esse campo pode receber os seguintes valores
 - `GridBagConstraints.CENTER`
 - `GridBagConstraints.NORTH`
 - `GridBagConstraints.NORTHEAST`
 - `GridBagConstraints.EAST`
 - ...

Configurando o GridBagLayout

```
1 restricoes.anchor = GridBagConstraints.CENTER;  
2 restricoes.fill = GridBagConstraints.NONE;
```

The screenshot shows a Java Swing window titled "Cadastrar Cliente" (Register Client). The window contains a form with the following fields and labels:

- Dados Cliente** (Client Data) - Section header
- Login** - Text input field
- Nome** - Text input field
- Rua** - Text input field
- Complemento** - Text input field
- Bairro** - Text input field
- Cidade** - Text input field
- Estado** - Text input field
- Cep** - Text input field
- Confirma** - Button

Configurando o GridBagLayout

```
1 restricoes.anchor = GridBagConstraints.WEST;  
2 restricoes.fill = GridBagConstraints.HORIZONTAL;
```

The screenshot shows a Java Swing window titled "Cadastrar Cliente". The window contains a form with the following fields and labels:

- Login**: A single-line text input field.
- Nome**: A single-line text input field.
- Rua**: A single-line text input field.
- Complemento**: A single-line text input field.
- Bairro**: A single-line text input field.
- Cidade**: A single-line text input field.
- Estado**: A single-line text input field.
- Cep**: A single-line text input field.

At the bottom right of the form is a button labeled "Confirma".

Exemplo

The image shows a Java Swing window titled "Cadastrar Cliente" (Register Client). The window has a standard title bar with minimize, maximize, and close buttons. The main content area is titled "Dados Cliente" (Client Data) and contains a form with the following fields:

- Login**: A single-line text input field.
- Nome**: A single-line text input field.
- Rua**: A single-line text input field.
- Complemento**: A single-line text input field.
- Barro**: A single-line text input field.
- Cidade**: A single-line text input field.
- Estado**: A single-line text input field.
- Cep**: A single-line text input field.

At the bottom right of the window, there is a button labeled "Confirma" (Confirm).

Exemplo

The screenshot shows a Java Swing window titled "Cadastrar Cliente" with a standard Mac OS X-style title bar (red, yellow, and green buttons). Below the title bar is a label "Dados Cliente". The form contains several labels and text input fields. A red grid is overlaid on the form, highlighting the input areas. The labels and their corresponding input fields are:

- Login
- Nome
- Rua
- Barro
- Estado
- Complemento
- Cidade
- Cep

A "Confirma" button is located at the bottom right of the window.

Outros Gerenciadores de Layout

- Java oferece outros gerenciadores de layout
 - **BoxLayout**: organiza todos os elementos em uma única linha ou coluna
- É possível também não usar nenhum gerenciador de layout, mas isso não é uma boa ideia de construção de interfaces em Java

Ordem de Travessia

- É possível alterar a ordem de travessia (tecla <TAB>) usando alguns métodos presentes no *Swing*. Para maiores informações consulte: “Core Java 2 Volume 1 - Fundamentos” (pág 435-436)

Sumário

- 1 Conceitos Introdutórios
- 2 Gerenciamento de Layout
- 3 Composição de Telas
- 4 Componentes Gráficos
- 5 Gerenciamento de Layout Sofisticado
- 6 Criando Menus**
- 7 Caixas de Diálogo

Menus

- O *Swing* suporta a criação de barras de menu. Para esse tipo de elemento não é necessário um gerenciador de layout
- Uma barra de menu no topo da janela contém os nomes dos menus suspensos
- Clicar em um nome abre o menu contendo os itens de menu e os submenus

Como Elaborar Menus

- Para se elaborar menus, primeiro a barra de menu deve ser criada

```
1 JMenuBar barraMenu = new JMenuBar();
```

- Após isso, devem ser criados objetos para cada menu

```
1 JMenu cadastrarMenu = new JMenu("Cadastrar");
```


Como Elaborar Menus

- Depois os itens de menu devem ser criados

```
1 JMenuItem cadastrarCliente = new JMenuItem("Cadastrar Cliente");
```

- Para depois serem adicionados aos objetos menu

```
1 cadastrarMenu.add(cadastrarCliente);
```

Como Elaborar Menus

- Por fim, adiciona-se menus à barra de menus

```
1 barraMenu.add(cadastrarMenu);
```

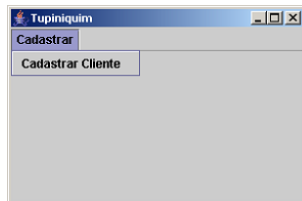
- E coloca-se a barra de menu na parte superior do quadro

```
1 frame.setJMenuBar(barraMenu);
```

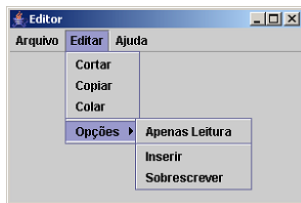
Código

```
1 public class GUITupiniquim extends JFrame {
2     private JMenuBar barraMenu = new JMenuBar();
3     private JMenu cadastrarMenu = new JMenu("Cadastrar");
4     private JMenuItem cadastrarCliente = new JMenuItem("Cadastrar ↵
      Cliente");
5
6     public GUITupiniquim() {
7         ...
8         cadastrarMenu.add(cadastrarCliente);
9         barraMenu.add(cadastrarMenu);
10        this.setJMenuBar(barraMenu);
11    }
12 }
```

Resultado



Um Menu mais Elaborado



Um Menu mais Elaborado: Código

```
1 JMenu arquivo = new JMenu("Arquivo");
2 JMenu editar = new JMenu("Editar");
3 JMenuItem cortar = new JMenuItem("Cortar");
4 JMenuItem copiar = new JMenuItem("Copiar");
5 JMenuItem colar = new JMenuItem("Colar");
6 editar.add(cortar);
7 editar.add(copiar);
8 editar.add(colar);
9 editar.addSeparator();
10 JMenu opcoesMenu = new JMenu("Opções");
11 JMenuItem apenasLeitura = new JMenuItem("Apenas Leitura");
12 JMenuItem inserir = new JMenuItem("Inserir");
13 JMenuItem sobrescrever = new JMenuItem("Sobrescrever");
14 opcoesMenu.add(apenasLeitura);
15 opcoesMenu.addSeparator();
16 opcoesMenu.add(inserir);
17 opcoesMenu.add(sobrescrever);
18 editar.add(opcoesMenu);
19 JMenu ajuda = new JMenu("Ajuda");
20 JMenuBar barraMenu = new JMenuBar();
21 barraMenu.add(arquivo);
22 barraMenu.add(editar);
23 barraMenu.add(ajuda);
24 this.setJMenuBar(barraMenu);
```

Como Responder a Eventos de Menu

- O tratamento de eventos do menu é idêntico ao tratamento de evento dos botões, portanto não precisa de maiores explicações
- Os menus podem conter outros elementos como ícones, caixas de seleção, botões de rádio, etc. Para maiores informações consulte: “Core Java 2 Volume 1 - Fundamentos” (pág 441-444)

Mnemônicos e Teclas de Atalho de Teclado

- Para se especificar o mnemônico (tecla de atalho) que ativará algum item de menu, coloque a letra desse mnemônico no construtor do item menu

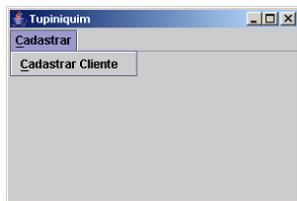
```
1 JMenuItem cadastrarCliente = new JMenuItem("Cadastrar Cliente", 'C');
```


Mnemônicos e Teclas de Atalho de Teclado

- Porém, somente podem ser associados via construtor mnemônicos para itens de menu
- Para associar mnemônicos a menus, use o seguinte método

```
1  cadastrarMenu.setMnemonic('C');
```

Resultado

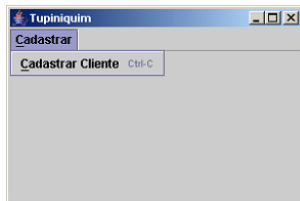


Mnemônicos e Teclas de Atalho de Teclado

- Diferente dos mnemônicos, as teclas de atalho são atalhos via teclado para selecionar itens de menu sem ser necessário abrir o menu
- Para isso faz-se

```
1  cadastrarCliente.setAccelerator(  
2      KeyStroke.getKeyStroke(KeyEvent.VK_C,  
3      InputEvent.CTRL_MASK));
```

Resultado



Sumário

- 1 Conceitos Introdutórios
- 2 Gerenciamento de Layout
- 3 Composição de Telas
- 4 Componentes Gráficos
- 5 Gerenciamento de Layout Sofisticado
- 6 Criando Menus
- 7 Caixas de Diálogo**

Caixas de Diálogo

- Além das janelas, existem as caixas de diálogo que servem para retornar ou para obter informações do usuário
- Existem caixas de diálogo do tipo modal (não permite que o usuário interaja com as outras janelas do aplicativo enquanto a mesma não for fechada) e não-modal

Caixas de Diálogo

- O **JOptionPane** tem um conjunto de diálogos predefinido para pedir ao usuário uma informação determinada
- Esses tipos podem ser chamados por meio dos métodos estáticos
 - `showMessageDialog` - espera um OK
 - `showConfirmDialog` - espera uma confirmação (OK/Cancel)
 - `showOptionDialog` - obtém uma opção do usuário dentre várias
 - `showInputDialog` - obtém uma linha digitada pelo usuário

Caixas de Diálogo

- Em um diálogo temos três principais elementos
 - Um ícone
 - Uma mensagem
 - E um ou mais botões de opção
- O ícone depende do tipo de mensagem
 - `ERROR_MESSAGE`
 - `INFORMATION_MESSAGE`
 - `WARNING_MESSAGE`
 - `QUESTION_MESSAGE`
 - `PLAIN_MESSAGE`

Caixas de Diálogo

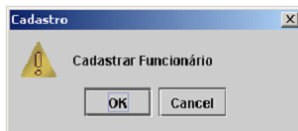
- O valor de retorno da criação de uma caixa de diálogo pode ser
 - `showMessageDialog` : nenhum
 - `showConfirmDialog` : um inteiro representando a opção escolhida
 - `showOptionDialog` : um inteiro representando a opção escolhida
 - `showInputDialog` : a string quer o usuário fornecer ou selecionar

Caixas de Diálogo

- No diálogo de confirmação, os botões que aparecem podem seguir as seguintes opções
 - OK_OPTION
 - CANCEL_OPTION
 - YES_OPTION
 - NO_OPTION
 - CLOSED_OPTION

Caixas de Diálogo

```
1 JOptionPane.showConfirmDialog(GUITupiniquim.this, "Cadastrar ↵  
  Funcionário", "Cadastro", JOptionPane.OK_CANCEL_OPTION, ↵  
  JOptionPane.WARNING_MESSAGE);
```



Caixas de Diálogo

- Caso seja necessário se criar diálogos mais refinados do que os oferecidos pelo **JOptionPane**, pode-se estender a classe **JDialog**, e criar um diálogo da mesma forma que se cria um frame com o **JFrame**
- Além disso, existem diálogos pré-existentes que servem para tarefas bem definidas, como: **JFileChooser** e **JColorChooser**