Interfaces Gráficas

POO

Prof. Márcio Delamaro

GUI

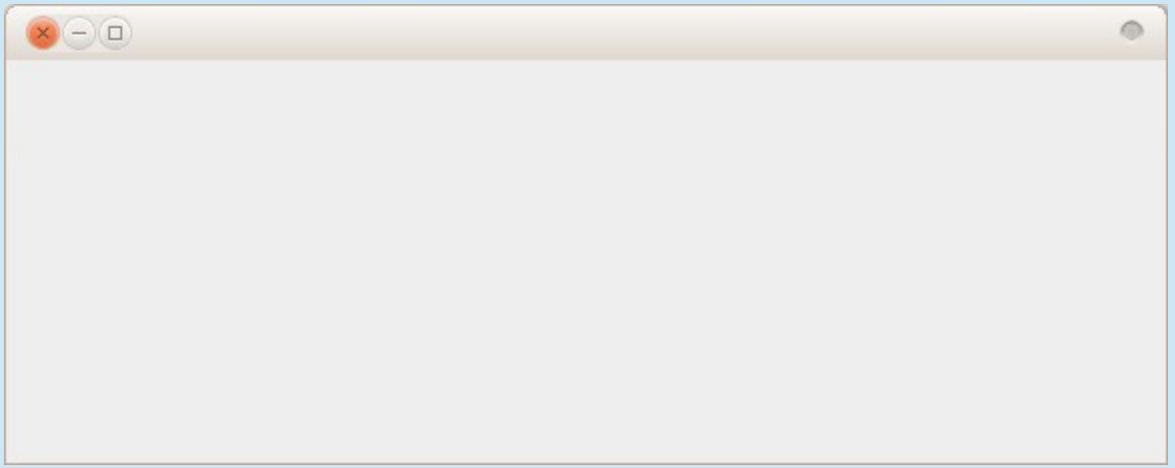
- Interface Gráfica do Usuário
- Em Java usamos o pacote Swing
- javax.swing
- Possui um conjunto de classes que permitem a construção de interfaces
- Vamos ver os conceitos básicos

Componente básico

- Toda GUI tem uma janela

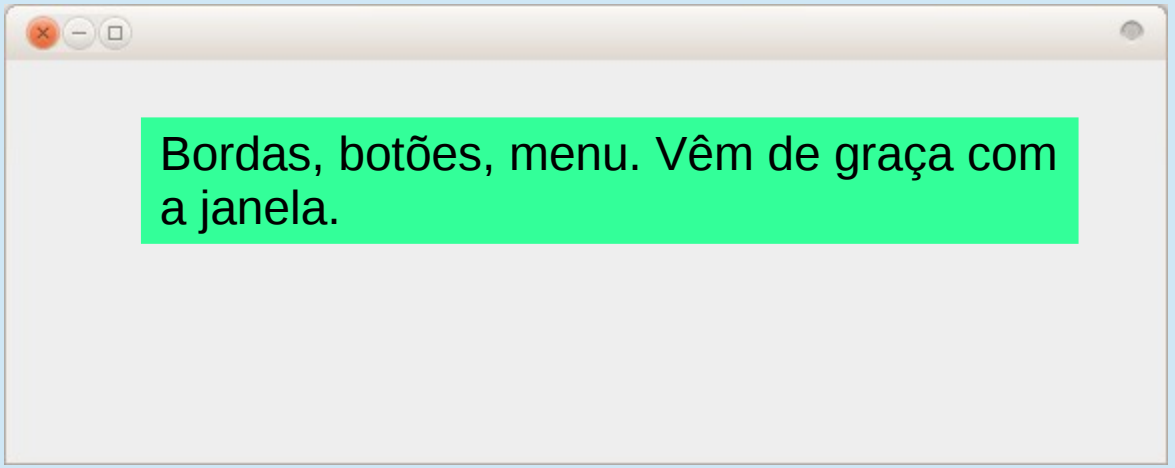
Componente básico

- Toda GUI tem uma janela



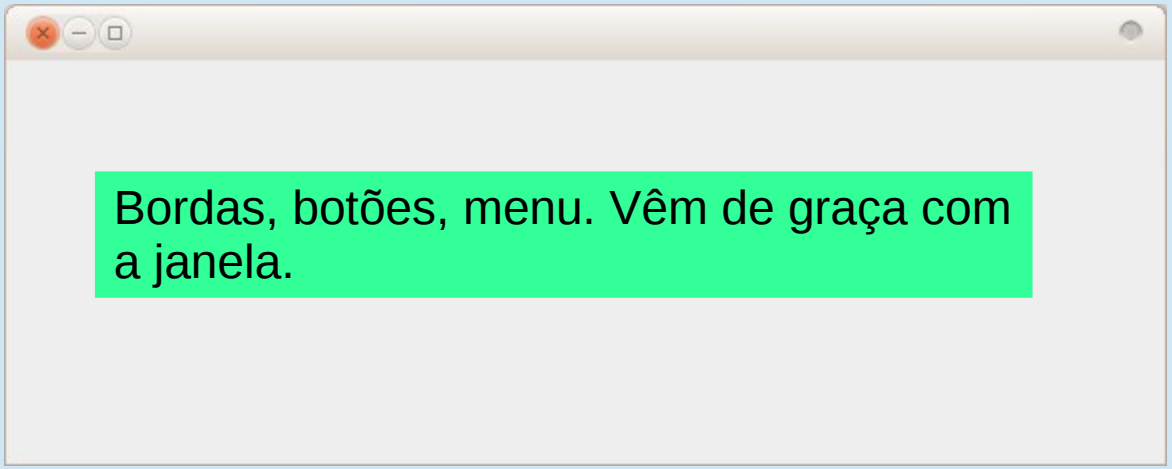
Componente básico

- Toda GUI tem uma janela



Componente básico

- Toda GUI tem uma janela



- Para que essa janela apareça, basta:
  - criar um objeto
  - mandar ele aparecer

javax.swing.JFrame

```
public class HelloGUI  {

public static void main(String[] args) {
    JFrame frame = new JFrame();
    frame.setVisible(true);
}

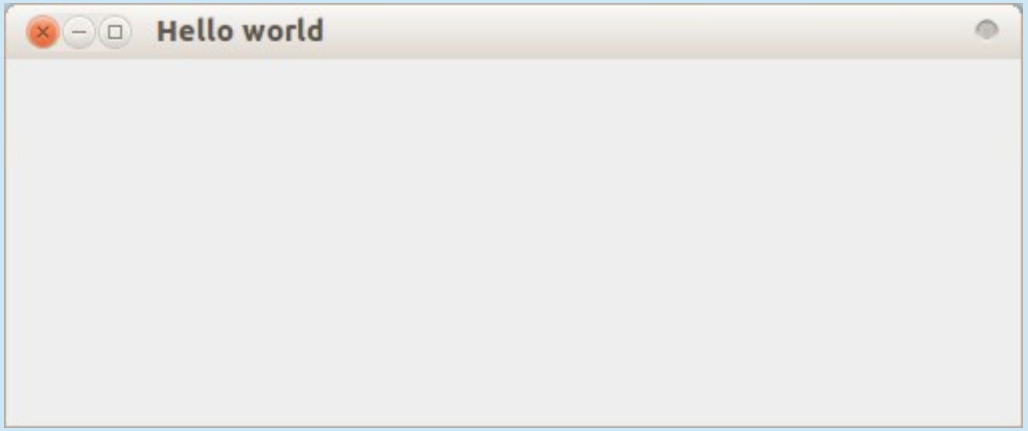
}
```

javax.swing.JFrame

```
public class HelloGUI  {

public static void main(String[] args) {
    JFrame frame = new JFrame("Hello world");
    frame.setVisible(true);
}

}
```

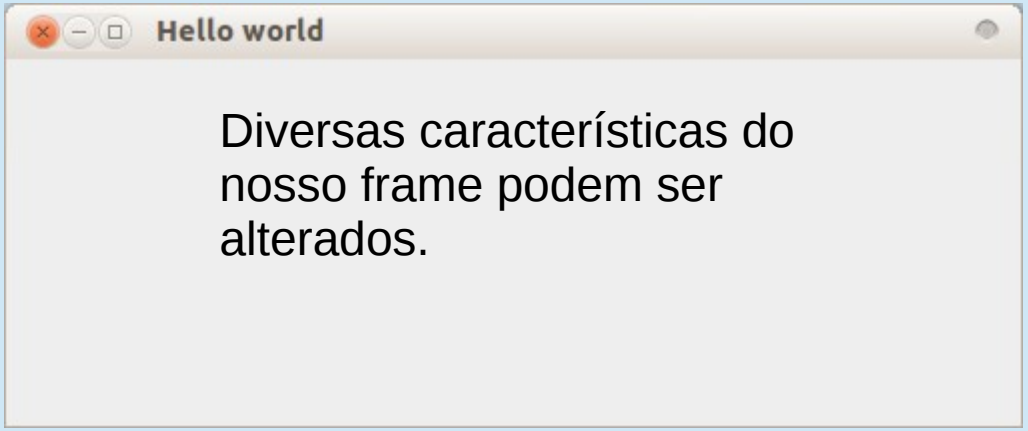


javax.swing.JFrame

```
public class HelloGUI  {


public static void main(String[] args) {
    JFrame frame = new JFrame("Hello world");
    frame.setVisible(true);
}


}
```



Mas antes disso...

- Não usamos diretamenta a classe Jframe
- Costumamos criar a nossa própria classe
- Que estende a classe Jframe

Nosso próprio frame

```
public class HelloGUI extends JFrame {


public static void main(String[] args) {
    HelloGUI frame = new HelloGUI();
    frame.setVisible(true);
}
}
```

Nosso próprio frame

```
public class HelloGUI extends JFrame {

public static void main(String[] args) {
    HelloGUI frame = new HelloGUI();
    frame.setVisible(true);
}
}
```

Nosso próprio frame com título

```
public class HelloGUI extends JFrame {

public HelloGUI()
{
   super("Hello world");
}


public static void main(String[] args) {
    HelloGUI frame = new HelloGUI();
    frame.setVisible(true);
}

}
```

Algumas configurações

```
public static void main(String[] args) {

     HelloGUI frame = new HelloGUI();

     frame.setSize(400, 200);
     frame.setDefaultCloseOperation(
                              JFrame.EXIT_ON_CLOSE);

     frame.setVisible(true);

   }
```

Algumas configurações

```
public static void main(String[] args) {

     HelloGUI frame = new HelloGUI();

     frame.setSize(400, 200);
     frame.setDefaultCloseOperation(
                              JFrame.EXIT_ON_CLOSE);

     frame.setVisible(true);

  }
```

Determina o tamanho da janela

Algumas configurações

```
public static void main(String[] args) {
     HelloGUI frame = new HelloGUI();
     frame.setSize(400, 200);
     frame.setDefaultCloseOperation(
                              JFrame.EXIT_ON_CLOSE);
     frame.setVisible(true);
  }
```

Determina o tamanho da janela

Determina o que fazer ao fechar a janela.

Interface

- Para criar a GUI precisamos dos components
- Vamos adicionar os componentes à Janela
- Ou seja, vamos adicionar objetos “dentro” do nosso frame
- Para cada tipo de componente temos uma classe no swing

JLabel

- Esse componente aparece apenas como uma mensagem (curta) dentro do frame

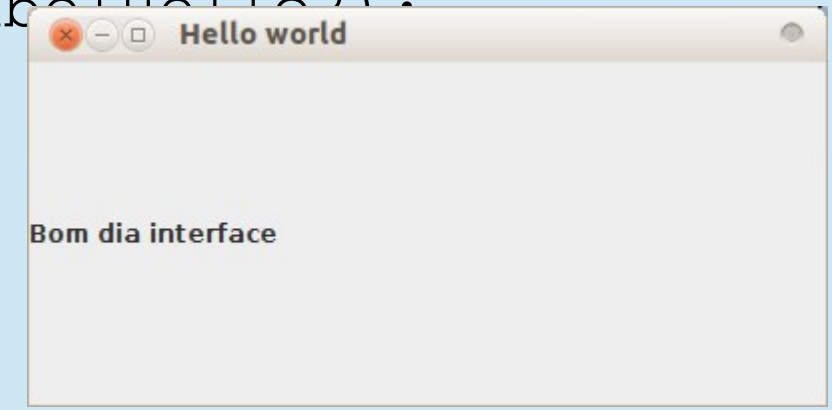


E o que acontece...

```
public HelloGUI()
{
     super("Hello world");
     JLabel labelHello = new JLabel("Hello
interface");
     this.add(labelHello);
     JLabel labelHello2 = new JLabel("Bom dia
interface");
     this.add(labelHello2);
}
```

E o que acontece...

```
public HelloGUI()
{
     super("Hello world");
     JLabel labelHello = new JLabel("Hello
interface");
     this.add(labelHello);
     JLabel labelHello2 = new JLabel("Bom dia
interface");
     this.add(labelHello2);
}
```



Layout

- É preciso determinar a forma como os componentes vão ser inseridos
- Qual vai ser o posicionamento deles dentro da janela
- Isso é feito determinando qual o leiaute da janela

FlowLayout

```
public HelloGUI()
{
  super("Hello world");
  this.setLayout(new FlowLayout());
  this.add(new JLabel("Label 1"));
  this.add(new JLabel("Label variado e bem grande 2"));
  this.add(new JLabel("Label três"));
  this.add(new JLabel("Label 4"));
  this.add(new JLabel("Label hello 5"));
}
```

FlowLayout

```
public HelloGUI()
{
  super("Hello world");
  this.setLayout(new FlowLayout());
  this.add(new JLabel("Label 1"));
  this.add(new JLabel("Label variado e bem grande 2"));
  this.add(new JLabel("Label três"));
  this.add(new JLabel("Label 4"));
  this.add(new JLabel("Label hello 5"));
}
```

FlowLayout

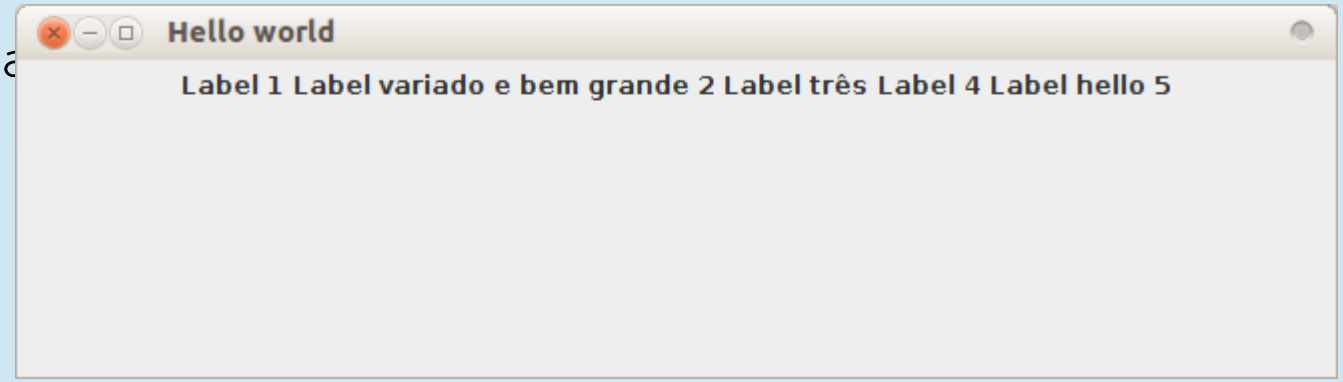
```
public HelloGUI()
{
  super("Hello world");
  this.setLayout(new FlowLayout(
  this.add(new JLabel("Label 1")
  this.add(new JLabel("Label variado e bem grande 2"));
  this.add(new JLabel("Label três"));
  this.add(new JLabel("Label 4"));
  this.add(new JLabel("Label hello 5"));
}
```



FlowLayout

```
public HelloGUI()
{
  super("Hello world");
  this.setLayout(new FlowLayout(
  this.add(new JLabel("Label 1")
  this.add(new JLabel("Label variado e bem grande 2"));
  this.add(new JLabel("Label três"));
  this.add(new JLabel("Label 4"));
  this.a
}
```





FlowLayout

- A flow layout arranges components in a directional flow, much like lines of text in a paragraph.
- The flow direction is determined by the container's componentOrientation property and may be one of two values:
  - ComponentOrientation.LEFT_TO_RIGHT
  - ComponentOrientation.RIGHT_TO_LEFT

FlowLayout

```
public HelloGUI()
{
  super("Hello world");
  this.setLayout(new FlowLayout());
  JPanel jp = (JPanel) this.getContentPane();
  jp.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
  this.add(new JLabel("Label 1"));
  this.add(new JLabel("Label variado e bem grande 2"));
  this.add(new JLabel("Label três"));
  this.add(new JLabel("Label 4"));
  this.add(new JLabel("Label hello 5"));
}
```

FlowLayout

```
public HelloGUI()
{
  super("Hello world");
  this.setLayout(new FlowLayout());
  JPanel jp = (JPanel) this.getContentPane();
  jp.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
  this.add(new JLabel("Label 1"));
  this.add(new JLabel("Label variado e bem grande 2"));
  this.add(new JLabel("Label três"));
  this.add(new JLabel("Label 4"));
  this.add(new JLabel("Label hello 5"));
}
```

FlowLayout

```
public HelloGUI()
{
  super("Hello world");
  this.setLayout(new FlowLayout());
  JPanel jp = (JPanel) this.getContentPane();
  jp.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
  this.add(new JLabel("Label 1"));
  this.add(new JLabel("Label variado e bem grande 2"));
  this.add(new JLabel("Label três"));
  this.add(new JLabel("Label 4"));
  this.add(new JLabel("Label hello 5"));
}
```



JPanel

- Esse objeto é onde os objetos são realmente colocados
- Dentro de um JFrame já existe um Jpanel
- Ao inserir um componente, estamos realmente inserindo no JPanel

No exemplo

```
public HelloGUI()
{
  super("Hello world");
  JPanel jp = (JPanel) this.getContentPane();
  jp.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
  jp.setLayout(new FlowLayout());
  jp.add(new JLabel("Label 1"));
  jp.add(new JLabel("Label variado e bem grande 2"));
  jp.add(new JLabel("Label três"));
  jp.add(new JLabel("Label 4"));
  jp.add(new JLabel("Label hello 5"));
}
```

No exemplo

```
public HelloGUI()
{
  super("Hello world");
  JPanel jp = (JPanel) this.getContentPane();
  jp.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
  jp.setLayout(new FlowLayout());
  jp.add(new JLabel("Label 1"));
  jp.add(new JLabel("Label variado e bem grande 2"));
  jp.add(new JLabel("Label três"));
  jp.add(new JLabel("Label 4"));
  jp.add(new JLabel("Label hello 5"));
}
```

Outros leiautes

- The ***GridLayout*** class is a layout manager that lays out a container's components in a rectangular grid.

Outros leiautes

- The ***GridLayout*** class is a layout manager that lays out a container's components in a rectangular grid.
  - `jp.setLayout(new GridLayout(3,2));`



BorderLayout

- A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center

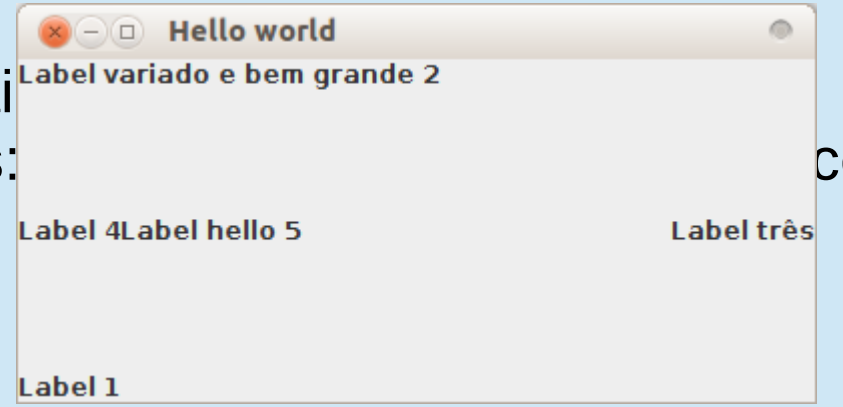
BorderLayout

- A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center

```
public HelloGUI()
{
        super("Hello world");
        JPanel jp = (JPanel) this.getContentPane();
        jp.setLayout(new BorderLayout());
        jp.add(new JLabel("Label 1"), BorderLayout.SOUTH);
        jp.add(new JLabel("Label variado e bem grande 2"), BorderLayout.NORTH);
        jp.add(new JLabel("Label três"), BorderLayout.EAST);
        jp.add(new JLabel("Label 4"), BorderLayout.WEST);
        jp.add(new JLabel("Label hello 5"), BorderLayout.CENTER);
    }
```

BorderLayout

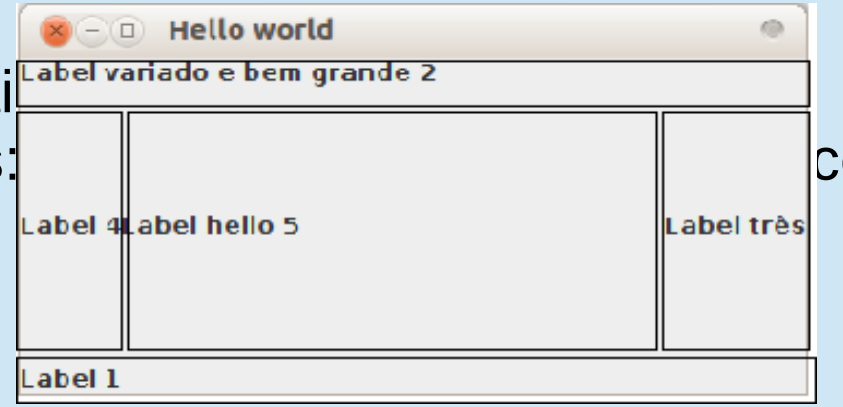
- A border layout lays out a contai
  components to fit in five regions: center




```
public HelloGUI()
{
        super("Hello world");
        JPanel jp = (JPanel) this.getContentPane();
        jp.setLayout(new BorderLayout());
        jp.add(new JLabel("Label 1"), BorderLayout.SOUTH);
        jp.add(new JLabel("Label variado e bem grande 2"), BorderLayout.NORTH);
        jp.add(new JLabel("Label três"), BorderLayout.EAST);
        jp.add(new JLabel("Label 4"), BorderLayout.WEST);
        jp.add(new JLabel("Label hello 5"), BorderLayout.CENTER);
    }
```

BorderLayout

- A border layout lays out a contai
  components to fit in five regions:



center

```
public HelloGUI()
{
        super("Hello world");
        JPanel jp = (JPanel) this.getContentPane();
        jp.setLayout(new BorderLayout());
        jp.add(new JLabel("Label 1"), BorderLayout.SOUTH);
        jp.add(new JLabel("Label variado e bem grande 2"), BorderLayout.NORTH);
        jp.add(new JLabel("Label três"), BorderLayout.EAST);
        jp.add(new JLabel("Label 4"), BorderLayout.WEST);
        jp.add(new JLabel("Label hello 5"), BorderLayout.CENTER);
    }
```

Interação

- Uma vez definidos os componentes da interface é preciso interagir com eles
- Por exemplo, ao clicar em um botão, nosso programa precisa saber que esse evento aconteceu
- Para isso, precisamos registrar um Listener junto ao componente

JButton

```
public Interact()
{
    super("Hello world");
    JPanel jp = (JPanel) this.getContentPane();
    jp.setLayout(new GridLayout(3,1));
    label = new JLabel(WELLCOME);
    nome = new JTextField();
    buttonOK = new JButton("Confirma");
    jp.add(label);
    jp.add(nome);
    jp.add(buttonOK);
}
```

JButton

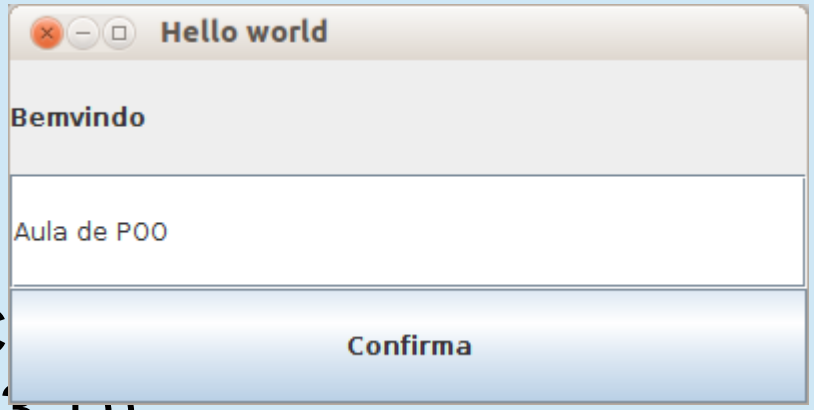
```
public Interact()
{
    super("Hello world");
    JPanel jp = (JPanel) this.getContentPane();
    jp.setLayout(new GridLayout(3,1));
    label = new JLabel(WELLCOME);
    nome = new JTextField();
    buttonOK = new JButton("Confirma");
    jp.add(label);
    jp.add(nome);
    jp.add(buttonOK);
}
```



3 components alinhados verticalmente

JButton

```
public Interact()
{
    super("Hello world");
    JPanel jp = (JPanel) this.getC
    jp.setLayout(new GridLayout(3,1));
    label = new JLabel(WELLCOME);
    nome = new JTextField();
    buttonOK = new JButton("Confirma");
    jp.add(label);
    jp.add(nome);
    jp.add(buttonOK);
}
```





3 components alinhados verticalmente

JButton Listener

- Nada acontece ao clicar o botão
- Isso porque o nosso programa não foi preparado para tratar o evento do botão
- Vamos usar o próprio objeto para fazer esse tramamento
- `buttonOK.addActionListener(this);`

ActionListener

- Para que a nossa classe possa tratar o evento ela precisa implementr essa interface
- Ou seja,
- public class Interact extends JFrame
  implements ActionListener {
- E implementar

```
@Override
public void actionPerformed(ActionEvent e) {


}
```

Em resumo

- Com a chamada
- `buttonOK.addActionListener(this);`
- Estamos dizendo que o método *actionPerformed*, do corrente objeto vai ser chamada ao se clicar no botão

actionPerformed

```
@Override
public void actionPerformed(ActionEvent e) {
        String s = nome.getText();
        s = WELLCOME + " " + s;
        label.setText(s);
}
```

actionPerformed

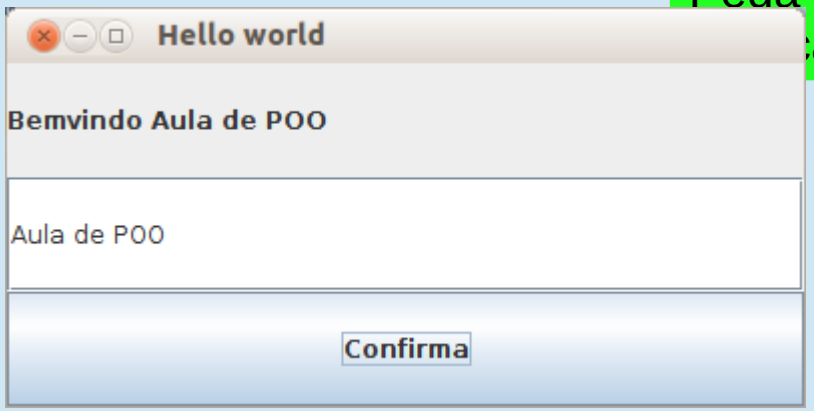
```
@Override
public void actionPerformed(ActionEvent e) {
        String s = nome.getText();
        s = WELLCOME + " " + s;
        label.setText(s);
}
```

Pega o nome digitado e modifica o label.
Coloca “Bemvindo “ + nome digitado.

actionPerformed

```
@Override
public void actionPerformed(ActionEvent e) {
        String s = nome.getText();
        s = WELLCOME + " " + s;
        label.setText(s);
}
```

Pega o nome digitado e modifica o label.
ca “Bemvindo “ + nome digitado.



Vários componentes para interagir

- Na mesma interface podemos ter vários componetes que geram ações
- Podemos ter um objeto listener para cada objeto
- Ou podemos ter um único listener para todos eles
- Nesse caso precisamos de uma forma de saber quem gerou o evento

Dois botões

```
buttonOK = new JButton("Confirma");
buttonLimpa = new JButton("Limpa");
buttonOK.addActionListener(this);
buttonLimpa.addActionListener(this);
```

Aproveitando...

```
jp.add(label);
jp.add(nome);
JPanel jp2 = new JPanel(new FlowLayout());
jp2.add(buttonOK);
jp2.add(buttonLimpa);
jp.add(jp2);
```

Aproveitando...

```
jp.add(label);
jp.add(nome);
JPanel jp2 = new JPanel(new FlowLayout());
jp2.add(buttonOK);
jp2.add(buttonLimpa);
jp.add(jp2);
```

Cria um novo painel para inserir os botões.

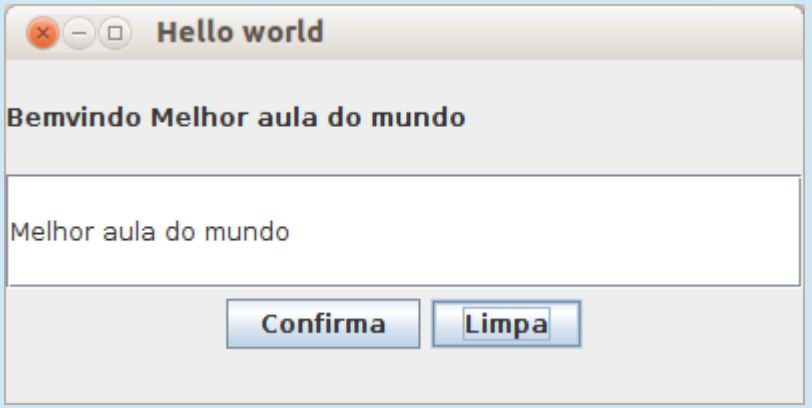
Insere esse painel na 3a posição do grid

Aproveitando...

```
jp.add(label);
jp.add(nome);
JPanel jp2 = new JPanel(new FlowLayout());
jp2.add(buttonOK);
jp2.add(buttonLimpa);
jp.add(jp2);
```

Cria um novo painel para inserir os botões.

Insere esse painel na 3a posição do grid



Mas não funciona

- O problema é que os dois botões estão sendo tratados pelo mesmo listenar
- Ou seja, estão fazendo a mesma coisa
- Vamos então identificar cada botão

```
buttonOK = new JButton("Confirma");
buttonLimpa = new JButton("Limpa");
buttonOK.addActionListener(this);
buttonOK.setActionCommand("confirma");
buttonLimpa.addActionListener(this);
buttonLimpa.setActionCommand("limpa");
```

Mas não funciona

- O problema é que os dois botões estão sendo tratados pelo mesmo listenar
- Ou seja, estão fazendo a mesma coisa
- Vamos então identificar cada botão

```
buttonOK = new JButton("Confirma");
buttonLimpa = new JButton("Limpa");
buttonOK.addActionListener(this);
buttonOK.setActionCommand("confirma");
buttonLimpa.addActionListener(this);
buttonLimpa.setActionCommand("limpa");
```

Simplesmente associamos um “nome” com o botão

E na hora de tratar

```
@Override
public void actionPerformed(ActionEvent e) {
  if ("confirma".equals(e.getActionCommand())) {
  String s = nome.getText();
  s = WELLCOME + " " + s;
  label.setText(s);
  }
  else
  if ("limpa".equals(e.getActionCommand())) {
  nome.setText("");
  String s = WELLCOME;
  label.setText(s);
  }
}
```

E na hora de tratar

```
@Override
public void actionPerformed(ActionEvent e) {
  if ("confirma".equals(e.getActionCommand())) {
  String s = nome.getText();
  s = WELLCOME + " " + s;
  label.setText(s);
  }
  else
  if ("limpa".equals(e.getActionCommand())) {
  nome.setText("");
  String s = WELLCOME;
  label.setText(s);
  }
}
```

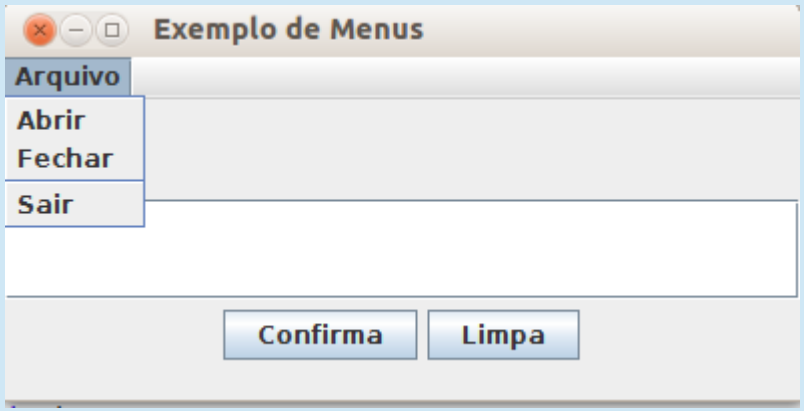
Vai retornar aquele string associado ao botão.

Menus

- Os menus são componentes que não precisam ser posicionados
- Eles ficam pendurados na barra de menus da janela (do frame)
- Uma barra de menu pode ter vários menus

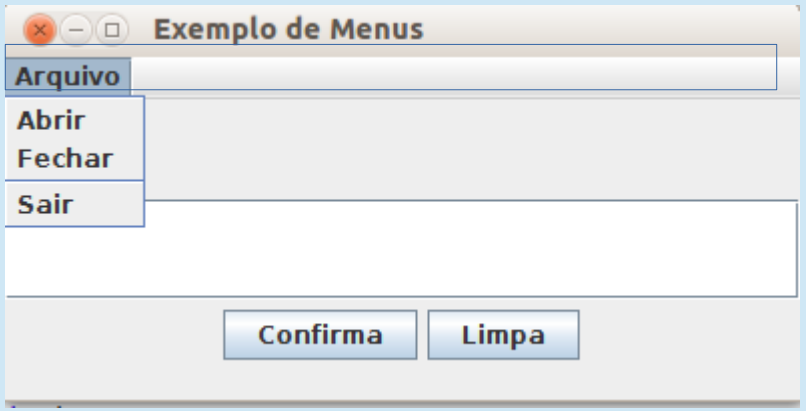
Menus

- Os menus são componentes que não precisam ser posicionados
- Eles ficam pendurados na barra de menus da janela (do frame)
- Uma barra de menu pode ter vários menus



Menus

- Os menus são componentes que não precisam ser posicionados
- Eles ficam pendurados na barra de menus da janela (do frame)
- Uma barra de menu pode ter vários menus

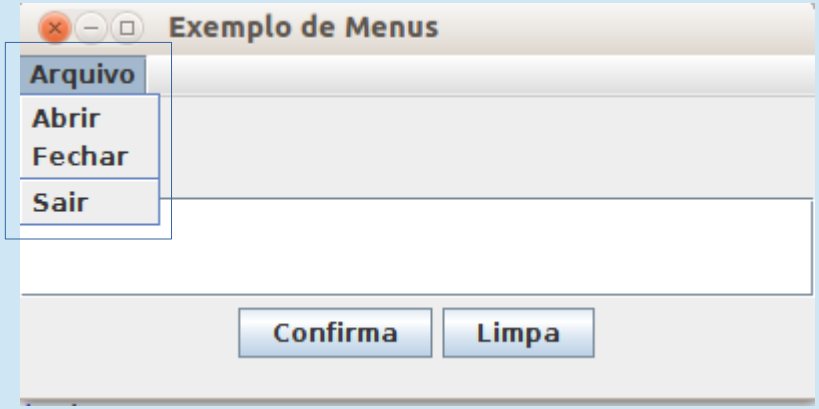


Barra de menus (JMenuBar)

Menus

- Os menus são componentes que não precisam ser posicionados
- Eles ficam pendurados na barra de menus da janela (do frame)
- Uma barra de menu pode ter vários menus

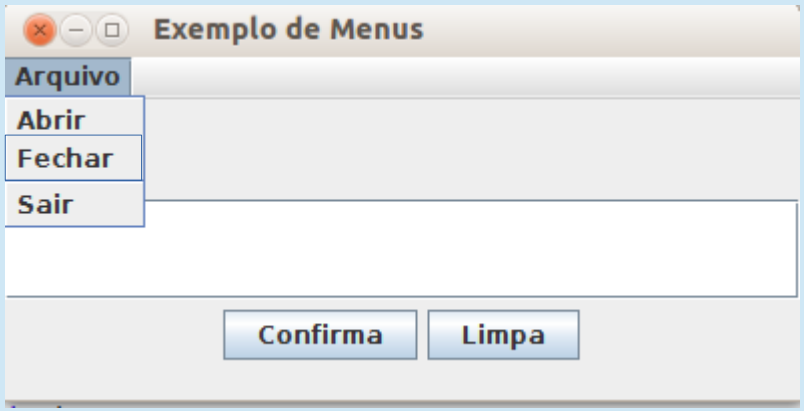
Menu
(JMenu)



Menus

- Os menus são componentes que não precisam ser posicionados
- Eles ficam pendurados na barra de menus da janela (do frame)
- Uma barra de menu pode ter vários menus

Menu item
(JMenuItem)



Como funciona

- Dentro de uma barra de menus (JMenuBar) nós inserimos os menus (JMenu)
- Dentro de um menu (JMenu) nós inserimos os itens de menu (JMenuItem)
- Cada item de menu, ao ser selecionado, gera um evento que pode ser tratado pelo ActionListener

Criar a barra

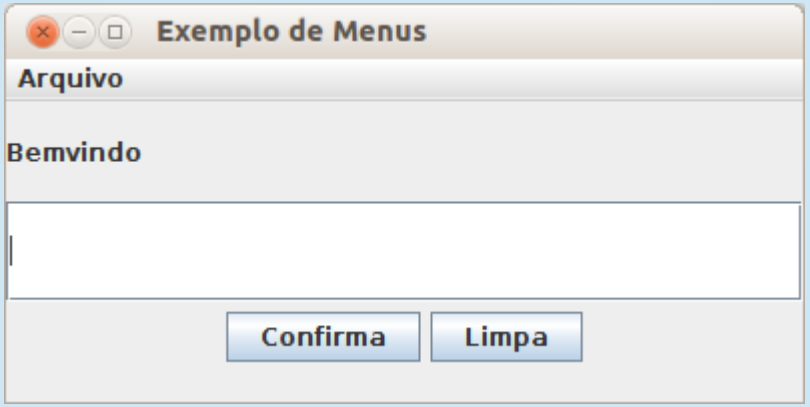
```
JMenuBar menuBar = new JMenuBar();
this.setJMenuBar(menuBar);
JMenu jb = new JMenu("Arquivo");
menuBar.add(jb);
```

- Insere a barra no frame
- Cria também o menu e insere na barra
- Menu está vazio

Criar a barra

```
JMenuBar menuBar = new JmenuBar();
this.setJMenuBar(menuBar);
JMenu jb = new JMenu("Arquivo");
menuBar.add(jb);
```

- Insere a barra no frame
- Cria também o menu e insere na barra
- Menu está vazio



Criar os itens

```
JMenuItem mi = new JMenuItem("Abrir");
mi.setActionCommand("abrir_arquivo");
mi.addActionListener(this);
jb.add(mi);
```

Criar os itens

```
JMenuItem mi = new JMenuItem("Abrir");
mi.setActionCommand("abrir_arquivo");
mi.addActionListener(this);
jb.add(mi);
```

Cria o item de menu “Abrir”.
Adiciona o listener e identifica o evento.
Adiciona o item ao menu que havia sido criado.

Criar os outros itens

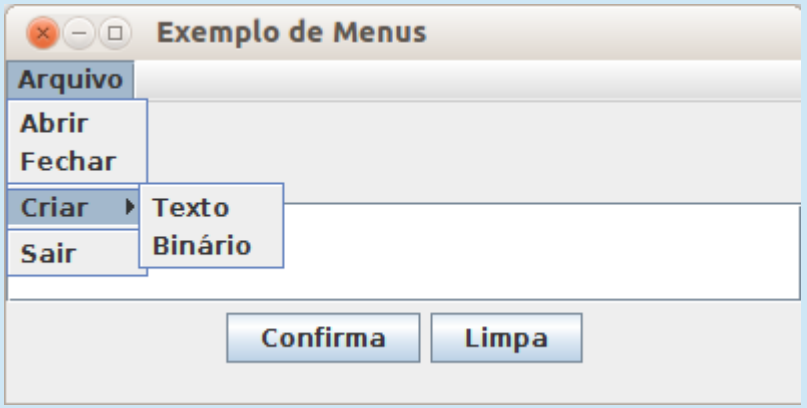
```
mi = new JMenuItem("Fechar");
mi.setActionCommand("fechar_arquivo");
mi.addActionListener(this);
jb.add(mi);
jb.addSeparator();
mi = new JMenuItem("Sair");
mi.setActionCommand("sair_programa");
mi.addActionListener(this);
jb.add(mi);
```

Criar os outros itens

```
mi = new JMenuItem("Fechar");
mi.setActionCommand("fechar_arquivo");
mi.addActionListener(this);
jb.add(mi);
jb.addSeparator();
mi = new JMenuItem("Sair");
mi.setActionCommand("sair_programa");
mi.addActionListener(this);
jb.add(mi);
```

Submenus

- Em vez de itens de menus, podemos adicionar um menu dentro de outro
- Isso faz com que criemos um sub menu



Implementando

```
JMenu criar = new JMenu("Criar");
jb.add(criar);
```

Criado um novo menu e inserido no que já existia

Implementando

```
JMenu criar = new JMenu("Criar");
jb.add(criar);
mi = new JMenuItem("Texto");
mi.setActionCommand("criar_texto");
mi.addActionListener(this);
criar.add(mi);
mi = new JMenuItem("Binário");
mi.setActionCommand("criar_binario");
mi.addActionListener(this);
criar.add(mi);
```

Implementando

```
JMenu criar = new JMenu("Criar");
jb.add(criar);
mi = new JMenuItem("Texto");
mi.setActionCommand("criar_texto");
mi.addActionListener(this);
criar.add(mi);
mi = new JMenuItem("Binário");
mi.setActionCommand("criar_binario");
mi.addActionListener(this);
criar.add(mi);
```

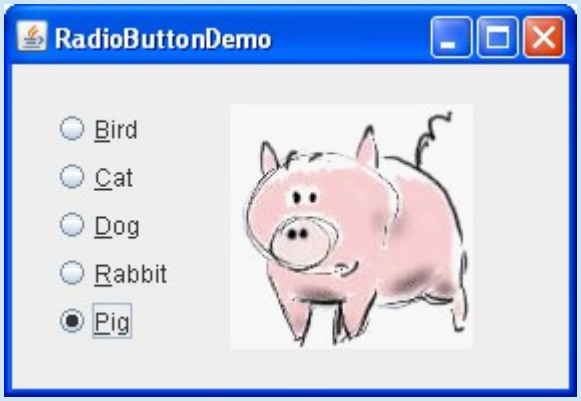
Criados e adicionados os itens de menu

Outros componentes

- Checkbox

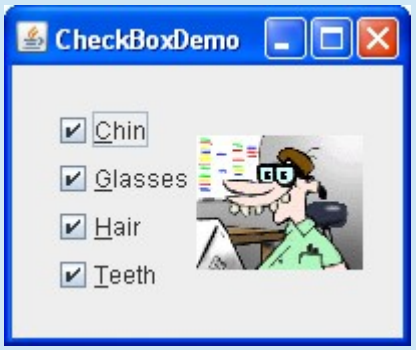


- Radio Button

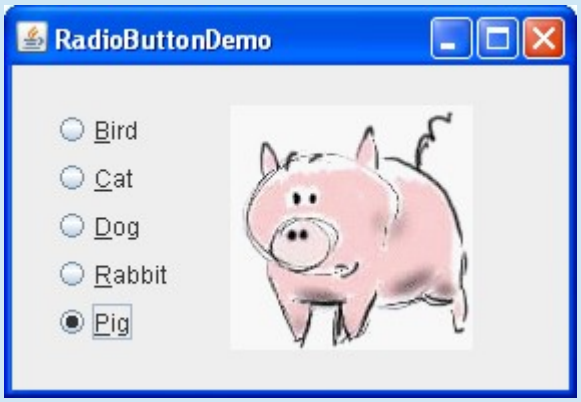


Outros componentes

- Checkbox



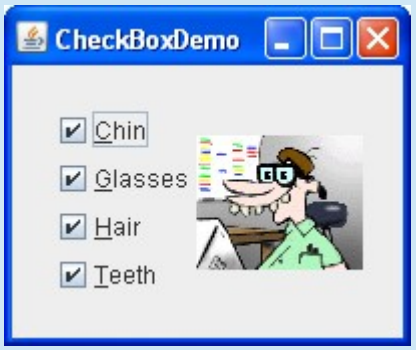
- Radio Button



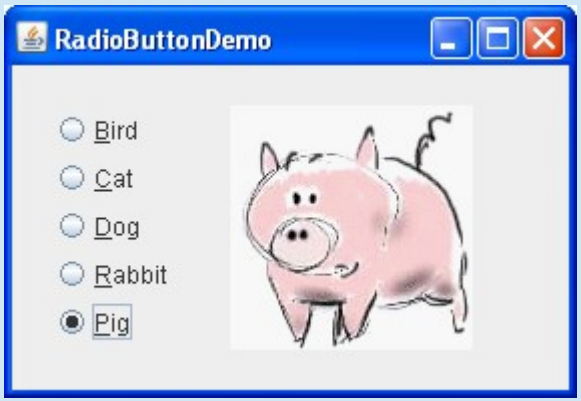
Check boxes are similar to radio buttons but their selection model is different, by convention. Any number of check boxes in a group — none, some, or all — can be selected. A group of radio buttons, on the other hand, can have only one button selected.

Outros componentes

- Checkbox



- Radio Button

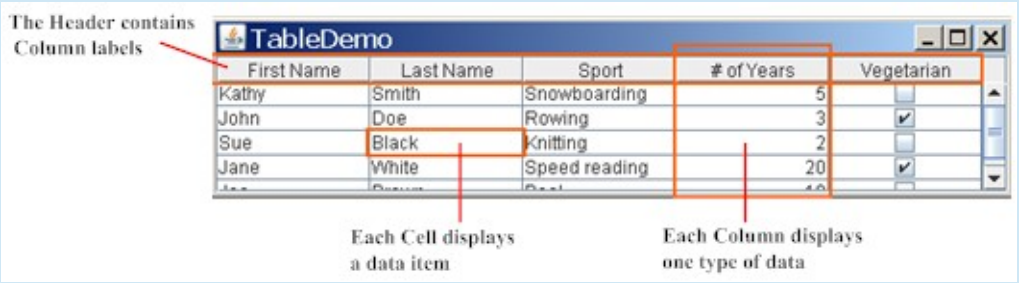


Check boxes are similar to radio buttons but their selection model is different, by convention. Any number of check boxes in a group — none, some, or all — can be selected. A group of radio buttons, on the other hand, can have only one button selected.

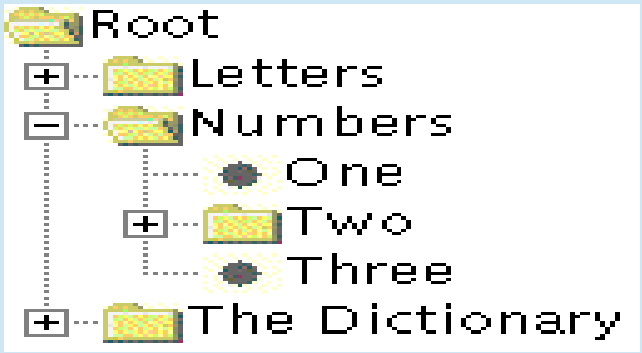
Em geral não é preciso registrar listener para esses componentes.

Outros Componentes

- JTable

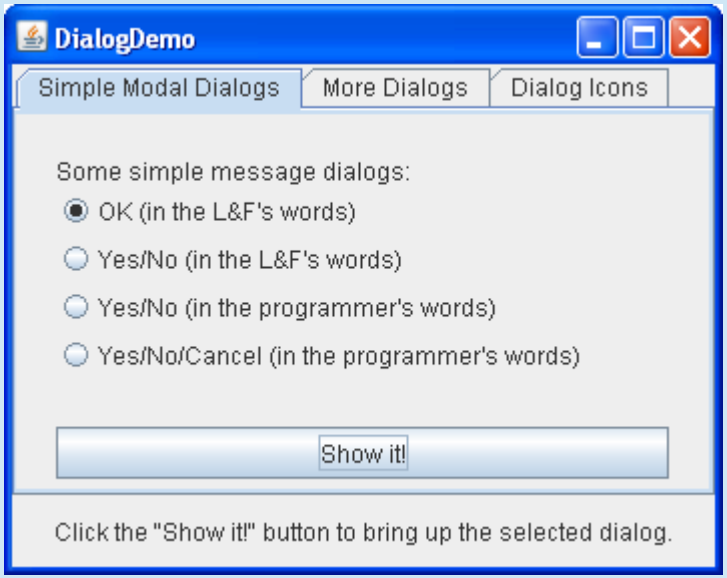


- JTree



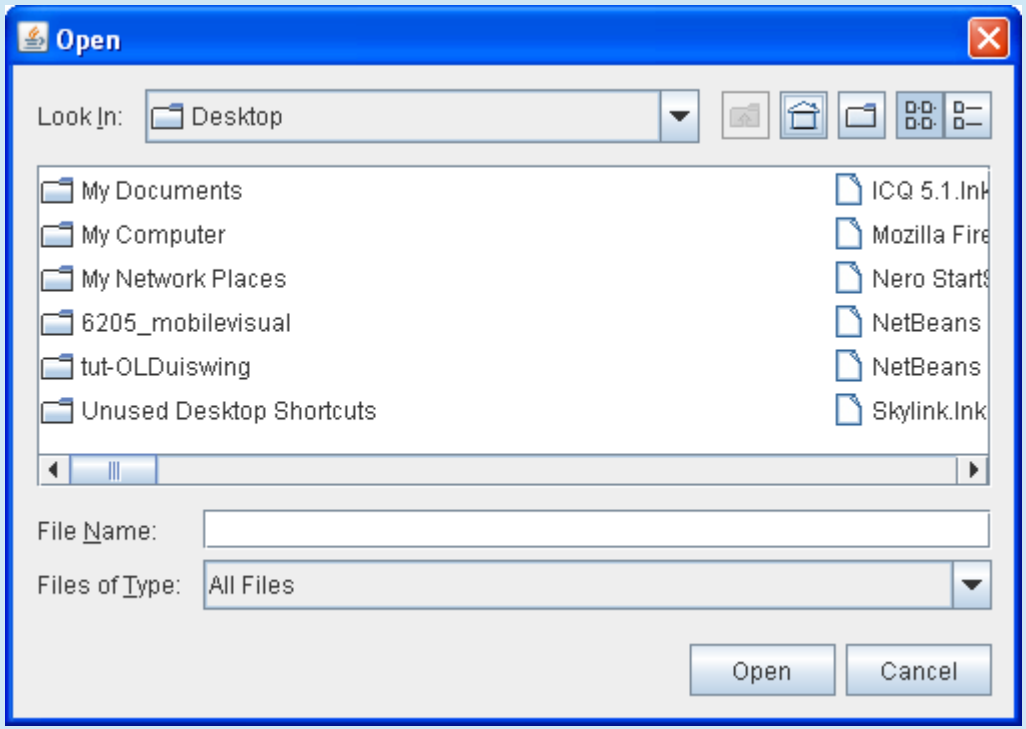
Diálogos

- Janeas que abrem separadas da interface
- Enquanto a janela não fecha, a interface fica travada



File chooser

- Dialogo específico para se escolher um arquivo



Look and feel

- É possível alterar a aparência dos componentes da GUI
- Para isso a API Java possui alguns “Look and Feel” que determinam a aparência
- É preciso saber quais estão instalados

UIManager

```
UIManager.LookAndFeelInfo[] looks =
  UIManager.getInstalledLookAndFeels();

for (int i = 0; i < looks.length; i++){

 System.out.println(looks[i].getClassName());

}
```

UIManager

```
UIManager.LookAndFeelInfo[] looks =
  UIManager.getInstalledLookAndFeels();

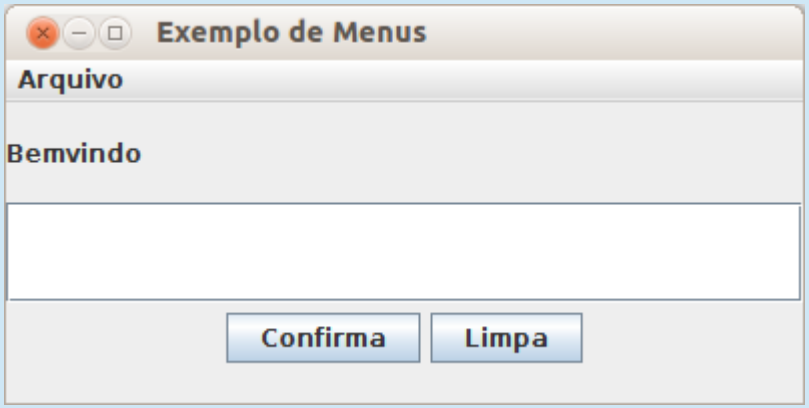
for (int i = 0; i < looks.length; i++){

 System.out.println(looks[i].getClassName());

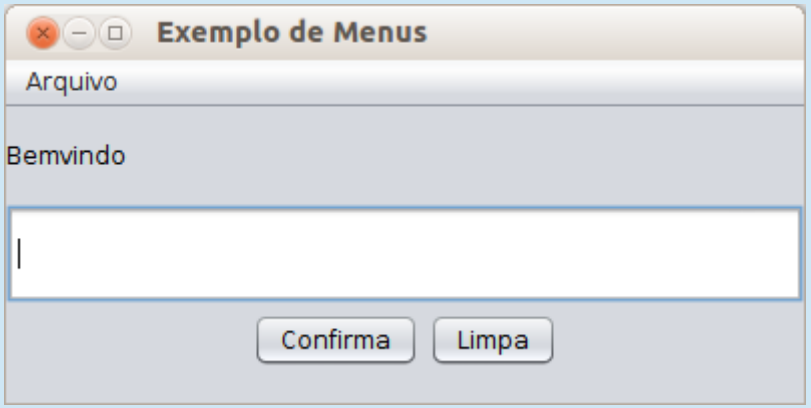
}


UIManager.setLookAndFeel(looks[k].getClassName());
```

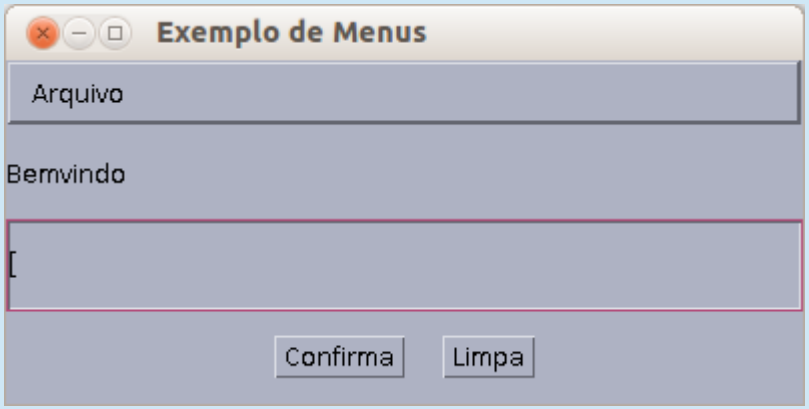
Look & feel exemplos



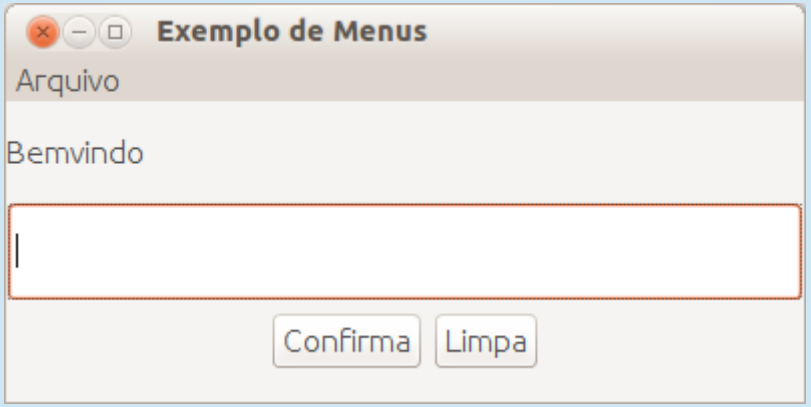
javax.swing.plaf.metal.MetalLookAndFeel



javax.swing.plaf.nimbus.NimbusLookAndFeel



com.sun.java.swing.plaf.motif.MotifLookAndFeel



com.sun.java.swing.plaf.gtk.GTKLookAndFeel