

Interface Gráfica

em Java

Vinicius Takeo Friedrich Kuwaki
Universidade do Estado de Santa Catarina

Seções

Introdução

Janelas

Painéis

Criando bordas

Textos e Imagens

Botões

Tipos de Layouts

Entradas de Texto

CheckBox

ComboBox e Listas

Tabelas

Janelas de opções

Introdução

- Java possui uma série de recursos nativos para a criação de interfaces gráficas;
- Iremos utilizar recursos de duas bibliotecas:
 - AWT;
 - A ideia do AWT é que os recursos gráficos atendam ao estilo da plataforma na qual são executadas;
 - Portanto, o design no Linux será diferente do design no Windows;
 - Swing;
 - É uma extensão da AWT;
- Basicamente, os componentes utilizados são da biblioteca Swing e os eventos (botão pressionado, etc.) são da biblioteca AWT;
- Vamos ver alguns componentes e como utilizá-los;

Introdução - JFrame

- **JFrame:** é a janela da aplicação, podendo agrupar outros componentes dentro de si;



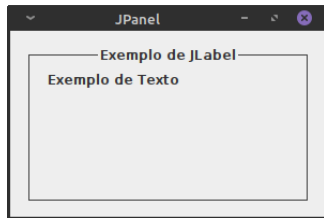
Introdução - JPanel

- **JPanel:** é um container para outros componentes, isto é, uma caixa que abriga os demais componentes;



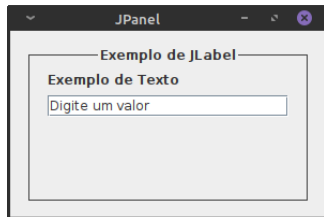
Exemplo de JFrame com um JPanel dentro e uma borda com título

- **JLabel:** exibe textos ou imagens na tela;



Exemplo de JLabel dentro de um JPanel

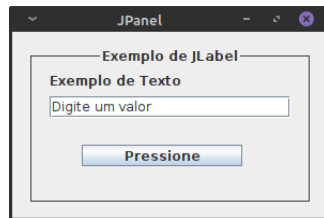
- **JTextField:** é uma caixa utilizada para a entrada de dados em texto;



Exemplo de JTextField dentro de um JPanel

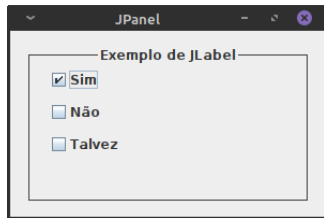
Introdução - JButton

- **JButton:** é um botão que executa uma determinada ação quando pressionado;



Exemplo de JButton dentro de um JPanel

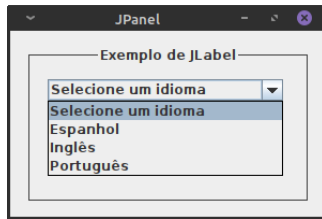
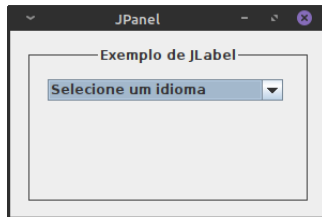
- **JCheckBox:** é uma caixa que possui dois estados booleanos, ou está marcada (true) ou está desmarcada (false);



Exemplo de JCheckBox dentro de um JPanel

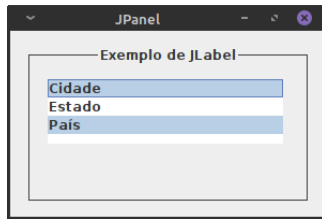
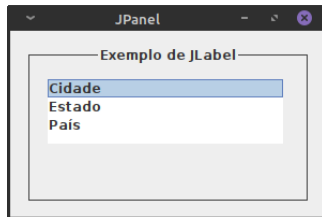
Introdução - JComboBox

- **JComboBox:** é uma gaveta que armazena uma lista de objetos do qual o usuário pode selecionar um deles;



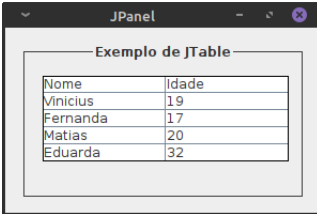
Introdução - JList

- **JList:** é uma lista de objetos, muito semelhante a uma tabela de uma só coluna;
- O usuário pode selecionar um ou mais itens da lista;



Introdução - JTable

- **JTable:** é uma tabela de dados que podem ser editáveis;

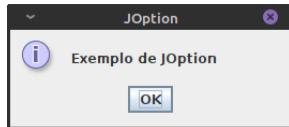


Exemplo de JTable

Nome	Idade
Vinicius	19
Fernanda	17
Matias	20
Eduarda	32

Introdução - JOptionPane

- **JOptionPane:** exibe um alerta na tela;



Introdução

- A seguir, veremos como implementar cada uma dessas componentes;
- Todos os códigos-fonte estão disponíveis nesse link.

Seções

Introdução

Janelas

Painéis

Criando bordas

Textos e Imagens

Botões

Tipos de Layouts

Entradas de Texto

CheckBox

ComboBox e Listas

Tabelas

Janelas de opções

Criando Janelas

- Vamos começar criando janelas;
- Para isso precisamos estender a classe JFrame;
- Vamos importar a biblioteca javax.swing.JFrame;
- Alguns métodos:
 - **setTitle(String titulo):** adiciona um título a janela;
 - **setDefaultCloseOperation(int arg0):** define o que acontece quando o botão X da janela é clicado, os argumentos que podem ser passados para esse método são:
 - **JFrame.DO_NOTHING_ON_CLOSE:** nada acontece quando a janela é fechada;
 - **JFrame.HIDE_ON_CLOSE:** apenas esconde a janela, alguns aplicativos desktop usam muito isso, por exemplo skype, steam, etc...
 - **JFrame.DISPOSE_ON_CLOSE:** fecha apenas a janela mas continua executando;
 - **JFrame.EXIT_ON_CLOSE:** fecha a janela e encerra aplicação;

Criando Janelas

- Alguns métodos:
 - **setBounds(int x, int y, int tamanhoX, int tamanhoY):** define onde a janela vai aparecer (x,y) em pixels e o tamanho da janela também em pixels;
 - **setResizable(boolean arg):** define se o usuário pode redimensionar a janela ou não;
 - **setBorder(Border borda):** define uma borda para a janela, veremos mais a frente sobre bordas;
 - **setLayout(LayoutManager layout):** define o layout da janela (também veremos mais a frente os tipos de layout);
 - **setIconImage(Image icone):** define o icone que aparece no sistema operacional;
 - **add(Component componente):** adiciona um componente (botão, um painel, etc.) a janela;
 - **setVisible(boolean arg0):** define se a JFrame estará visível ao usuário ou não;
- Os métodos destacados em azul podem ser usados por todos os tipos de componentes que veremos a seguir;

Criando Janelas - Implementação

- Vamos começar importando o JFrame e extendo a classe:
- Depois colocaremos um método **main()** para executar o JFrame e faremos a construção do JFrame dentro do construtor;

```
import javax.swing.JFrame;  
  
class ExemploJFrame extends JFrame {  
    public ExemploJFrame(){  
    }  
  
    public static void main(String[] args) {  
    }  
}
```

Criando Janelas - Implementação

- Vamos instanciar um objeto da classe `ExemploJFrame` que estende a classe `JFrame` e torná-la visível com o método **`setVisible()`**;

```
import javax.swing.JFrame;

class ExemploJFrame extends JFrame {

    public ExemploJFrame(){
    }

    public static void main(String [] args) {

        ExemploJFrame exemplo = new ExemploJFrame();
        exemplo.setVisible(true);

    }

}
```

Criando Janelas - Implementação

- Agora, vamos apenas mexer no construtor da classe;
- Vamos definir um título usando o método **setTitle()**;
- Também vamos definir as dimensões e onde o JFrame vai aparecer, com o método **setBounds()**;
- Vamos fazer ele aparecer a 50 pixels em x e em y do canto superior esquerdo da tela;
- Com um tamanho de 300 x 300 pixels;

```
import javax.swing.JFrame;

class ExemploJFrame extends JFrame {

    public ExemploJFrame(){

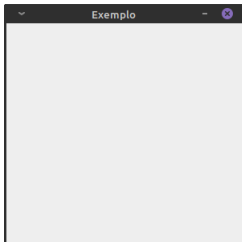
        setTitle("Exemplo");
        setBounds(50, 50, 300, 300);

    }

}
```

Criando Janelas - Implementação

- Vamos definir o que acontecerá ao pressionar o botão de fechar usando o método **setDefaultCloseOperation();**
- E também não deixaremos o usuário redimensionar a tela com o **setResizable();**



```
import javax.swing.JFrame;  
  
class ExemploJFrame extends JFrame {  
  
    public ExemploJFrame(){  
  
        setTitle("Exemplo");  
        setBounds(50, 50, 300, 300);  
        setDefaultCloseOperation(JFrame.  
EXIT_ON_CLOSE);  
        setResizable(false);  
    }  
  
}
```

Seções

Introdução

Janelas

Painéis

Criando bordas

Textos e Imagens

Botões

Tipos de Layouts

Entradas de Texto

CheckBox

ComboBox e Listas

Tabelas

Janelas de opções

Criando Painéis

- Agora vamos criar painéis;
- Eles servem para organizar melhor o código e os componentes da interface dentro da janela (JFrame);
- Recomenda-se um arquivo separado para cada painel criando uma classe que estenda o JPanel;
- JPanel possuem muitas propriedades semelhantes ao JFrame;
- Entretanto, eles não existem sem um JFrame;

- Alguns métodos:
 - **setBounds(int x, int y, int tamanhoX, int tamanhoY):** define onde o painel estará dentro do componente que o abriga, mesmo método do JFrame;
 - **setBorder(Border borda):** define uma borda para o painel, veremos mais a frente sobre bordas;
 - **setLayout(LayoutManager layout):** define o layout do painel (também veremos mais a frente os tipos de layout);
 - **add(Component componente):** adiciona um componente (botão, um painel e etc.) no painel;
 - **setBackground(Color cor):** define a cor de fundo do painel;

Criando Painéis - Implementação

- Vamos primeiro importar a classe `Color` e a `JPanel`;
- Depois vamos estender a classe `JPanel` e utilizaremos o método **`setBackground()`** para definir uma cor dentro do construtor;
- A classe `Color` possui várias constantes estáticas usadas para definir as cores, utilizaremos a cor `MAGENTA`;

```
import java.awt.Color;
import javax.swing.JPanel;

public class ExemploJPanel extends JPanel {

    public ExemploJPanel() {
        setBackground(Color.MAGENTA);
    }
}
```

Criando Painéis - Implementação

- Agora vamos criar outra classe para ser o JFrame que abrigará esse JPanel;
- Igual fizemos anteriormente, mas dessa vez vamos utilizar o método **add()** do JFrame para adicionarmos o JPanel dentro da janela;
- É uma boa prática criar uma variável e depois adicionar o objeto, mas como esse é um exemplo, criei o JPanel direto na chamada do método **add()**;

```
import javax.swing.JFrame;

class Principal extends JFrame {

    public Principal() {
        setTitle("Exemplo");
        setBounds(50, 50, 300, 300);
        setDefaultCloseOperation(JFrame.
EXIT_ON_CLOSE);
    }

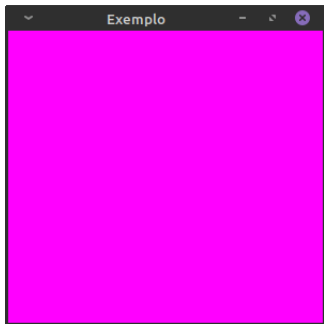
    public static void main(String[] args) {

        Principal exemplo = new Principal();
        exemplo.add(new ExemploJPanel());
        exemplo.setVisible(true);

    }
}
```

Criando Painéis - Implementação

- Como não definimos um layout para o JFrame, o JPanel vai assumir todo o tamanho do JFrame:



```
import javax.swing.JFrame;

class Principal extends JFrame {

    public Principal() {
        setTitle("Exemplo");
        setBounds(50, 50, 300, 300);
        setDefaultCloseOperation(JFrame.
EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        Principal exemplo = new Principal();
        exemplo.add(new ExemploJPanel());
        exemplo.setVisible(true);

    }

}
```

Criando Painéis - Paineis Multi-Abas

- É possível também criar painéis com várias abas utilizando a classe `JTabbedPane`;
- Cada “aba” será um painel diferente;
- Para utilizá-la, basta estender a classe `JTabbedPane` e usar o método **`addTab(String titulo, Componente componente)`** para criar uma nova aba;
- Vamos ver um exemplo:

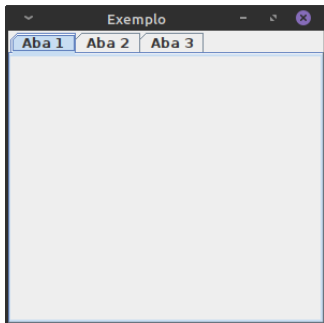
Criando Painéis - Paineis Multi-Abas

- Vamos estender a classe `JTabbedPane` e usar o método **`addTab()`**:

```
import javax.swing.JPanel;  
import javax.swing.JTabbedPane;  
  
public class ExemploJTabbedPane extends  
    JTabbedPane {  
  
    public ExemploJTabbedPane() {  
        this.addTab("Aba 1", new JPanel());  
        this.addTab("Aba 2", new JPanel());  
        this.addTab("Aba 3", new JPanel());  
    }  
}
```

Criando Painéis - Paineis Multi-Abas

- Feito isso, podemos colocar essa classe que estende o JTabbedPane dentro de um JFrame:



```
import javax.swing.JFrame;

public class Principal extends JFrame {

    public Principal() {
        setTitle("Exemplo");
        setBounds(50, 50, 300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        Principal exemplo = new Principal();
        exemplo.add(new ExemploJTabbedPane());
        exemplo.setVisible(true);

    }
}
```

Criando Painéis - Paineis Scrollaveis

- Outro tipo de painel que podemos usar é o JScrollPane;
- Quando o componente que vamos colocar dentro do painel é grande demais e/ou as vezes pode crescer, como é o caso de listas e tabelas;
- Usamos o JScrollPane para deslocar para os lados e para cima e baixo;
- Veremos como usar o básico de JScrollPane quando criarmos tabelas;

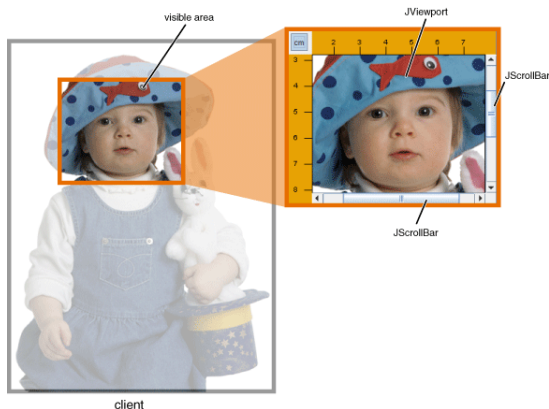


Figura 1: Exemplo da documentação. Fonte: docs.oracle.com

Seções

Introdução

Janelas

Painéis

Criando bordas

Textos e Imagens

Botões

Tipos de Layouts

Entradas de Texto

CheckBox

ComboBox e Listas

Tabelas

Janelas de opções

- Agora que vimos como criar JFrames e JPanels, vamos criar bordas para destacá-los;
- A biblioteca Swing possui um Factory para facilitar a criação;
- Basta importar a classe `javax.swing.BorderFactory`;
- Acesse a [documentação](#) para ver todos os métodos;
- Utilizaremos dois deles:
 - **`createLineBorder(Color cor)`**: retorna um Border com a cor passada como parâmetro;
 - **`createTitledBorder(String titulo)`**: retorna um Border com o título (é possível passar outro border antes do título para “incrementá-lo”);

Criando bordas - Implementação

- Primeiro, vamos importar as bibliotecas necessárias:
 - `java.awt.Color;`
 - `javax.swing.JPanel;`
 - `javax.swing.border.Border;`
 - `javax.swing.BorderFactory;`
- Vamos estender a classe `JPanel` e definir um construtor, utilizando o método **`createLineBorder()`** para criarmos uma borda preta:

```
import java.awt.Color;

import javax.swing.JPanel;
import javax.swing.border.Border;
import javax.swing.BorderFactory;

public class ExemploBorder extends JPanel {

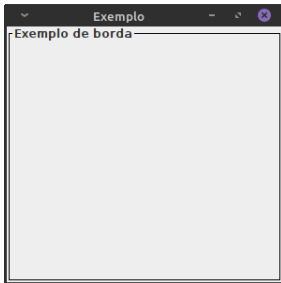
    public ExemploBorder() {

        Border lineBorder = BorderFactory.
            createLineBorder( Color.BLACK);

    }
```

Criando bordas - Implementação

- Utilizando a borda criada, vamos passa-la como parâmetro do método **createTitledBorder()** para criar uma borda com título:
- Depois vamos setar essa borda retornada pelo factory como a borda do JPanel;



```
import java.awt.Color;

import javax.swing.JPanel;
import javax.swing.border.Border;
import javax.swing.BorderFactory;

public class ExemploBorder extends JPanel {

    public ExemploBorder() {

        Border lineBorder = BorderFactory.
            createLineBorder(Color.BLACK);

        setBorder(BorderFactory.
            createTitledBorder(lineBorder, "Exemplo de
            borda"));
    }
}
```

- Mais tarde veremos como definir layouts, assim o JPanel pode ter o tamanho ajustado e a borda não ficará “colada” na janela;

Seções

Introdução

Janelas

Painéis

Criando bordas

Textos e Imagens

Botões

Tipos de Layouts

Entradas de Texto

CheckBox

ComboBox e Listas

Tabelas

Janelas de opções

Exibindo Textos

- Agora que vimos como criar JFrames e JPanels, vamos ver como fazer a saída de Strings em janelas e painéis;
- Para isso vamos utilizar o JLabel;
- O JLabel possui alguns métodos importantes:
 - **setText(String text)**: define o texto exibido;
 - **setIcon(Icon icone)**: define o icone exibido;
 - **setHorizontalAlignment(int arg0)**: define o alinhamento horizontal:
 - **JLabel.LEFT**: texto começa na esquerda;
 - **JLabel.CENTER**: texto começa no centro;
 - **JLabel.RIGHT**: texto começa na direita;
 - **setVerticalAlignment(int arg0)**: define o alinhamento vertical:
 - **JLabel.TOP**: texto começa na topo;
 - **JLabel.CENTER**: texto começa no centro;
 - **JLabel.BOTTOM**: texto começa em baixo;
 - **setBounds(int x, int y, int tamX, int tamY)**: define a origem e tamanho, igual ao JPanel;

Exibindo Textos

- Vamos primeiro importar as bibliotecas:
 - java.awt.Color para a borda;
 - javax.swing.JLabel;
 - javax.swing.JPanel;
 - javax.swing.BorderFactory;
- Vamos estender a classe JPanel e no construtor definirmos o layout como `null` (mais a frente veremos o porquê);

```
import java.awt.Color;

import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.BorderFactory;

public class ExemploJLabel extends JPanel {

    public ExemploJLabel() {

        setLayout(null);

    }

}
```

- Para vermos melhor como funciona o alinhamento horizontal, vamos criar três JLabels;
- Cada uma terá 150 pixels de largura e 20 pixels de altura, utilizaremos o método **setBounds()** para isso;
- Utilizaremos o método **setBorder()** para colocar uma borda preta ao redor do texto, para melhor visualizar o alinhamento;
- Todos os textos irão aparecer em 20 pixels de distância em x e y do começo do JPanel;

Exibindo Textos

- A primeira JLabel será alinhada à esquerda, utilizaremos o método **setHorizontalAlignment()**;
- No construtor passaremos a String que será exibida;
- Utilizando o método **setBounds()** para definirmos o tamanho e a origem;
- Definiremos uma borda preta também;

```
public ExemploJLabel() {  
  
    setLayout( null );  
  
    JLabel texto = new JLabel("Exemplo de texto");  
    texto.setBounds(20, 20, 150, 20);  
    texto.setHorizontalAlignment(JLabel.LEFT);  
    texto.setBorder(BorderFactory.createLineBorder(Color.black));  
    add(texto);  
  
}  
  
}
```


Exibindo Textos

- Já a segunda será alinhada com o centro;
- Utilizando o método **setBounds()** para definirmos o tamanho e a origem, que dessa vez estará em $y = 50$;
- A JLabel anterior estava em $y = 20$, e tinha 20 de altura, logo ela acaba em $y = 40$, daremos mais 10 pixels de espaço;

```
public ExemploJLabel() {  
  
    ...  
  
    JLabel texto2 = new JLabel("Exemplo de texto");  
    texto2.setBounds(20, 50, 150, 20);  
    texto2.setHorizontalAlignment(JLabel.CENTER);  
    texto2.setBorder(BorderFactory.createLineBorder(Color.black));  
    add(texto2);  
}
```

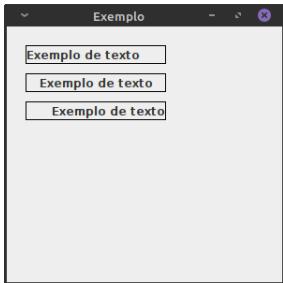
Exibindo Textos

- Por fim, a terceira estará alinhada à direita;
- Seu y será 80, (30 pixels a mais da JLabel anterior);

```
public ExemploJLabel() {  
    ...  
    JLabel texto3 = new JLabel("Exemplo de texto");  
    texto3.setBounds(20, 80, 150, 20);  
    texto3.setHorizontalAlignment(JLabel.RIGHT);  
    texto3.setBorder(BorderFactory.createLineBorder(Color.black));  
    add(texto3);  
}
```

Exibindo Textos

- Adicionado esse JPanel contendo as JLabels em um JFrame, obtemos o seguinte resultado:



- Note a diferença dos alinhamentos;

```
import javax.swing.JFrame;

public class Principal extends JFrame {

    public Principal() {
        setTitle("Exemplo");
        setBounds(50, 50, 300, 300);
        setDefaultCloseOperation(JFrame.
EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        Principal exemplo = new Principal();
        exemplo.add(new ExemploJLabel());
        exemplo.setVisible(true);

    }
}
```

Exibindo Imagens - Implementação

- É possível utilizar o JLabel para exibir imagens também;
- Vamos criar outra classe que estende o JPanel e adicionar um JLabel;
- Ao instanciar o JLabel iremos passar um novo objeto do tipo ImageIcon como parâmetro;
- Na construção do ImageIcon informaremos o caminho da imagem;

```
import javax.swing.ImageIcon;  
import javax.swing.JLabel;  
import javax.swing.JPanel;  
  
public class ExemploImagem extends JPanel {  
  
    public ExemploImagem() {  
  
        JLabel imagem = new JLabel(new  
            ImageIcon("jlabel/udesc.png"));  
        add(imagem);  
  
    }  
  
}
```

Exibindo Imagens - Implementação

- Adicionado esse novo JPanel ao JFrame, vamos apenas aumentar o tamanho em x para que a imagem caiba inteira dentro;
- Vamos definir o tamanho em x do JFrame como 400;



```
import javax.swing.JFrame;

public class Principal extends JFrame {

    public Principal() {
        setTitle("Exemplo");
        setBounds(50, 50, 400, 300);
        setDefaultCloseOperation(JFrame.
EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        Principal exemplo = new Principal();
        exemplo.add(new ExemploImagem());
        exemplo.setVisible(true);

    }
}
```

Seções

Introdução

Janelas

Painéis

Criando bordas

Textos e Imagens

Botões

Tipos de Layouts

Entradas de Texto

CheckBox

ComboBox e Listas

Tabelas

Janelas de opções

Criando botões

- Antes de vermos como funciona os layouts em Java, vamos aprender a criar botões e manipular suas ações;
- Botões e muitos outros componentes em Java aproveitam do padrão Observer para implementar suas funcionalidades;
- De fato, faz sentido ficar “observando” algo e disparar um evento quando algo acontecer;
- Para isso, os botões em Java observam eventos pré-programados em instâncias de classes que implementam a interface `ActionListener`;
- Para uma classe implementar essa interface, ela só precisa implementar o método **`actionPerformed()`**;
- Vamos ver um exemplo a seguir;

- Assim como os outros componentes apresentados anteriormente, o JButton possui métodos importantes, vale destacar alguns:
 - **addActionListener(ActionListener actionListener)**: adiciona um observador que observa quando o botão é pressionado e executa uma ação;
 - **setIcon(Icon icone)**: adiciona um icone ao botão;
 - **setBounds(...)**: define as dimensões e origem, assim como nos componentes anteriores;
 - **setText(String texto)**: define o texto do botão;

Criando botões - Implementação

- Primeiro vamos criar um JPanel, assim como fizemos nos componentes anteriores;
- Vamos importar as bibliotecas:
 - java.awt.Color; (utilizaremos para ação)
 - java.awt.event.ActionEvent;
 - java.awt.event.ActionListener;
 - javax.swing.JButton;
 - javax.swing.JPanel;

```
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JPanel;

public class ExemploJButton extends JPanel {

    public ExemploJButton() {
```

Criando botões - Implementação

- Em nosso exemplo, vamos criar um botão que ao ser pressionado troca a cor do JPanel;
- Para isso, vamos criar um método que pega a cor do painel utilizando o método **getBackground()** e verifica se a cor é vermelha;
- Se ela for, troca para azul, caso não seja, troca para vermelha;

```
public void trocaCor() {  
    if (getBackground().equals(Color.BLUE)) {  
        setBackground(Color.RED);  
    } else {  
        setBackground(Color.BLUE);  
    }  
}
```

Criando botões - Implementação

- Agora, vamos instanciar um objeto do tipo JButton no construtor e definir o texto do botão;

```
public ExemploJButton() {  
    JButton button = new JButton("Pressione");  
    add(button);  
}
```

Criando botões - Implementação

- Depois, vamos utilizar o método **addActionListener()** para adicionar o observador de eventos;
- Esse método recebe como parâmetro uma instância de alguma classe que implementa a interface `ActionListener`;

```
public ExemploJButton() {  
    JButton button = new JButton("Pressione");  
    add(button);  
    button.addActionListener( ... );  
}
```

Criando botões - Implementação

- Em vez de criar em um arquivo separado, vamos criar uma classe anônima que implementa a interface ActionListener dentro da instanciação de um objeto dela;
- Como só vamos usar ela uma vez, faz mais sentido ela ser anônima;
- Além do mais, dessa forma ela pode acessar componentes do JPanel por estar dentro do escopo do JPanel;

```
public ExemploJButton() {  
    JButton button = new JButton("Pressione");  
    add(button);  
    button.addActionListener( new ActionListener(){  
        // Implementacao  
    } );  
}
```

Criando botões - Implementação

- A interface ActionListener pede a implementação de apenas um método: **void actionPerformed(ActionEvent arg);**
- Nele será definida a lógica a ser executada quando o botão for pressionado;

```
public ExemploJButton() {  
  
    JButton button = new JButton("Pressione");  
    add(button);  
    button.addActionListener( new ActionListener() {  
  
        @Override  
        public void actionPerformed(ActionEvent arg0) {  
            // Funcionalidades do botao  
        }  
  
    } );  
}
```

Criando botões - Implementação

- Então, quando o botão for pressionado, vamos chamar o método **trocaCor()** que criamos anteriormente;

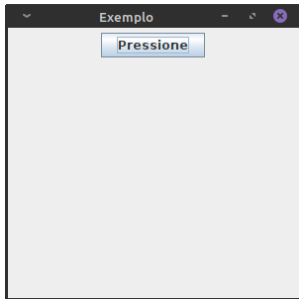


Figura 2: Janela ao ser criada

```
public ExemploJButton() {  
  
    JButton button = new JButton("Pressione");  
    add(button);  
    button.addActionListener( new ActionListener(){  
  
        @Override  
        public void actionPerformed(ActionEvent arg0)  
        {  
            trocaCor();  
        }  
    } );  
}
```

Criando botões - Implementação

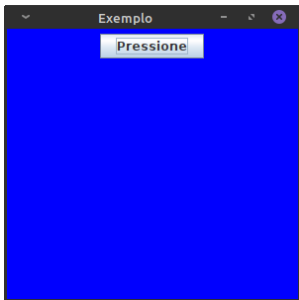


Figura 3: Botão quando pressionado e a cor for vermelha, ou nula

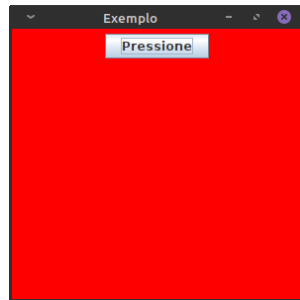


Figura 4: Botão quando pressionado e a cor for azul

Seções

Introdução

Janelas

Painéis

Criando bordas

Textos e Imagens

Botões

Tipos de Layouts

Entradas de Texto

CheckBox

ComboBox e Listas

Tabelas

Janelas de opções

Criando Layouts

- Agora, vamos ver como organizar os componentes dentro de um container (JPanel, JFrame e etc.);
- Existem alguns tipos de layouts para isso;
- Veremos cinco deles:
 - **Layout Absoluto:** delega para os objetos a posição deles na tela e seus tamanhos;
 - **GridLayout:** divide a tela em várias células, basicamente uma matriz $n \times m$;
 - **FlowLayout:** posiciona os componentes lado a lado, da esquerda para a direita e de cima para baixo;
 - **BorderLayout:** posiciona os componentes de acordo com as regiões da tela (norte, sul, leste, etc...);
 - **CardLayout:** permite que um só componente ocupe todo o espaço, e eventualmente “reveze” com os demais;

Criando Layouts - Layout Absoluto

- O primeiro layout que veremos é o layout absolutos;
- Na verdade, já utilizamos ele anteriormente;
- Quando você define o layout como sendo `null`, utilizando o método **`setLayout()`**, esse componente terá um layout absoluto;
- Isto é, quem define a disposição dos componentes na tela são os próprios componentes, utilizando os métodos:
 - **`setBound(int x, int y, int tamX, int tamY)`** para definir a localização em x e y e o tamanho;
 - **`setLocation(int x, int y)`** para definir só a localização;
 - **`setSize(int tamX, int tamY)`** para definir só o tamanho;
- Para o exemplo, vamos usar JButtons (não criaremos ações para eles) para visualizarmos como eles se dispõem na tela;

Criando Layouts - Layout Absoluto

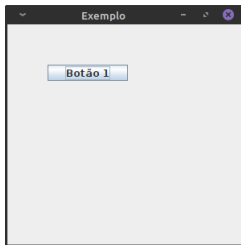
- Vamos criar um JFrame e definir o seu layout;
- Utilizando o método **setLayout()** passando **null** como argumento;

```
import javax.swing.JButton;  
import javax.swing.JFrame;  
  
class ExemploNullLayout extends JFrame {  
    public ExemploNullLayout() {  
        setTitle("Exemplo");  
        setSize(300, 300);  
        setLocation(50, 50);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setLayout(null);  
    }  
}
```

Criando Layouts - Layout Absoluto

- Agora, no método **main()** vamos adicionar botões;
- Poderia ser feito também no construtor caso queira;
- Vamos fazer o primeiro botão aparecer em $x = 50$ e $y = 50$;
- Ele terá um tamanho de 100 pixels de largura (x) e 20 de altura (y);

```
public static void main(String[] args) {  
  
    ExemploNullLayout exemplo = new ExemploNullLayout();  
    exemplo.setVisible(true);  
  
    JButton botao1 = new JButton("Botao 1");  
    botao1.setBounds(50, 50, 100, 20);  
    exemplo.add(botao1);  
}
```



Criando Layouts - Layout Absoluto

- O segundo botão, irá surgir um pouco mais a direita do primeiro, em $x = 120$;
- E estará em baixo do botão 1, em $y = 8$, com o mesmo tamanho que ele;

```
public static void main(String[] args) {  
  
    JButton botao2 = new JButton("Botao 2");  
    botao2.setBounds(120, 80, 100, 20);  
    exemplo.add(botao2);  
}
```



Criando Layouts - Layout Absoluto

- Para finalizar, o terceiro seguirá a mesma lógica;
- Vai estar “entre” os dois botões em x e abaixo dos dois;
- Com o mesmo tamanho também;

```
public static void main(String[] args) {  
  
    JButton botao3 = new JButton("Botao 3");  
    botao3.setBounds(85, 110, 100, 20);  
    exemplo.add(botao3);  
}
```



Criando Layouts - Layout Absoluto

- As vantagens de se usar o Absolut Layout é que o programador tem total controle em pixels sobre os componentes;
- Vale ressaltar também que cada componente tem o seu layout, isto é, você pode ter um JPanel com um layout absoluto, dentro de um JFrame com um layout de grades (GridLayout) que veremos agora;

Criando Layouts - GridLayout

- O GridLayout, ou layout de grade, divide o espaço dentro de um container em uma matriz $n \times m$;
- Novamente, vamos utilizar botões;
- Precisamos importar a biblioteca `import java.awt.GridLayout` para usarmos o layout de grade;
- Vamos repetir os passos do exemplo anterior, e definir o layout direto no JFrame;

Criando Layouts - GridLayout

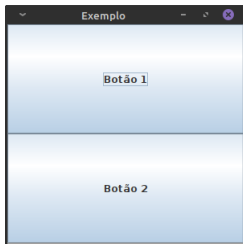
- Vamos utilizar o método **setLayout()** novamente, mas dessa vez, passando um objeto `GridLayout()`;
- Vamos criar um layout de 2×3 , isso é, duas linhas e três colunas;

```
import javax.swing.JButton;  
import javax.swing.JFrame;  
  
import java.awt.GridLayout;  
  
public class ExemploGridLayout extends JFrame {  
    public ExemploGridLayout() {  
        setTitle("Exemplo");  
        setSize(300, 300);  
        setLocation(50, 50);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setLayout(new GridLayout(2, 3));  
    }  
}
```

Criando Layouts - GridLayout

- Note que, como definimos o layout como 2x3, seis espaços serão disponibilizados;
- Vamos instanciar 5 botões;
- Vamos deixar um posição vazia;
- Vamos adicionar primeiro 2 botões;
- Como definimos o layout com duas linhas, a tela será dividida em duas;

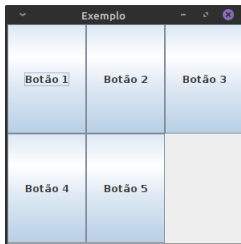
```
public static void main(String[] args) {  
  
    ExemploGridLayout exemplo = new ExemploGridLayout();  
    exemplo.setVisible(true);  
  
    JButton botao1 = new JButton("Botao 1");  
    exemplo.add(botao1);  
  
    JButton botao2 = new JButton("Botao 2");  
    exemplo.add(botao2);  
}
```



Criando Layouts - GridLayout

- Mas vamos adicionar mais três botões:
- Veja como os componentes se dividiram dentro do espaço que definimos de duas linhas e três colunas;
- Note também que não definimos o tamanho do botão e nem a posição onde ele aparece, quem faz isso é o layout;

```
public static void main(String[] args) {  
  
    JButton botao3 = new JButton("Botao 3");  
    exemplo.add(botao3);  
  
    JButton botao4 = new JButton("Botao 4");  
    exemplo.add(botao4);  
  
    JButton botao5 = new JButton("Botao 5");  
    exemplo.add(botao5);  
}
```



Criando Layouts - FlowLayout

- Agora vamos ver o FlowLayout;
- Ele possui similaridades com o GridLayout, entretanto, você não define a quantidade de linhas e colunas;
- O layout administra isso automaticamente;
- Para esse layout, utilizamos a biblioteca `import java.awt.FlowLayout;`

Criando Layouts - FlowLayout

- Assim, como fizemos nos dois exemplos anteriores, vamos criar um JFrame com esse layout;
- Setando ele com o método **setLayout()** do JFrame;

```
import java.awt.FlowLayout;

import javax.swing.JButton;
import javax.swing.JFrame;

public class ExemploFlowLayout extends JFrame {

    public ExemploFlowLayout() {
        setTitle("Exemplo");
        setSize(300, 300);
        setLocation(50, 50);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());
    }
}
```

Criando Layouts - FlowLayout

- Vamos começar criando três botões;
- Eles irão ficar dispostos um ao lado do outro;



```
public static void main(String[] args) {  
  
    ExemploFlowLayout exemplo = new ExemploFlowLayout();  
    exemplo.setVisible(true);  
  
    JButton botao1 = new JButton("Botao 1");  
    exemplo.add(botao1);  
  
    JButton botao2 = new JButton("Botao 2");  
    exemplo.add(botao2);  
  
    JButton botao3 = new JButton("Botao 3");  
    exemplo.add(botao3);  
}
```

Criando Layouts - FlowLayout

- Observe o que acontece quando adicionamos um quarto botão:
- O layout escolhe um local para ele;



```
public static void main(String[] args) {  
  
    ExemploFlowLayout exemplo = new ExemploFlowLayout();  
    exemplo.setVisible(true);  
  
    JButton botao1 = new JButton("Botao 1");  
    exemplo.add(botao1);  
  
    JButton botao2 = new JButton("Botao 2");  
    exemplo.add(botao2);  
  
    JButton botao3 = new JButton("Botao 3");  
    exemplo.add(botao3);  
  
    JButton botao4 = new JButton("Botao 4");  
    exemplo.add(botao4);  
}
```


Criando Layouts - FlowLayout

- E quando adicionamos um quinto botão...



```
public static void main(String[] args) {  
  
    ExemploFlowLayout exemplo = new ExemploFlowLayout();  
    exemplo.setVisible(true);  
  
    JButton botao1 = new JButton("Botao 1");  
    exemplo.add(botao1);  
  
    JButton botao2 = new JButton("Botao 2");  
    exemplo.add(botao2);  
  
    JButton botao3 = new JButton("Botao 3");  
    exemplo.add(botao3);  
  
    JButton botao4 = new JButton("Botao 4");  
    exemplo.add(botao4);  
  
    JButton botao5 = new JButton("Botao 5");  
    exemplo.add(botao5);  
  
}
```

Criando Layouts - BorderLayout

- O próximo layout que veremos dispõe os componentes de acordo com a região da tela;
- Isto é, separa a tela em:
 - Norte;
 - Sul;
 - Leste;
 - Oeste;
 - E centro;
- Para definirmos a localização do componente na tela, passamos o local no método **add()**;
- Vamos ver melhor isso no exemplo;
- Para usar o BorderLayout, precisamos da biblioteca: `java.awt.BorderLayout`;

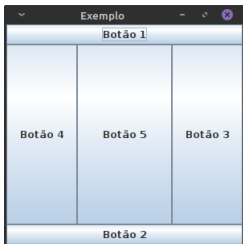
Criando Layouts - BorderLayout

- E vamos começar repetindo novamente os passos dos exemplos anteriores;
- No método **setLayout()** vamos passar um objeto BorderLayout;

```
import java.awt.BorderLayout;  
  
import javax.swing.JButton;  
import javax.swing.JFrame;  
  
class ExemploBorderLayout extends JFrame {  
    public ExemploBorderLayout() {  
        setTitle("Exemplo");  
        setSize(300, 300);  
        setLocation(50, 50);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setLayout(new BorderLayout());  
    }  
}
```

Criando Layouts - BorderLayout

- E para cada uma das posições, vamos utilizar o método **add()**:



```
public static void main(String[] args) {  
  
    ExemploBorderLayout ex = new ExemploBorderLayout();  
    ex.setVisible(true);  
  
    JButton botao1 = new JButton("Botao 1");  
    ex.add(botao1, BorderLayout.NORTH);  
  
    JButton botao2 = new JButton("Botao 2");  
    ex.add(botao2, BorderLayout.SOUTH);  
  
    JButton botao3 = new JButton("Botao 3");  
    ex.add(botao3, BorderLayout.EAST);  
  
    JButton botao4 = new JButton("Botao 4");  
    ex.add(botao4, BorderLayout.WEST);  
  
    JButton botao5 = new JButton("Botao 5");  
    ex.add(botao5, BorderLayout.CENTER);  
  
}
```

Criando Layouts - CardLayout

- Por fim, vamos ver o CardLayout;
- Esse layout cria uma espécie de “revezamento” entre os componentes;
- Hora um, hora outro aparecem na tela;
- Para trocar o componente que está aparecendo, utiliza-se o método **next()** do layout;
- Nesse exemplo, iremos manter o layout como um atributo do JFrame para podermos acessá-lo mais facilmente;
- O método **next()** recebe como parâmetro o componente que abriga o layout;
- Vamos usar a biblioteca **import java.awt.CardLayout;**
- Nesse exemplo faremos um pouco diferente dos exemplos dos demais layouts;
- Vamos definir o painel, o layout e um botão como atributos da classe que estende o JFrame;

Criando Layouts - CardLayout

- Primeiro, vamos importar o CardLayout;
- Utilizaremos um BorderLayout para dividirmos a tela entre o botão e um painel;

```
import java.awt.CardLayout;  
import java.awt.BorderLayout;  
import java.awt.Color;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
import javax.swing.JButton;  
import javax.swing.JFrame;  
import javax.swing.JPanel;
```

Criando Layouts - CardLayout

- Também vamos criar um botão que irá trocar o painel ao ser clicado;

```
public class ExemploCardLayout extends JFrame {  
  
    private CardLayout cardLayout = new CardLayout();  
    private JPanel painelCard = new JPanel(cardLayout);  
    private JButton botao = new JButton("Trocar");  
  
    public ExemploCardLayout() {  
  
        setTitle("Exemplo");  
        setSize(300, 300);  
        setLocation(50, 50);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setLayout(new BorderLayout());  
    }  
}
```

Criando Layouts - CardLayout

- Agora vamos adicionar o painel no centro e o botão no sul;
- Além de criamos dois painéis a serem revezados pelo CardLayout, um vermelho e um azul;

```
public ExemploCardLayout() {  
    add(painelCard, BorderLayout.CENTER);  
    add(botao, BorderLayout.SOUTH);  
  
    JPanel vermelho = new JPanel();  
    vermelho.setBackground(Color.RED);  
    painelCard.add(vermelho);  
  
    JPanel azul = new JPanel();  
    azul.setBackground(Color.blue);  
    painelCard.add(azul);  
}
```


Criando Layouts - CardLayout

- Vamos adicionar um ActionListener para o botão que irá trocar o painel em exibição no CardLayout;
- Utilizaremos o método **next()** do layout, passando o painel como parâmetro;

```
public ExemploCardLayout() {  
    botao.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent arg0) {  
            cardLayout.next(painelCard);  
        }  
    });  
}
```

Criando Layouts - CardLayout

- Por fim, um método **main()** para executar o código;

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

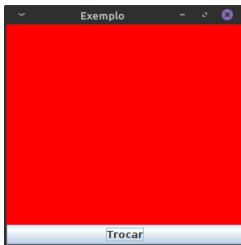


Figura 5: Painel com a cor vermelha a frente no CardLayout
82/122



Figura 6: Painel com a cor azul a frente no CardLayout

Seções

Introdução

Janelas

Painéis

Criando bordas

Textos e Imagens

Botões

Tipos de Layouts

Entradas de Texto

CheckBox

ComboBox e Listas

Tabelas

Janelas de opções

Criando caixas de entrada de texto

- A maioria dos programas que criamos precisa que o usuário entre com valores que serão processados por ele;
- Em interfaces gráficas temos recursos para isso;
- O `JTextFields` fazem a entrada de dados para nossos programas;
- Podendo também realizar a saída;
- Eles possuem os métodos:
 - **`getText()`**: que retorna a `String` que está dentro da caixa;
 - **`setText(String text)`**: que define a `String` dentro da caixa;
 - **`setEditable(boolean b)`**: define se o usuário pode ou não alterar seu texto;
 - **`setColumns()`**: define a quantidade de colunas que a caixa vai ter (não é a quantidade de caracteres!)

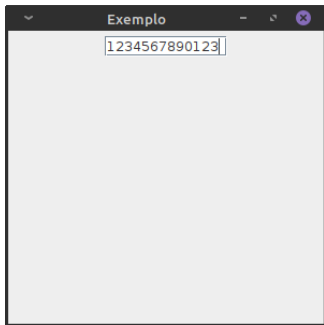
Criando caixas de entrada de texto - JTextField

- Vamos ver um exemplo bem simples;
- Vamos criar um JPanel contendo um JTextField de 10 colunas;

```
import javax.swing.JPanel;  
import javax.swing.JTextField;  
  
public class ExemploJTextField extends JPanel {  
    public ExemploJTextField() {  
        JTextField textField = new JTextField();  
        textField.setColumns(10);  
        add(textField);  
    }  
}
```

Criando caixas de entrada de texto - JTextField

- Veja a quantidade de números que cabem em 10 colunas:



```
import javax.swing.JFrame;

public class Principal extends JFrame {

    public Principal() {
        setTitle("Exemplo");
        setBounds(50, 50, 300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        Principal exemplo = new Principal();
        exemplo.add(new ExemploJTextField());
        exemplo.setVisible(true);

    }
}
```

Seções

Introdução

Janelas

Painéis

Criando bordas

Textos e Imagens

Botões

Tipos de Layouts

Entradas de Texto

CheckBox

ComboBox e Listas

Tabelas

Janelas de opções

Criando Checkboxes

- Agora veremos um componente para lidarmos com variáveis booleanas;
- Não seria prático fazer o usuário digitar um conjunto de valores e para cada um verificar se ele é true ou false, certo?
- Para isso, usaremos o JCheckBox;
- Alguns métodos que podem ser usados:
 - **setText(String texto):** define o texto que ficará ao lado da caixa de marcação;
 - **setIcon(Icon icone):** se você preferir uma imagem ao invés de um texto, utilize esse método;
 - **setSelected(boolean b):** define se a caixa está marcada ou não;
 - **isSelected():** retorna true caso a caixa esteja marcada e false caso contrário;
- Utilizaremos a biblioteca `javax.swing.JPanel;`

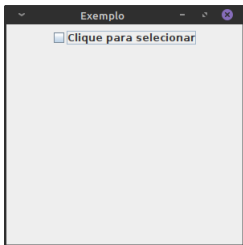
Criando Checkboxes - Implementação

- Como fizemos anteriormente, vamos colocar o componente (JComboBox) dentro de um painel;
- Passaremos o texto no construtor;

```
import javax.swing.JCheckBox;  
import javax.swing.JPanel;  
  
public class ExemploJCheckBox extends JPanel {  
    public ExemploJCheckBox() {  
        JCheckBox checkBox = new JCheckBox("Clique para  
selecionar");  
        add(checkBox);  
    }  
}
```

Criando Checkboxes - Implementação

- Colocando esse painel dentro de um JFrame temos:



```
import javax.swing.JFrame;

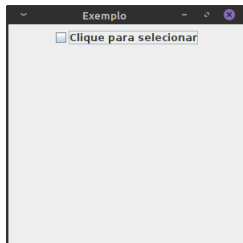
public class Principal extends JFrame {

    public Principal() {
        setTitle("Exemplo");
        setBounds(50, 50, 300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

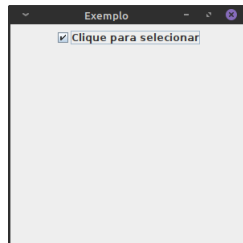
    public static void main(String[] args) {

        Principal exemplo = new Principal();
        exemplo.add(new ExemploJCheckBox());
        exemplo.setVisible(true);
    }
}
```

Criando Checkboxes - Implementação



- **isSelected()** retorna **false**;



- **isSelected()** retorna **true**;

Seções

Introdução

Janelas

Painéis

Criando bordas

Textos e Imagens

Botões

Tipos de Layouts

Entradas de Texto

CheckBox

ComboBox e Listas

Tabelas

Janelas de opções

Criando ComboBoxes

- JComboBox e List têm finalidades diferentes, entretanto, suas formas de criação são semelhantes;
- Vamos começar criando JComboBox;
- Elas são tipadas para uma classe T;
- Podendo em seu construtor receber um **array** contendo os itens que ficarão dentro da JComboBox;
- Esse array precisa ser do tipo T definido na criação;
- Caso a lista precise ser criada dinamicamente, utilize o método **addItem(T objeto)** para adicionar um item a JComboBox;

- Alguns métodos da JComboBox:
 - **getSelectedItem():** retorna um Object contendo o objeto selecionado;
 - **getSelectedIndex():** retorna a posição do array em que se encontra o item atualmente selecionado;
 - **setMaximumRowCount(int max):** define um limite para a quantidade de itens que podem existir no JComboBox;
 - **insertItemAt(T objeto, int index):** adiciona um item no index especificado;
 - **removeItem(Object item):** remove o item passado como parâmetro;
 - **removeItemAt(int index):** remove o item na posição passada como parâmetro;
 - **removeAllItems():** remove todos os itens;
 - **getItemAt(int index):** retorna o item na posição tal;

Criando ComboBoxes - Implementação

- Vamos primeiro criar um array de Strings;
- Passaremos esse array no construtor do JComboBox;
- Note que tipamos o JComboBox com o mesmo tipo do array;

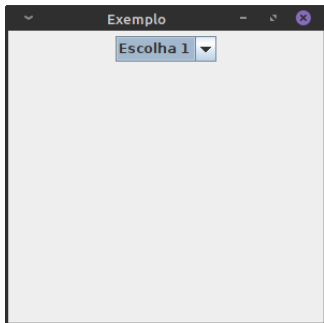
```
import javax.swing.JComboBox;
import javax.swing.JPanel;

public class ExemploJComboBox extends JPanel {

    public ExemploJComboBox() {
        String [] itens = new String [] { "Escolha 1", "Escolha
2" };
        JComboBox<String> comboBox = new JComboBox<String>(
            itens);
        add(comboBox);
    }
}
```

Criando ComboBoxes - Implementação

- O resultado obtido é:



```
import javax.swing.JFrame;

public class Principal extends JFrame {

    public Principal() {
        setTitle("Exemplo");
        setBounds(50, 50, 300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        Principal exemplo = new Principal();
        exemplo.add(new ExemploJComboBox());
        exemplo.setVisible(true);

    }
}
```


Criandos Listas

- Como mencionado anteriormente, a criação das JLists é muito semelhante a criação das JComboBox;
- Ambas são tipadas com um genérico T;
- A diferença é que, ao contrário do JComboBox, aqui todos os itens ficam visíveis;
- Alguns métodos:
 - **getSelectedIndices()**: retorna um array de inteiros contendo os indexes selecionados;
 - **getSelectedValues()**: retorna os itens selecionados na forma de um array de genéricos T;
 - **getSelectedValuesList()**: retorna os itens selecionados na forma de uma List de T;
 - **getSelectedIndex()**: retorna o primeiro index selecionado;
 - **getSelectedValue()**: retorna o primeiro item selecionado;
 - **removeSelectionInterval(int inicio, int fim)**: remove os itens entre o início e o fim;

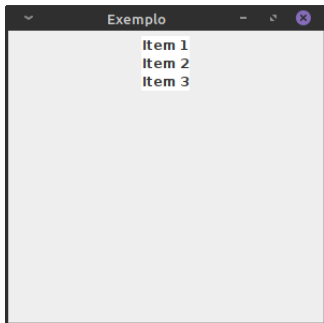
Criando Listas - Implementação

- Vamos primeiro criar um array de Strings;
- Passaremos esse array no construtor;

```
import javax.swing.JList;  
import javax.swing.JPanel;  
  
public class ExemploJList extends JPanel {  
    public ExemploJList() {  
        String [] array = new String [] { "Item 1", "Item 2", "  
Item 3" };  
        JList<String> jList = new JList<String>(array);  
        add(jList);  
    }  
}
```

Criando Listas - Implementação

- O resultado obtido é:



```
import javax.swing.JFrame;

public class Principal extends JFrame {

    public Principal() {
        setTitle("Exemplo");
        setBounds(50, 50, 300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        Principal exemplo = new Principal();
        exemplo.add(new ExemploJList());
        exemplo.setVisible(true);

    }
}
```

Seções

Introdução

Janelas

Painéis

Criando bordas

Textos e Imagens

Botões

Tipos de Layouts

Entradas de Texto

CheckBox

ComboBox e Listas

Tabelas

Janelas de opções

Criando Tabelas

- JTables são um dos componentes mais complexos que existem;
- Mas isso não significa que são os mais difíceis;
- Para criar tabelas, precisaremos estender a classe `AbstractTableModel`;
- Esta irá servir de modelo para a `JTable` desenhar a tabela na tela;
- A classe `AbstractTableModel` pede a implementação dos seguintes métodos:
 - **`public int getColumnCount()`**: retorna a quantidade de colunas da tabela;
 - **`public int getRowCount()`** retorna a quantidade de linhas da tabela;
 - **`public Object getValueAt(int linha, int coluna)`**: retorna o valor a ser desenhado na célula (linha, coluna) da tabela;
- É recomendável sobrescrever o método **`public String getColumnName(int coluna)`** para que as colunas tenham seu título definido;
- Note que esse método só será usado pela tabela caso ela esteja dentro de um `JScrollPane`, caso contrário, será necessário usar outro artifício para colocar nome nas colunas.

Criando Tabelas

- Para redesenhar a tabela a cada vez que ela for modificada, utilize o método **fireTableStructureChanged()**;
- Vamos ver um exemplo;
- Primeiro vamos criar uma classe chamada Cidade:

Criando Tabelas - Implementação

- Vamos utilizar a classe Cidade para representar uma tabela;
- Cada linha da tabela será um objeto do tipo cidade;

```
public class Cidade {  
  
    private String nome;  
    private String estado;  
  
    public Cidade(String nome, String estado) {  
        this.nome = nome;  
        this.estado = estado;  
    }  
  
    public String getNome() {  
        return this.nome;  
    }  
  
    public String getEstado() {  
        return this.estado;  
    }  
  
}
```

Criando Tabelas - Implementação

- Agora vamos estender a classe `AbstractTableModel`;
- Teremos um atributo nessa classe que será uma `List` de cidades;

```
import java.util.LinkedList;
import java.util.List;

import javax.swing.table.AbstractTableModel;

public class Tabela extends AbstractTableModel {
    private List<Cidade> cidades = new LinkedList<Cidade>();
}
```


Criando Tabelas - Implementação

- Como cada cidade só possui dois atributos, teremos duas colunas apenas;
- O método **getColumnCount()** vai retornar sempre 2;

```
import java.util.LinkedList;
import java.util.List;

import javax.swing.table.AbstractTableModel;

public class Tabela extends AbstractTableModel {

    ...

    @Override
    public int getColumnCount() {
        return 2;
    }
}
```

Criando Tabelas - Implementação

- As linhas vão representar os objetos da list;
- Logo, no método **getRowCount()** vamos retornar quantos objetos a lista tem;

```
import java.util.LinkedList;
import java.util.List;

import javax.swing.table.AbstractTableModel;

public class Tabela extends AbstractTableModel {

    ...

    @Override
    public int getRowCount() {
        return cidades.size();
    }
}
```

Criando Tabelas - Implementação

- E no método **getValueAt()**, vamos retornar os atributos para cada célula da tabela;
- Para isso, precisaremos acessar atributo a atributo das cidades;
- Como as colunas são fixas e as linhas variam, vamos fazer um switch case nas colunas;

```
import java.util.LinkedList;
import java.util.List;

import javax.swing.table.AbstractTableModel;

public class Tabela extends AbstractTableModel {

    ...

    @Override
    public Object getValueAt(int linha, int coluna) {
        switch (coluna) {
            case 0: // Coluna 0 = Nome da Cidade
                return cidades.get(linha).getNome();
            case 1: // Coluna 1 = Nome do Estado
                return cidades.get(linha).getEstado();
            default:
                throw new IllegalArgumentException();
        }
    }
}
```

Criando Tabelas - Implementação

- Agora vamos criar o método que vai retornar o nome das colunas;
- Para isso, basta um switch case para a coluna;

```
import java.util.LinkedList;
import java.util.List;

import javax.swing.table.AbstractTableModel;

public class Tabela extends AbstractTableModel {
    ...

    public String getColumnName(int coluna) {
        switch (coluna) {
            case 0:
                return "Nome da Cidade";
            case 1:
                return "Estado";
            default:
                throw new IllegalArgumentException();
        }
    }
}
```

Criando Tabelas - Implementação

- Precisamos de um método para adicionar cidades também;
- Esse método irá redesenhar a tabela toda vez que uma nova cidade for inserida na lista;

```
import java.util.LinkedList;
import java.util.List;

import javax.swing.table.AbstractTableModel;

public class Tabela extends AbstractTableModel {

    ...

    public void adicionarCidade(Cidade cidade) {
        cidades.add(cidade);
        this.fireTableStructureChanged();
    }
}
```

Criandos Tabelas - Implementação

- Agora vamos criar um Painel e colocar nossa tabela;
- Mas vamos utilizar um JScrollPane ao invés de um JPanel;
- O JScrollPane permite o usuário descer a tela conforme ela cresce;
- Para isso, o componente que irá “crescer” é adicionado usando o método **setViewportView()**;
- No caso, o nosso componente será um JTable;
- No construtor do JTable passaremos uma instância da classe que estende o AbstractTableModel;

Criando Tabelas - Implementação

- Primeiro criamos o modelo (a classe que estende o AbstractTableModel) e depois a JTable;
- Manteremos o modelo como uma variável para podermos acessar os métodos dela;

```
import javax.swing.JScrollPane;  
import javax.swing.JTable;  
  
public class ExemploJTable extends JScrollPane {  
    public ExemploJTable() {  
        Tabela cidades = new Tabela();  
        JTable table = new JTable(cidades);  
        setViewportView(table);  
    }  
}
```

Criando Tabelas - Implementação

- E então adicionamos três cidades;
- O método **adicionar()** já realiza todo o trabalho de redesenhar a tabela;

```
import javax.swing.JScrollPane;  
import javax.swing.JTable;  
  
public class ExemploJTable extends JScrollPane {  
    public ExemploJTable() {  
        Tabela cidades = new Tabela();  
        JTable table = new JTable(cidades);  
        setViewportView(table);  
        cidades.adicionarCidade(new Cidade("Joinville", "Santa  
Catarina"));  
        cidades.adicionarCidade(new Cidade("Curitiba", "Parana"));  
        cidades.adicionarCidade(new Cidade("Florianopolis", "Santa  
Catarina"));  
    }  
}
```


Criando Tabelas - Implementação

- Aqui está o resultado:



A screenshot of a Java Swing window titled "Exemplo". Inside the window is a table with two columns: "Nome da Cidade" and "Estado". The table contains three rows of data: Joinville (Santa Catarina), Curitiba (Parana), and Florianopolis (Santa Catarina). Below the table is a large, empty rectangular area.

Nome da Cidade	Estado
Joinville	Santa Catarina
Curitiba	Parana
Florianopolis	Santa Catarina

```
import javax.swing.JFrame;

public class Principal extends JFrame {

    public Principal() {
        setTitle("Exemplo");
        setBounds(50, 50, 300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        Principal exemplo = new Principal();
        exemplo.add(new ExemploJTable());
        exemplo.setVisible(true);

    }
}
```

Seções

Introdução

Janelas

Painéis

Criando bordas

Textos e Imagens

Botões

Tipos de Layouts

Entradas de Texto

CheckBox

ComboBox e Listas

Tabelas

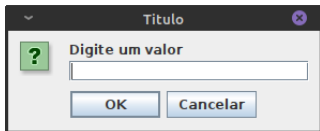
Janelas de opções

Criando Janelas de opções e alertas

- Por fim, vamos ver uma forma de tratar exceções e realizar a entrada de dados utilizando o JOptionPane;
- A classe JOptionPane possui vários métodos estáticos, vamos apresentar alguns a seguir;
- Utilizaremos apenas os que exibem uma pequena janela;

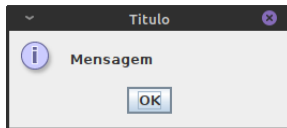
Criando Janelas de opções e alertas - InputDialog

- Para criar diálogos de entrada de dados:
 - **JOptionPane.showInputDialog(String mensagem);**
 - **JOptionPane.showInputDialog(Component componente, String mensagem, String titulo, int enumTipo);**
- O enumTipo pode ser:
 - JOptionPane.ERROR_MESSAGE;
 - JOptionPane.INFORMATION;
 - JOptionPane.WARNING_MESSAGE;
 - JOptionPane.QUESTION_MESSAGE;
 - JOptionPane.PLAIN_MESSAGE;
- A chamada desse método abre uma janela de diálogo e retorna uma String quando o usuário a fecha;



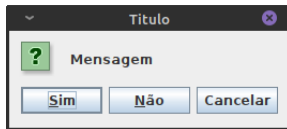
Criando Janelas de opções e alertas - MatDialog

- Para criar diálogos de mensagem:
 - **JOptionPane.showMessageDialog(String mensagem);**
 - **JOptionPane.showMessageDialog(Component componente, String mensagem, String titulo, int enumTipo);**
- O enumTipo é dos mesmos tipos apresentados no slide anteriormente;
- A chamada desse método abre uma janela de diálogo;



Criando Janelas de opções e alertas - ConfirmDialog

- Para criar diálogos de confirmação:
 - **JOptionPane.showConfirmDialog(String mensagem);**
 - **JOptionPane.showConfirmDialog(Component componente, String mensagem, String titulo, int enumTipo);**
- O enumTipo é dos mesmos tipos apresentados no slide anteriormente;
- A chamada desse método abre uma janela de confirmação;



- Quando fechada a janela pode retornar os seguintes valores:
 - **-1:** caso o usuário feche a janela
 - **0:** caso o usuário clique em Sim
 - **1:** caso o usuário clique em Não
 - **2:** caso o usuário clique em Cancelar

Criando Janelas de opções e alertas - Exemplo

- Vamos fazer um exemplo de um botão que ao pressionado abre uma janela de mensagem;
- Para isso vamos criar um Painel contendo um JButton;
- Sua ação irá chamar o método **showMessageDialog()**

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

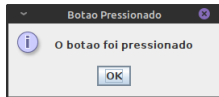
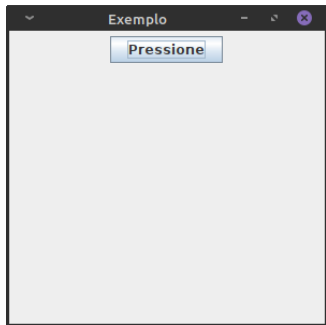
import javax.swing.JButton;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class ExemploJOptionPane extends JPanel {


    public ExemploJOptionPane() {
        JButton button = new JButton("Pressione");
        add(button);
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent arg0) {
                JOptionPane.showMessageDialog(null, "O botao foi
pressionado", "Botao Pressionado",JOptionPane.INFORMATION_MESSAGE);
            }
        });
    }
}
```

Criando Janelas de opções e alertas - Exemplo

- E o resultado é:



```
import javax.swing.JFrame;  
  
public class Principal extends JFrame {  
  
    public Principal() {  
        setTitle("Exemplo");  
        setBounds(50, 50, 300, 300);  
        setDefaultCloseOperation(JFrame.  
EXIT_ON_CLOSE);  
    }  
  
    public static void main(String[] args) {  
  
        Principal exemplo = new Principal();  
        exemplo.add(new ExemploJOptionPane());  
        exemplo.setVisible(true);  
  
    }  
}
```


 KUWAKI, V. T. F. Modelo de slides udesc lattex. In: . [S.l.]: Disponível em: <<https://github.com/takeofriedrich/slidesUdescLattex>>. Acesso em: 24 jan. 2020.

Duvidas:
Vinicius Takeo Friedrich Kuwaki
vtkwki@gmail.com
github.com/takeofriedrich