

**Curso**

# Pacotes, Lambdas, Streams, Interfaces Gráficas

---

Atualizado até o Java 21 &  
Eclipse 2023-09



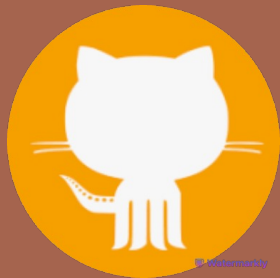
**Prof. Msc. Antonio B. C. Sampaio Jr**  
engenheiro de software & professor

@abctreinamentos  
@amazoncodebr

[www.abctreinamentos.com.br](http://www.abctreinamentos.com.br)  
[www.amazoncode.com.br](http://www.amazoncode.com.br)



# REPOSITÓRIO GITHUB



antonio-sampaio-jr / **pacotes-lambdas-streams**

# CONTEÚDO PROGRAMÁTICO



- UNIDADE 1 – PACOTES, ERROS E EXCEÇÕES
- UNIDADE 2 – ANOTAÇÕES E ENTRADA/SAÍDA
- UNIDADE 3 – GENÉRICOS
- UNIDADE 4 – FRAMEWORK COLLECTIONS
- UNIDADE 5 – NOVIDADES JAVA 8
- UNIDADE 6 - APLICAÇÕES GRÁFICAS EM JAVA
  - Bibliotecas Gráficas

## CONTEÚDO PROGRAMÁTICO



- UNIDADE 6 - APLICAÇÕES GRÁFICAS EM JAVA (Continuação)
  - Componentes Swing: JLabel, JTextField, JPasswordField, JTextArea e JButton
  - Componentes Swing: JCheckBox, JRadioButton, JComboBox, JMenuBar, Jmenu, JMenuitem e JScrollBar
  - Caixas de Diálogo
  - Gerenciadores de Layout
  - Eventos

# CONTEÚDO PROGRAMÁTICO



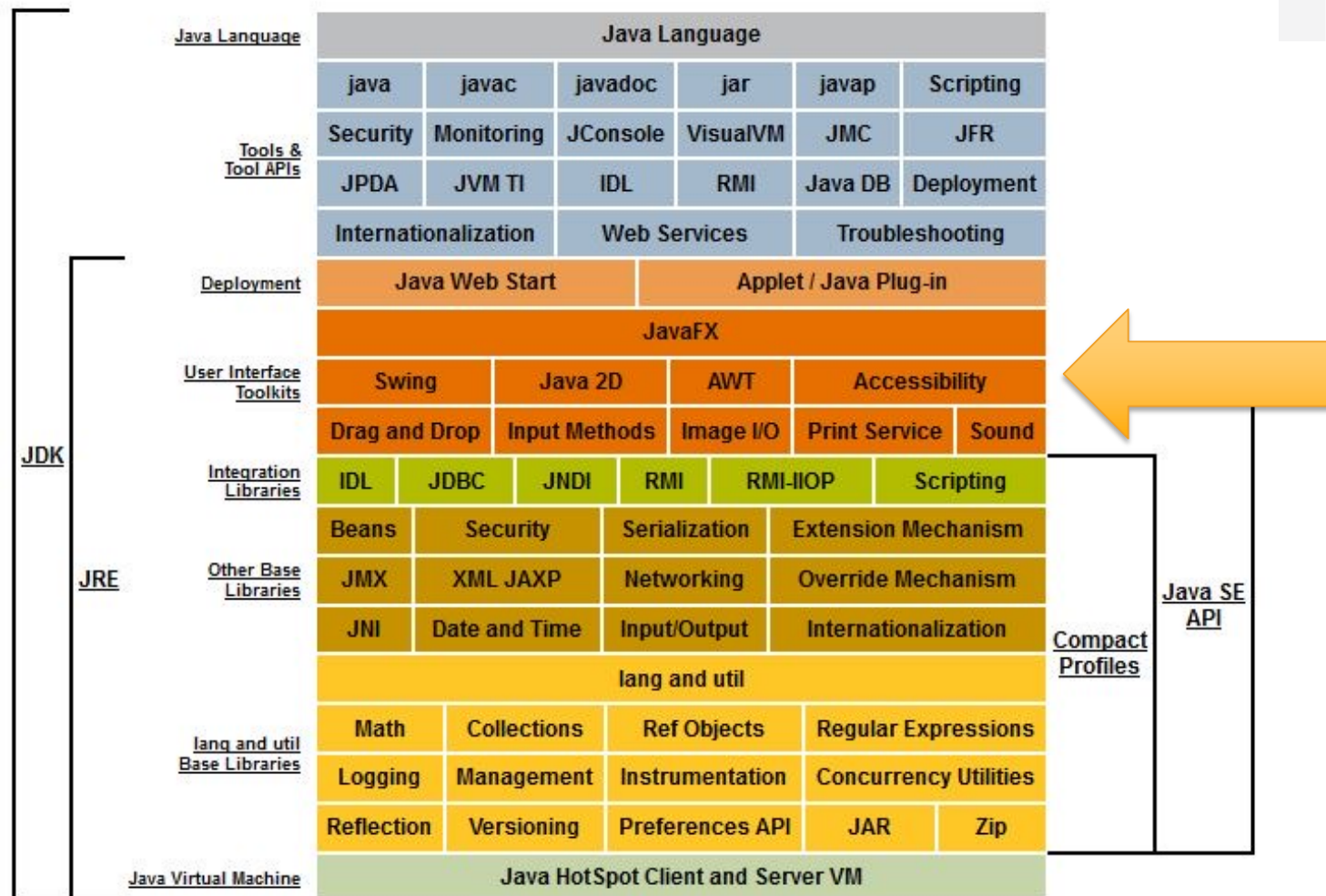
- UNIDADE 6 - APLICAÇÕES GRÁFICAS EM  
JAVA (Continuação)
  - Look and Feel
  - Editor Visual [NOVO]
  - JavaFX
  - Projeto OpenJFX [NOVO]
  - Conclusão [NOVO]



# Bibliotecas Gráficas

# Bibliotecas Gráficas

- No Java 8 SE API, os pacotes gráficos estão destacados (em cor laranja) na área conhecida como *User Interface Toolkit*.



# Bibliotecas Gráficas

- Dos pacotes destacados, os mais importantes para o nosso estudo estão listados abaixo:
- **AWT (Abstract Window Toolkit)** - O AWT é composto de classes que definem diversos componentes necessários na construção de uma interface gráfica. O AWT foi desenvolvido para as plataformas JDK 1.0 e 1.1. Os seus principais pacotes são: **java.awt.\*** e **java.awt.event.\***
- **Swing** – Surgiu no Java 2 como evolução do AWT. Os seus principais pacotes são: **javax.swing.\*** e **javax.swing.event.\***
- **JavaFx** – É a API Java mais avançada para a criação de aplicações RIA (*Rich Internet Application*). Sua forma de programar facilita muito a vida do desenvolvedor. É totalmente independente do AWT e do Swing.
  - O JavaFX 8 está incluído no JDK 8 e é a recomendação oficial para a criação de aplicações gráficas com o Java 8.

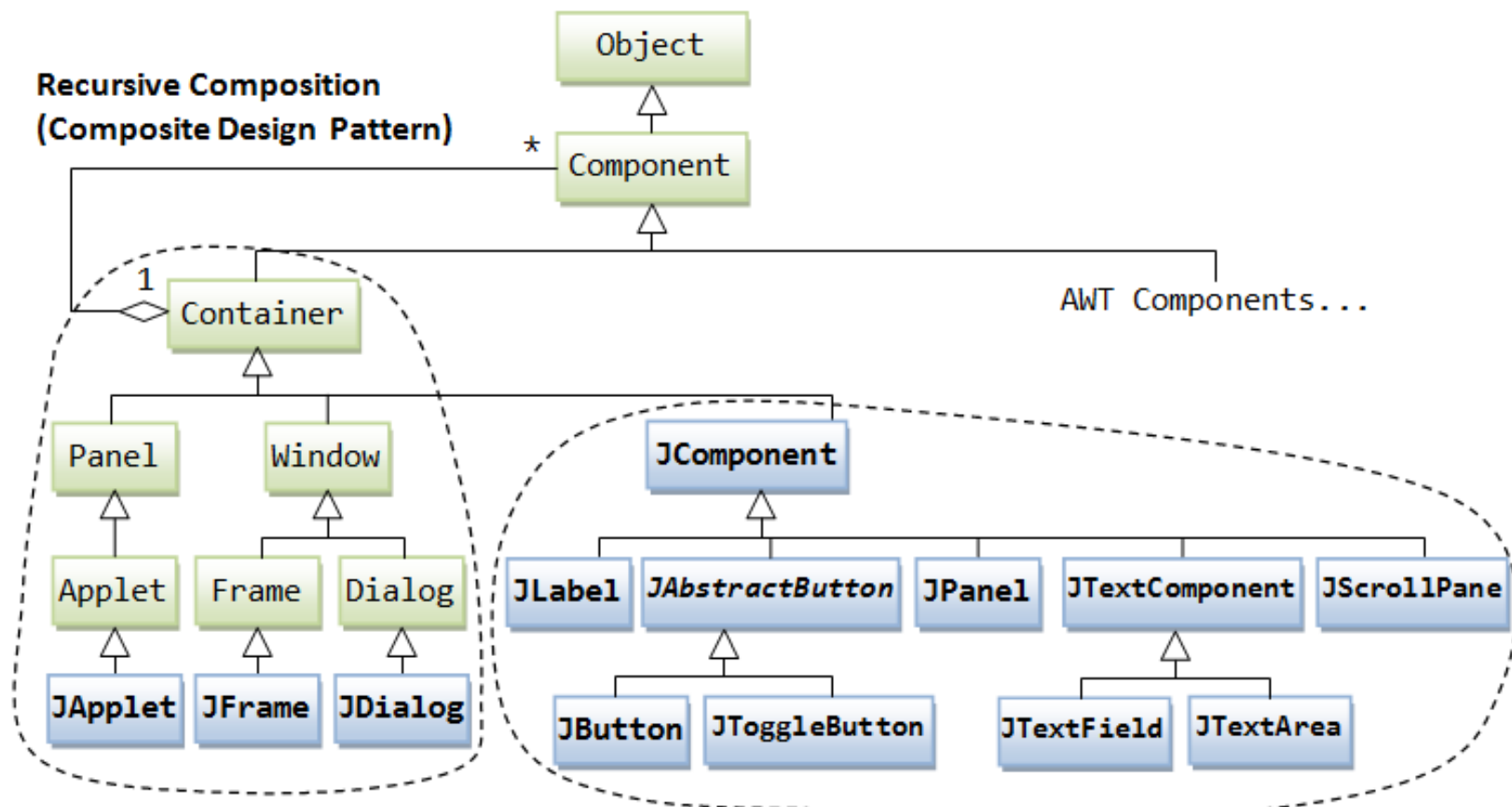


# Bibliotecas Gráficas

- As bibliotecas gráficas são bastante simples no que diz respeito a conceitos necessários para usá-las.
- **AWT** e **Swing** são bibliotecas gráficas oficiais incluídas em qualquer JDK. Além dessas, existem algumas outras bibliotecas de terceiros, sendo o SWT (*Standard Widget toolkit*) a mais conhecida.
- Neste curso, serão estudadas as tecnologias **Swing** e **JavaFx**.

# Arquitetura Swing

- Baseada no padrão de projeto (*design pattern*) **Composite**, divide as classes em dois grupos: **containers** e **componentes**. Um **container** é utilizado para agrupar um ou vários **componentes** e outros **containers**.



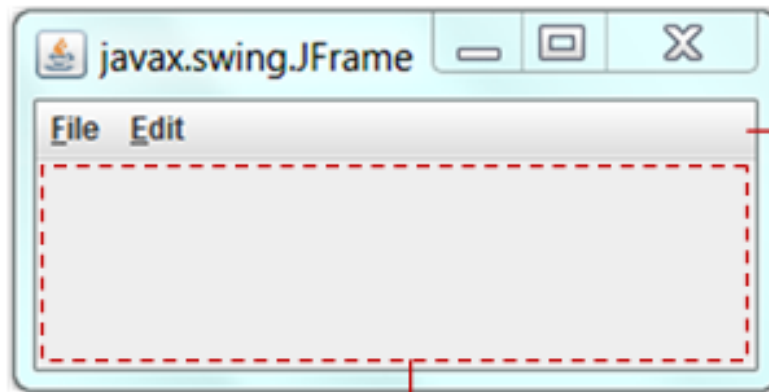
# Elementos Swing

- A biblioteca gráfica Swing está organizada em **Container**, **Componentes** e **Gerenciadores de Layout**.
- **Container** (**JFrame**, **JDialog** e **JApplet**) é o local onde são adicionados os componentes e outros containers.
- **Componentes** (**JButton**, **JLabel**, **JScrollBar**, **JOptionPane**, etc.) são os elementos gráficos adicionados a um Container.
- **Gerenciadores de Layout** (**FlowLayout**, **BorderLayout**, etc) são os elementos responsáveis pelo posicionamento dos componentes adicionados a um determinado Container
- A seguir, cada um desses elementos será estudado de forma detalhada.

# Container Swing

- São três os containers Swing: **JFrame**, **JDialog** e **JApplet**
- O **JFrame** é utilizado para criar aplicações gráficas baseadas em “Janelas” que possuem: ícone, título, botões de minimizar e maximizar, menu, etc., conforme ilustrado pela imagem abaixo.

`javax.swing.JFrame`



Menu Bar  
(Optional)

**Content Pane**

© Chua Hock-Chuan

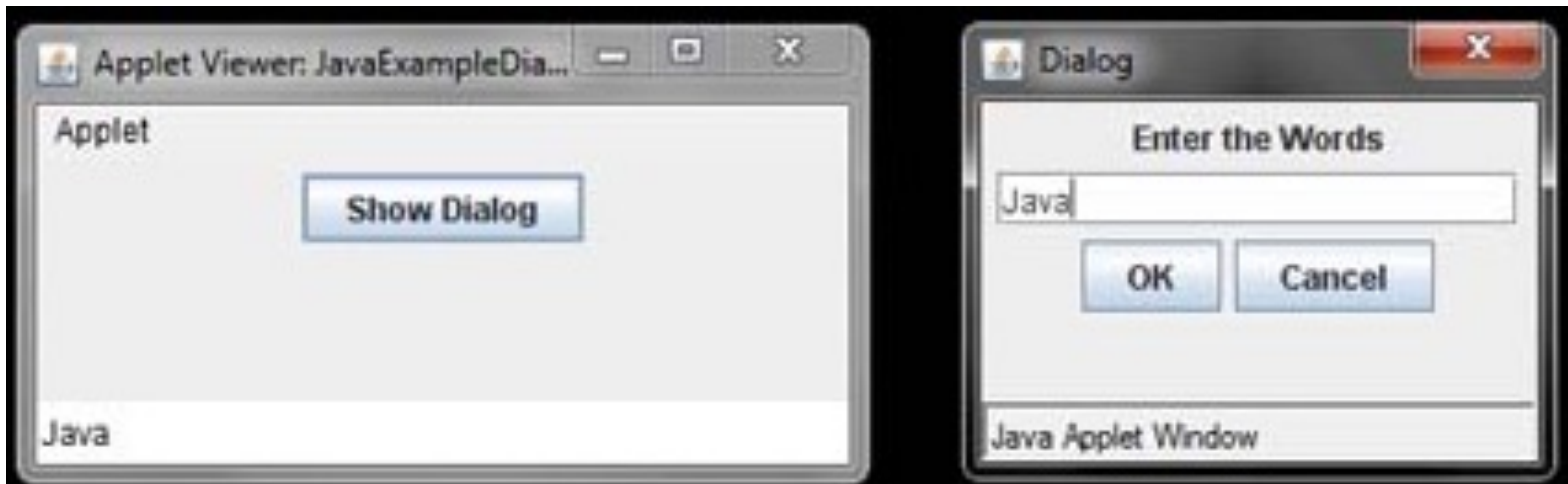
```
Container cp = aJFrame.getContentPane();  
aJFrame.setContentPane(aPanel);
```

# Container Swing

- O **Content-Pane** é um container secundário utilizado para adicionar componentes e organizar a sua disposição de layout.

```
Container ct = jframe.getContentPane();  
ct.setLayout(new FlowLayout());  
ct.add(new JButton("OK"));
```

- O **JDialog** é utilizado como uma janela secundária de pop-up. Já o **JApplet** é utilizado para criar Applets em uma página Web.



# Container Swing

- Exemplo de um JFrame e de um JApplet

```
import javax.swing.*;

public class ContentApp {
    public static void main(String args[]) {
        JFrame frame = new JFrame("Primeira Aplicação");
        frame.setSize(350,250);
        frame.setVisible(true);
    }
}
```

```
import javax.swing.*;

public class JAppletApp extends JApplet {
    public void init()
    {
        JOptionPane.showMessageDialog(null,"Curso de Java");
    }
}
```

# Container Swing


- **PRINCIPAIS MÉTODOS**

- **setSize(int,int)** - define a altura e a largura da janela.
- **getContentPane()** - acessa a “área de uso” do Container.
- **setLayout(LayoutManager)** - define o gerenciador de layout utilizado.
- **setLocation(int,int)** – define a posição da janela na tela (x,y).
- **setTitle(String)** – alterar o título da janela.
- **setVisible(boolean)** - exibe a janela.
- **add(Component,int)** - adiciona um componente na “área de uso” do Container.

# Exercícios

- 1) [CESPE - 2010 - TRE-BA] Em programação orientada a objetos, o pacote tem como função agrupar classes dentro de um grupo. Em Java, o pacote Swing (javax.swing) é composto de várias classes para a implementação de interfaces gráficas em desktop.  
  
A) Certo   B) Errado
- 2) Implementar as classes **ContentApp** e **JAppletApp**.





# **Componentes Swing: JLabel, JTextField, JPasswordField, JTextArea e JButton**

# Componentes Swing

- São os elementos gráficos da interface (botões, textos, rótulos, caixas de checagem, listas, etc.) que estão definidos no pacote **javax.swing** e cujo prefixo é “J”.
- Abaixo são apresentados os principais componentes Swing:



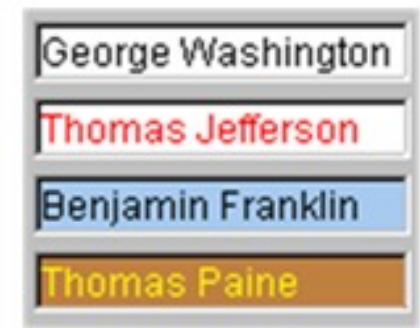
Buttons



Combo Box



List



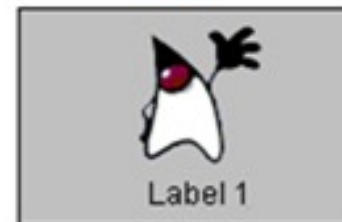
TextField



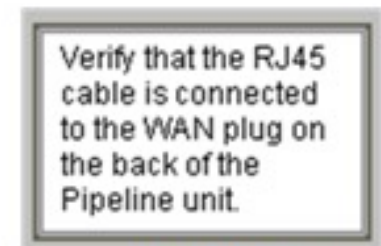
Slider



Menu



Label



Text Area

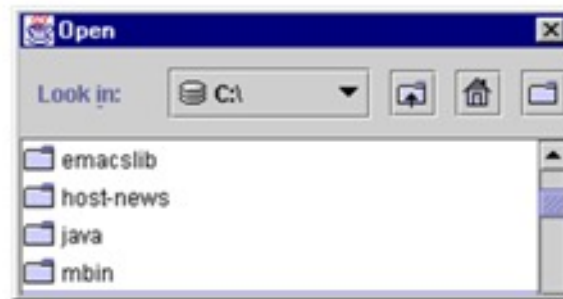
# Componentes Swing



Tool Tip



Progress Bar



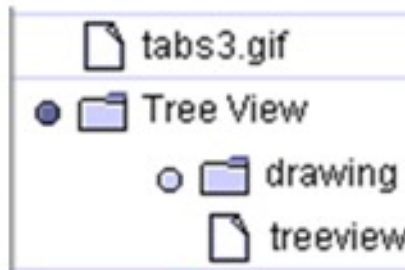
File Chooser



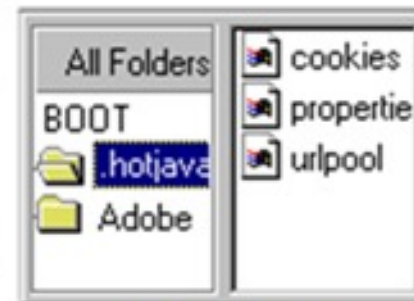
Color Chooser

First Na...	Last Name
Mark	Andrews
Tom	Ball
Alan	Chung
Jeff	Dinkins

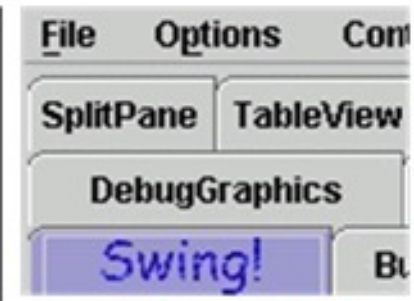
Table



Tree



Split Pane



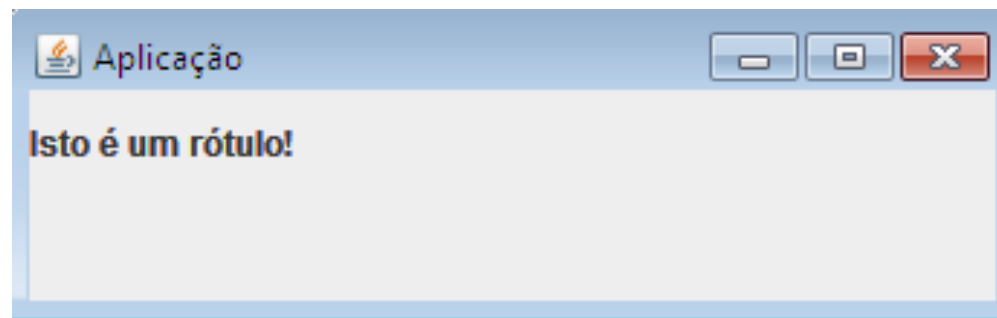
Tabbed Pane

- A seguir, os principais componentes Swing são apresentados:

# JLabel

- É o componente que representa um rótulo de texto.

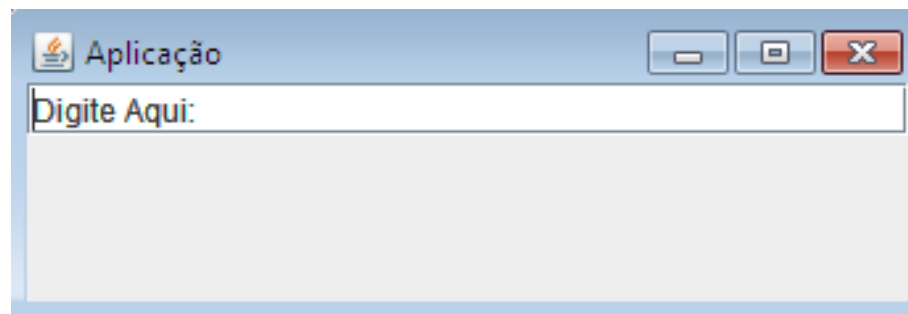
```
import javax.swing.*;import java.awt.*;  
public class LabelApp{  
    public static void main(String args[]){  
        JFrame frame = new JFrame("Aplicação");  
        frame.setSize(350,250);  
        Container cont = frame.getContentPane();  
        cont.setLayout(new GridLayout(2,1));  
        cont.add(new JLabel("Isto é um rótulo!"));  
        frame.setVisible(true);    }  
}
```



# JTextField

- É o componente que representa um campo de texto.

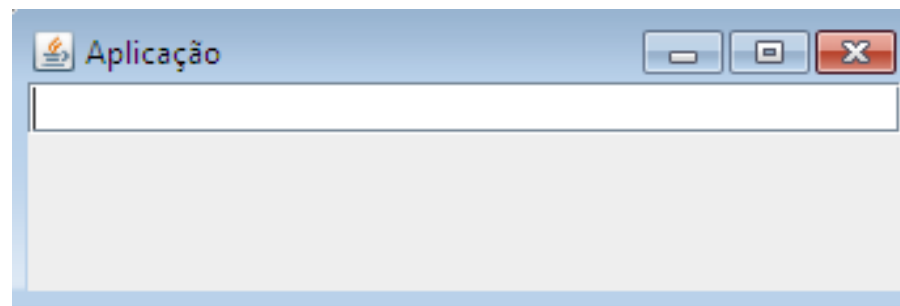
```
import javax.swing.*;import java.awt.*;
public class JTFApp{
    public static void main(String args[]) {
        JFrame frame = new JFrame("Aplicação");
        frame.setSize(350,250);
        Container cont = frame.getContentPane();
        JTextField tf = new JTextField("Digite Aqui:");
        cont.add(tf,BorderLayout.NORTH);
        frame.setVisible(true);}
}
```



# JPasswordField

- É o componente que representa um campo de texto protegido.

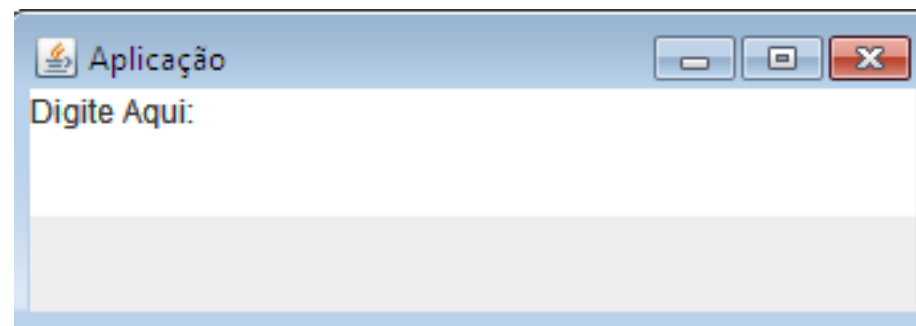
```
import javax.swing.*;import java.awt.*;  
public class JPFAApp{  
    public static void main(String args[]) {  
        JFrame frame = new JFrame("Aplicação");  
        frame.setSize(350,250);  
        Container cont = frame.getContentPane();  
        JPasswordField tf = new JPasswordField();  
        cont.add(tf,BorderLayout.NORTH);  
        frame.setVisible(true);}  
}
```



# JTextArea

- É o componente que o usuário pode informar várias linhas de texto.

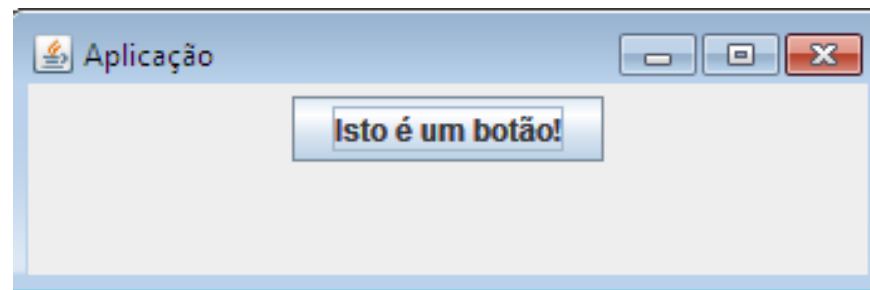
```
import javax.swing.*;import java.awt.*;
public class JTAApp{
    public static void main(String args[]) {
        JFrame frame = new JFrame("Aplicação");
        frame.setSize(350,250);
        Container cont = frame.getContentPane();
        JTextArea ta = new JTextArea("Digite Aqui:");
        cont.add(ta,BorderLayout.NORTH);
        frame.setVisible(true);}
}
```



# JButton

- É o componente que representa um botão destinado a executar uma ação.

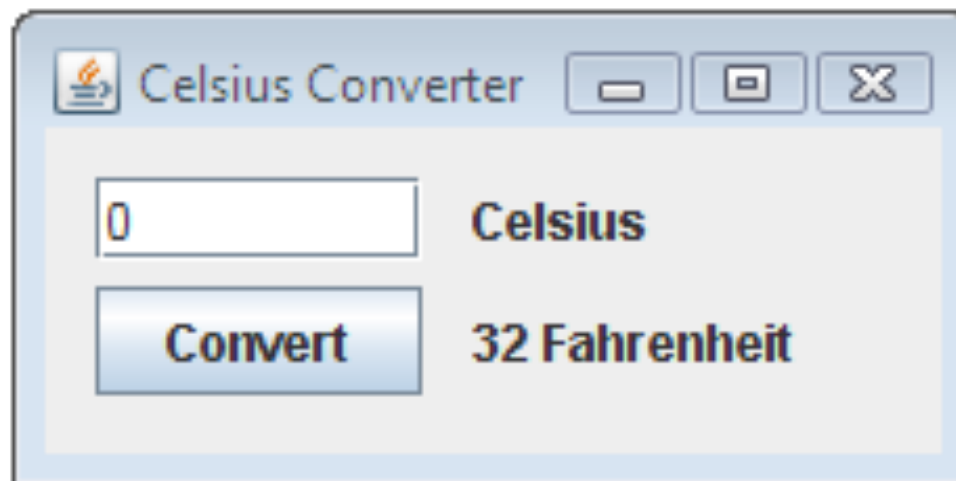
```
import javax.swing.*;import java.awt.*;  
public class ButtonApp{  
    public static void main(String args[]){  
        JFrame frame = new JFrame("Aplicação");  
        frame.setSize(350,250);  
        Container cont = frame.getContentPane();  
        cont.setLayout(new FlowLayout());  
        cont.add(new JButton("Isto é um botão!"));  
        frame.setVisible(true);    }  
}
```






# Exercícios

- 1) Implementar as classes **LabelApp**, **JTFApp**, **JPFAApp**, **JTAApp** e **ButtonApp**.
- 2) Escreva a classe **ConverterApp** que represente a interface gráfica abaixo.





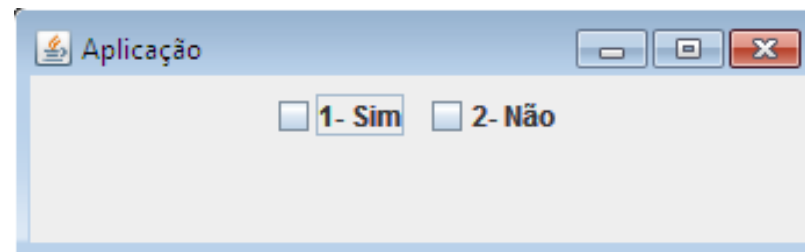
# **Componentes Swing:**

- JCheckBox,**
- JRadioButton,**
- JComboBox,**
- JMenuBar, JMenu,**
- JMenuItem e**
- JScrollBar**

# JCheckBox

- É o componente que representa uma caixa de seleção, cujos elementos podem ser selecionados ou não.

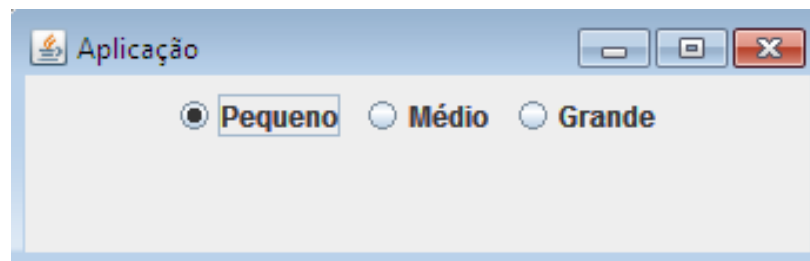
```
import javax.swing.*;import java.awt.*;  
public class CheckApp {  
    public static void main(String args[]){  
        JFrame f = new JFrame("Aplicação");  
        f.setSize(350,250);  
        Container cont = f.getContentPane();  
        cont.setLayout(new FlowLayout());  
        cont.add(new JCheckBox("1- Sim"));  
        cont.add(new JCheckBox("2- Não"));  
        f.setVisible(true);  }}
```



# JRadioButton

- É o componente que representa uma lista única de opções.

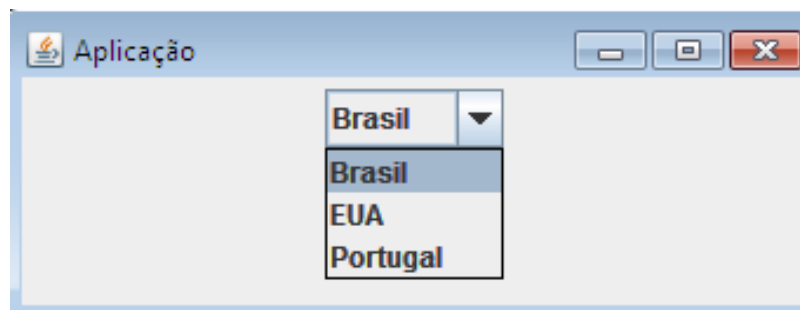
```
{ JFrame f = new JFrame("Aplicação");  
  f.setSize(350,250);  
  Container cont = f.getContentPane();  
  cont.setLayout(new FlowLayout());  
  ButtonGroup btg = new ButtonGroup();  
  JRadioButton rb = new JRadioButton("Pequeno",true);  
  btg.add(rb);  cont.add(rb);  
  rb = new JRadioButton("Médio");  
  btg.add(rb);  cont.add(rb);  
  rb = new JRadioButton("Grande");  
  btg.add(rb);  cont.add(rb); f.setVisible(true); }
```



# JComboBox

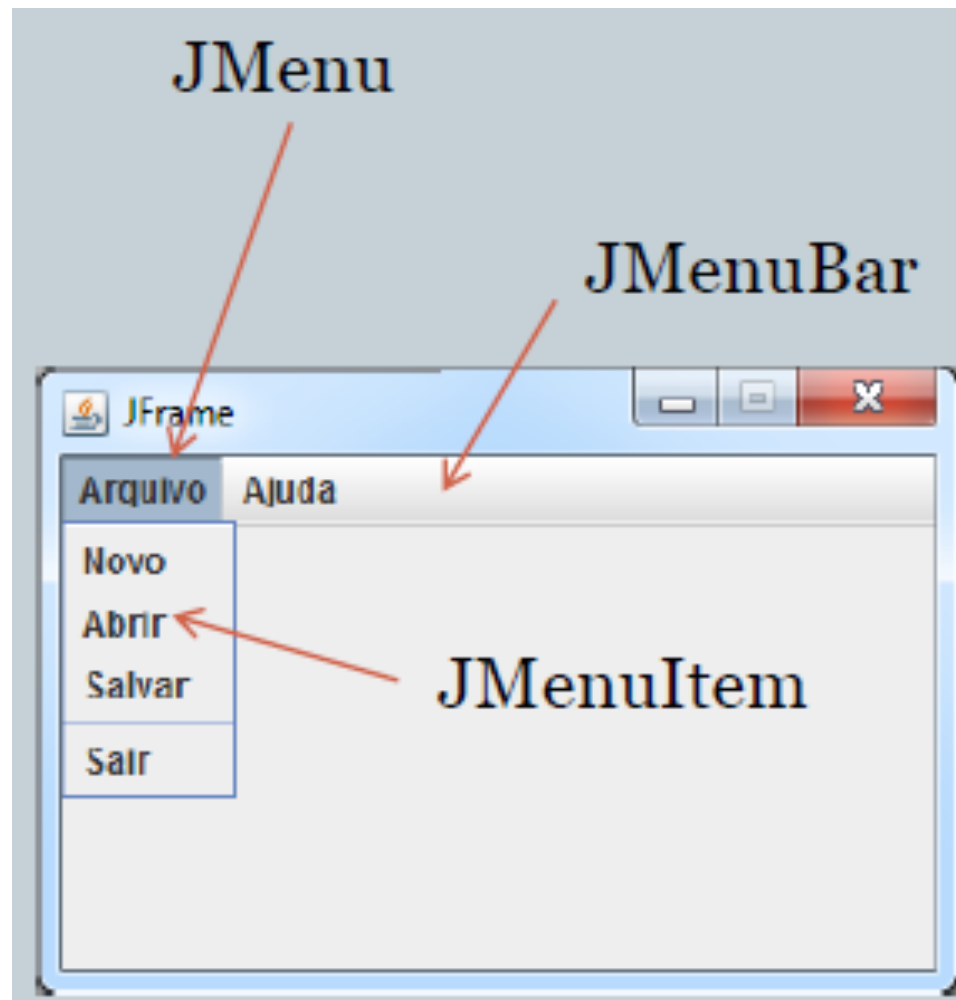
- É o componente que representa uma lista de elementos selecionáveis.

```
public static void main(String args[]) {  
    JFrame f = new JFrame("Aplicação");  
    f.setSize(350,250);  
    Container cont = f.getContentPane();  
    cont.setLayout(new FlowLayout());  
    String[] init = {"Brasil","EUA","Portugal"};  
    JComboBox<String> combo = new JComboBox<>(init);  
    cont.add(combo);  
    f.setVisible(true);  
}
```



# JMenuBar, JMenu JMenuItem

- São componentes que representam um menu de opções.



# JMenuBar, JMenu JMenuItem

- São componentes que representam um menu de opções.

```
public static void main(String args[]) {  
    JFrame frame = new JFrame("JFrame");  
    frame.setSize(350,250);  
    Container cont = f.getContentPane();  
    JMenuBar barra = new JMenuBar();  
    JMenu m1 = new JMenu("Arquivo");  
    JMenuItem m11 = new JMenuItem("Novo");  
    JMenuItem m12 = new JMenuItem("Abrir");  
    JMenuItem m13 = new JMenuItem("Salvar");  
    JMenuItem m14 = new JMenuItem("Sair");  
    JMenu m2 = new JMenu("Ajuda");  
    JMenuItem m21 = new JMenuItem("Conteudo");  
    JMenuItem m22 = new JMenuItem("Sobre");  
}
```

# JMenuBar, JMenu JMenuItem

- São componentes que representam um menu de opções.

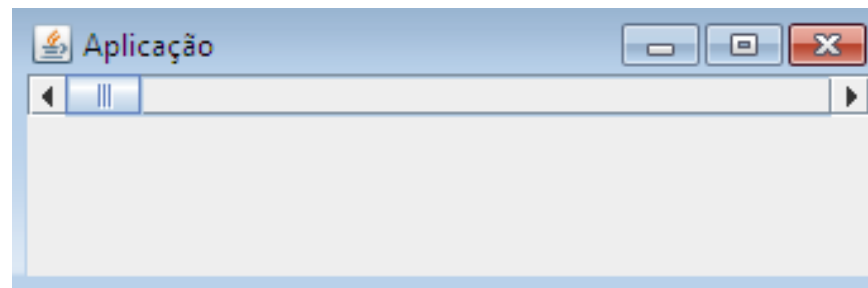
```
m1.add(m11) ;  
m1.add(m12) ;  
m1.add(m13) ;  
m1.addSeparator() ;  
m1.add(m14) ;  
m2.add(m21) ;  
m2.addSeparator() ;  
m2.add(m22) ;  
barra.add(m1) ;  
barra.add(m2) ;  
frame.setJMenuBar(barra) ;  
frame.setVisible(true) ;  
}
```



# JScrollBar

- É o componente que representa uma barra de rolagem.

```
import javax.swing.*;import java.awt.*;
public class ScrollApp{
    public static void main(String args[]) {
        JFrame f = new JFrame("Aplicação");
        f.setSize(350,250);
        Container cont = f.getContentPane();
        JScrollBar sbar = new JScrollBar(JScrollBar.HORIZONTAL);
        cont.add(sbar,BorderLayout.NORTH);
        f.setVisible(true);}
}
```

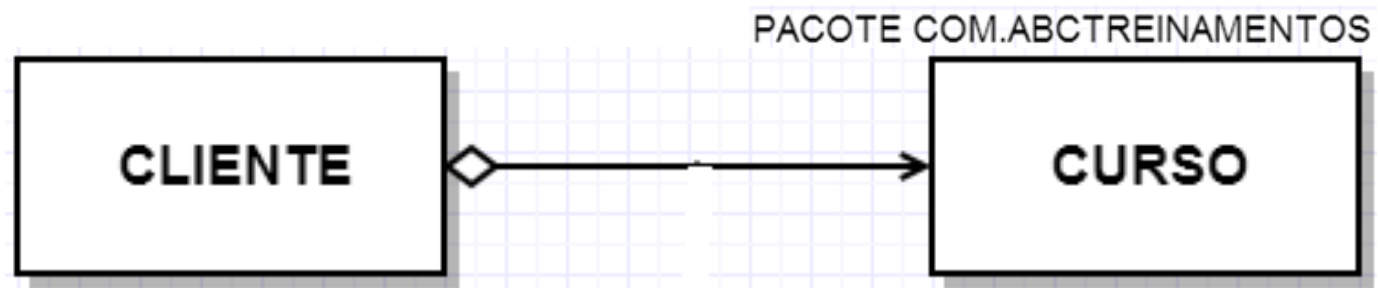


# Outros Componentes Swing

- Existem vários outros componentes Swing que poderão ser utilizados:
  - JFileChooser
  - JTable
  - JSlider
  - JProgressBar
  - JSpinner
  - JToolBar
  - JTabbedPane
  - JCheckBoxMenuItem
  - JRadioButtonMenuItem
  - JPopupMenu
  - JDesktopPane
  - JInternalFrame
  - Etc.

# Exercícios

- 1) Implemente as classes **CheckApp** (e as suas variações) e **ScrollApp**.
- 2) Escreva uma interface gráfica **LojaVirtual** que represente o diagrama de classes abaixo:





# Caixas de Diálogo

# Caixas de Diálogo

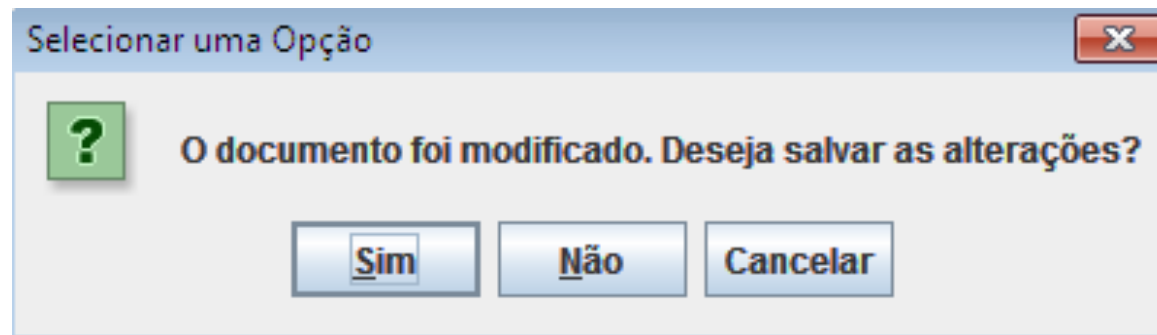
- Existem quatro caixas de diálogo padrão:
  - **ConfirmDialog**
  - **InputDialog**
  - **MessageDialog**
  - **OptionDialog**
- Os métodos que criam as caixas de diálogo são estáticos, não sendo necessário a criação de objetos para a sua utilização.



# ConfirmDialog

- É a caixa de diálogo que solicita a confirmação do usuário.

```
int resposta = JOptionPane.showConfirmDialog  
(null, "Documento modificado. Salvar as alterações?");
```

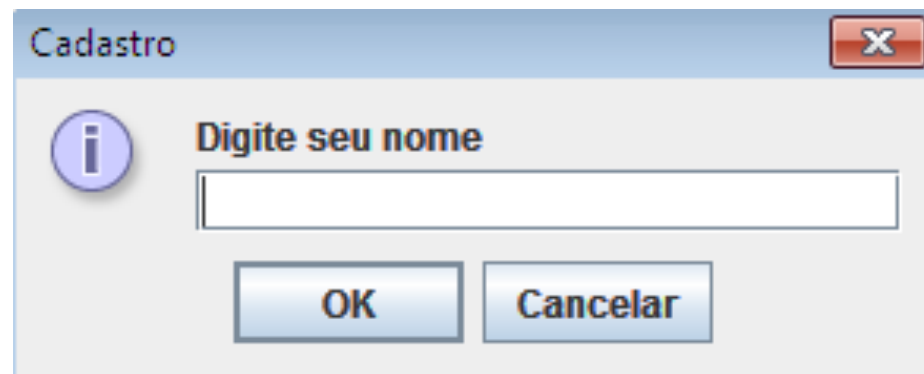


- O retorno da função é a opção escolhida pelo usuário, que pode ser: `JOptionPane.YES_OPTION(0)`, `JOptionPane.NO_OPTION(1)` ou `JOptionPane.CANCEL_OPTION(2)`.

# InputDialog

- É a caixa de diálogo que solicita a entrada de dados pelo usuário.

```
String nome = JOptionPane.showInputDialog(null,  
"Digite seu nome", "Cadastro", JOptionPane.INFORMATION_MESSAGE) ;
```

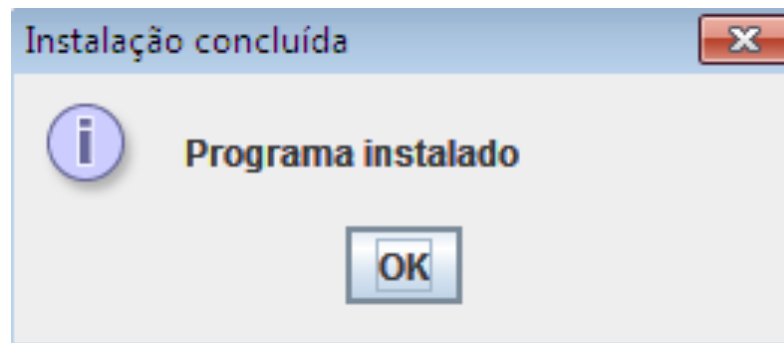


- O retorno da função é uma String contendo o texto digitado pelo usuário. Caso o usuário clique em **Cancel** ou feche a janela, é retornado **null**.

# MessageDialog

- É a caixa de diálogo que apresenta uma mensagem ao usuário.

```
JOptionPane.showMessageDialog(null, "Programa instalado",  
"Instalação concluída", JOptionPane.INFORMATION_MESSAGE);
```



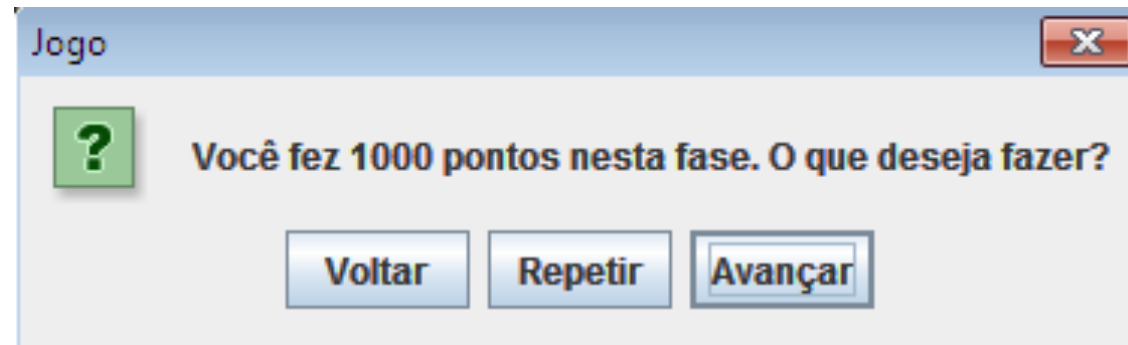
- O método **showMessageDialog(...)** não retorna valor.



# OptionDialog

- É a caixa de diálogo que apresenta uma mensagem ao usuário.

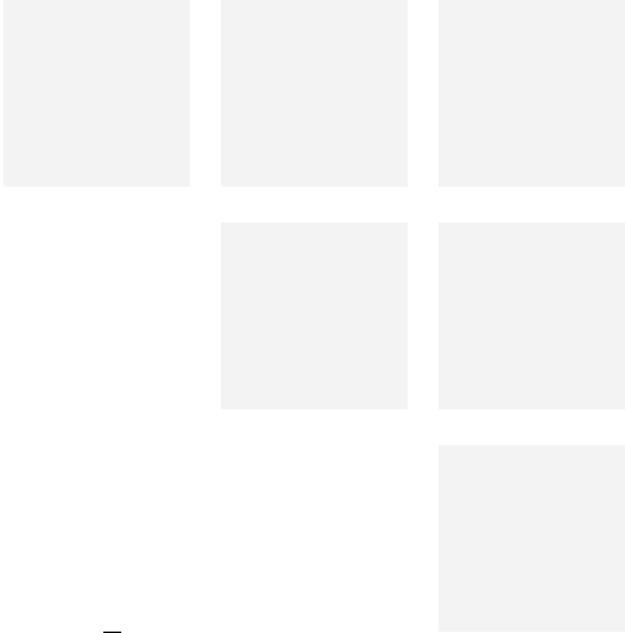

```
String[] opcoes = {"Voltar", "Repetir", "Avançar"};  
int resposta;  
resposta = JOptionPane.showOptionDialog(null,  
"Você fez 1000 pontos nesta fase. O que deseja fazer?",  
"Jogo", 0, JOptionPane.QUESTION_MESSAGE, null, opcoes, opcoes[2]);
```



- O retorno da função é um inteiro, indicando a posição do vetor relativo ao botão selecionado.

# Exercício

- 1) Escreva a classe **DialogApp** que implemente as caixas de diálogo padrão: **MessageDialog, InputDialog e ConfirmDialog**.



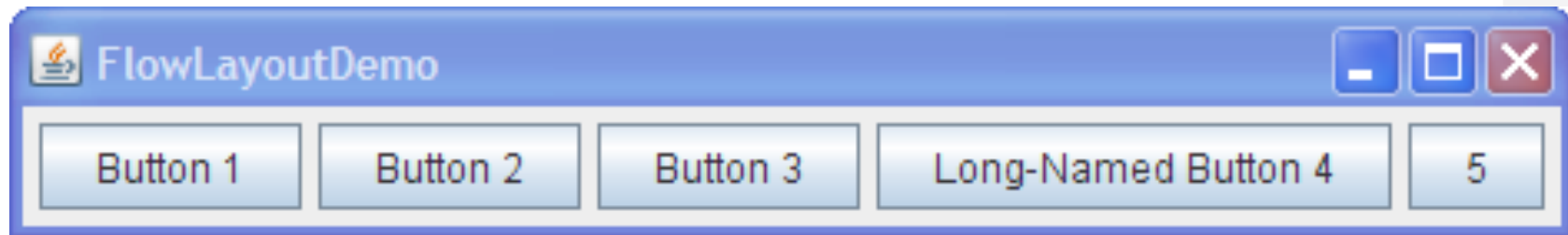
# Gerenciadores de Layout

# Gerenciadores de Layout

- São os responsáveis pela disposição dos componentes Swing na tela.
- O Java já vem com uma série de Layouts diferentes, que determinam como os elementos serão dispostos na tela, seus tamanhos preferenciais, como eles se comportarão quando a janela for redimensionada e muitos outros aspectos.
- Ao escrever uma aplicação Swing, deve ser indicado qual o gerenciador de Layout deverá ser utilizado. Por padrão, é utilizado o `FlowLayout` para os containers `JDialog` e `JApplet`. No caso do `JFrame` é utilizado o `BorderLayout`.
- Os gerenciadores de Layouts estão listados a seguir: **`BorderLayout`**, **`BoxLayout`**, **`CardLayout`**, **`FlowLayout`**, **`GridBagLayout`**, **`GridLayout`**, **`GroupLayout`** e **`SpringLayout`**.

# FlowLayout

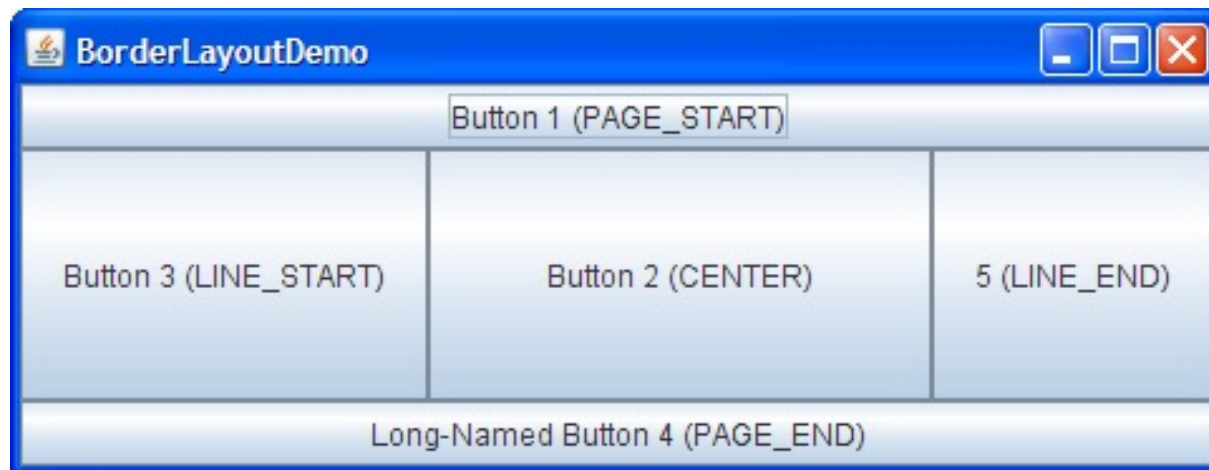
- É o gerenciador de layout padrão para cada JDialog e JApplet. Os componentes são colocados em uma única fila, iniciando uma nova caso o espaço horizontal não seja suficientemente grande.



```
...  
JFrame f = new JFrame("FlowLayoutDemo");  
Container cont = f.getContentPane();  
cont.setLayout(new FlowLayout());  
cont.add(new JButton("Button 1"));  
cont.add(new JButton("Button 2"));  
...
```

# BorderLayout

- É o gerenciador de layout padrão para cada JFrame. Os componentes são colocados em até cinco áreas: PAGE\_START, PAGE\_END, LINE\_START, LINE\_END e CENTER.

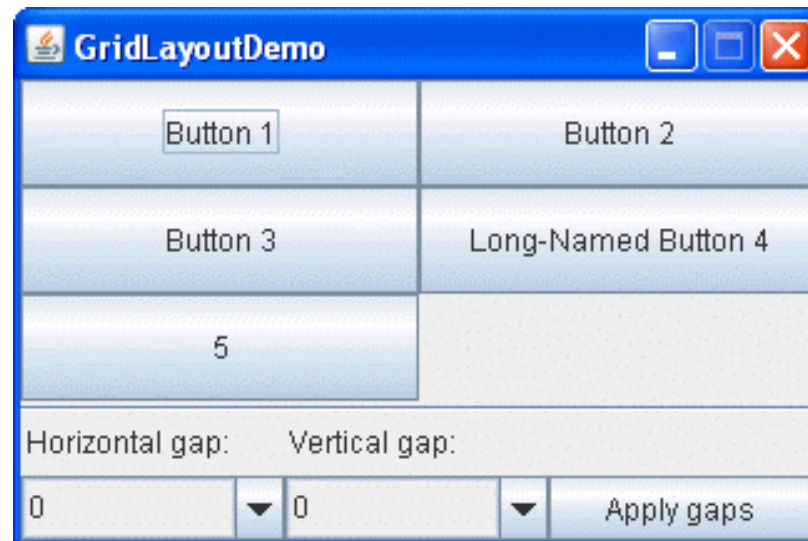


...

```
JFrame f = new JFrame("BorderLayoutDemo");  
Container cont = f.getContentPane();  
cont.setLayout(new BorderLayout());  
cont.add(new JButton("Button 1 (PAGE_START)"), BorderLayout.PAGE_START);  
cont.add(new JButton("Button 2 (PAGE CENTER)"), BorderLayout.CENTER);
```

# GridLayout

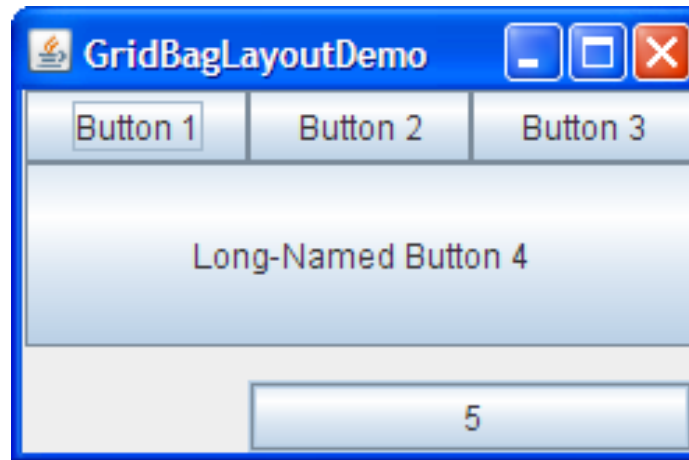
- É o gerenciador de layout que divide a área de um container em uma grade de células com o mesmo tamanho. Cada componente é alocado para uma célula.



```
JFrame f = new JFrame("GridLayoutDemo");  
Container cont = f.getContentPane();  
cont.setLayout(new GridLayout(0,2));  
cont.add(new JButton("Button 1"));  
cont.add(new JButton("Button 2"));
```

# GridBagLayout

- É o gerenciador de layout que divide a área de um container em uma grade de células com tamanhos distintos. Cada componente pode ocupar uma ou várias células.

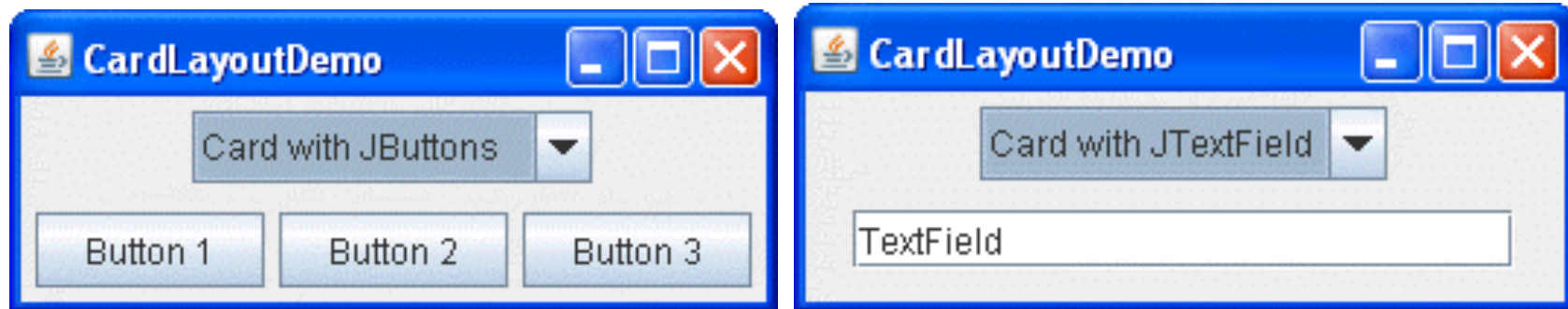


```
JFrame f = new JFrame("GridBagLayoutDemo");  
Container cont = f.getContentPane();  
cont.setLayout(new GridBagLayout());  
GridBagConstraints c = new GridBagConstraints();  
c.fill = GridBagConstraints.HORIZONTAL;  
c.gridx = 0; c.gridy = 0;  
cont.add(new JButton("Button 1"), c);
```



# CardLayout

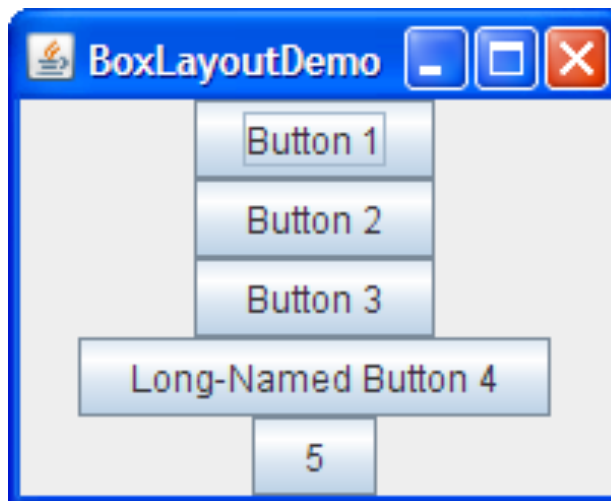
- É o gerenciador de layout que possibilita gerenciar dois ou mais componentes (geralmente objetos JPanel) que compartilham a mesma área de apresentação.



```
JFrame f = new JFrame("CardLayoutDemo");
Container cont = f.getContentPane();
JPanel card1 = new JPanel();
card1.add(new JButton("Button 1")); ...
JPanel card2 = new JPanel();
card2.add(new JTextField("TextField", 20));
cards = new JPanel(new CardLayout());
cards.add(card1, BUTTONPANEL);
cards.add(card2, TEXTPANEL);
```

# BoxLayout

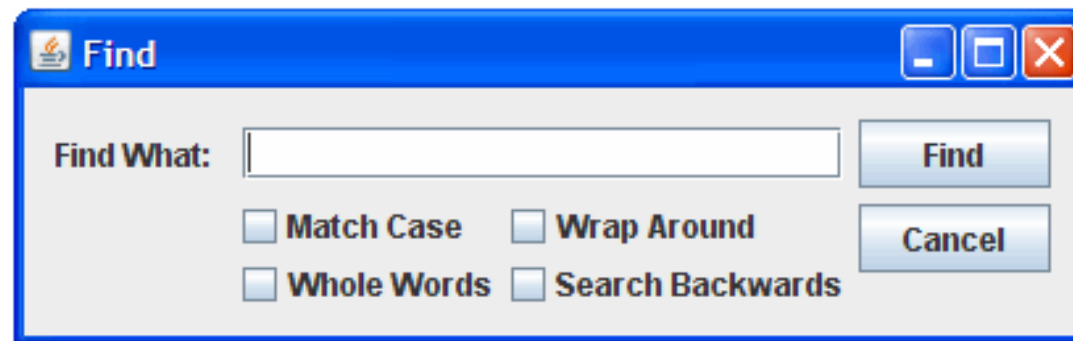
- É o gerenciador de layout que organiza os componentes em uma única linha ou coluna. Respeita tamanhos máximos solicitados dos componentes e também permite alinhar componentes.



```
JFrame f = new JFrame("BoxLayoutDemo");  
Container cont = f.getContentPane();  
cont.setLayout(new BoxLayout(), BoxLayout.Y_AXIS);  
cont.add(new JButton("Button 1"));  
cont.add(new JButton("Button 2"));
```

# GroupLayout

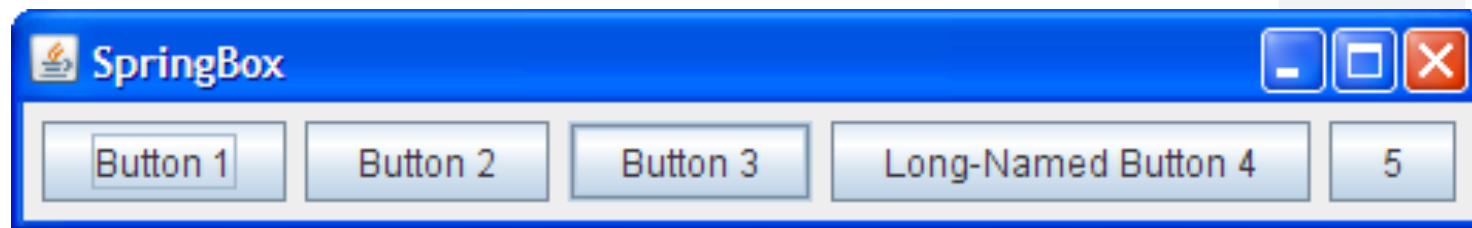
- É o gerenciador de layout que trabalha com os esquemas horizontais e verticais separadamente. O layout é definido para cada dimensão de forma independente.



```
GroupLayout layout = new GroupLayout(panel);  
panel.setLayout(layout);  
layout.setAutoCreateGaps(true);  
layout.setAutoCreateContainerGaps(true);
```

# SpringLayout

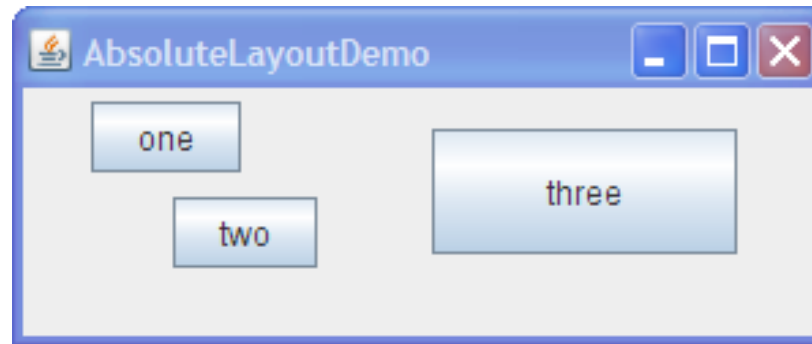
- É o gerenciador de layout que permite que seja especificado as relações precisas entre as bordas dos componentes sob o seu controle.



```
JFrame f = new JFrame("SpringBox");  
Container cont = f.getContentPane();  
cont.setLayout(new SpringLayout());  
cont.add(new JButton("Button 1"));  
cont.add(new JButton("Button 2"));  
cont.add(new JButton("Button 3"));  
cont.add(new JButton("Long-Named Button 4"));  
cont.add(new JButton("5"));
```

# Sem Gerenciador de Layout

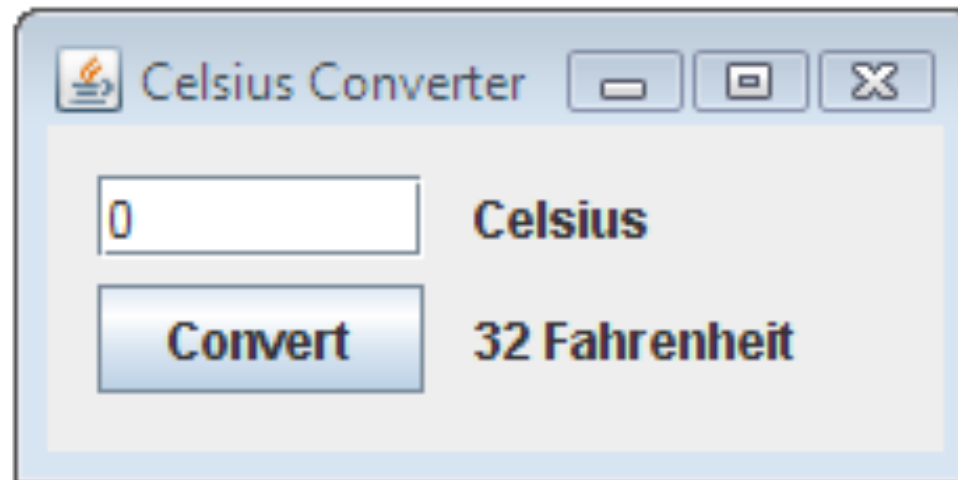
- Embora existam vários gerenciadores de layout, é possível criar aplicações gráficas sem utilizar nenhum deles. Basta definir o método **setLayout(null)**.



```
JFrame f = new JFrame("AbsoluteLayoutDemo");  
Container cont = f.getContentPane();  
cont.setLayout(null);  
cont.add(new JButton("one"));  
cont.add(new JButton("two"));  
cont.add(new JButton("three"));
```

# Exercício

- 1) Alterar o gerenciador de layout utilizado na classe **ConverterApp**.  
Observar a mudança de comportamento dos seus componentes gráficos.





# Eventos

# Eventos

- Um Evento é a ocorrência de alguma ação em um dado instante de tempo (que pode ser instantâneo).
- Exemplos de Eventos: movimento do mouse, clique de mouse, entrada de dados, seleção de uma opção em um menu, etc.
- No Java, cada componente Swing (*event source*) pode estar associado a diversos tipos de Eventos (*event listeners*).





# Event Source

- Os principais tipos de eventos (*event source*) estão definidos nos pacotes **java.awt.event.\*** e **javax.swing.event.\***
- Abaixo estão listados os principais *event source* definidos pelo Java:
  - **ActionEvent** (fonte: componentes de ação);
  - **MouseEvent** (fonte: componentes afetados pelo mouse);
  - **ItemEvent** (fonte: checkboxes e similares);
  - **AdjustmentEvent** (fonte: scrollbars);
  - **TextEvent** (fonte: componentes de texto);
  - **WindowEvent** (fonte: janelas);
  - **FocusEvent** (fonte: componentes em geral);
  - **KeyEvent** (fonte: componentes afetados pelo teclado).

# Event Listeners

- Abaixo estão listados os principais *event listeners* suportados pelos componentes Swing:
  - **ActionListener** (`java.awt.event.ActionListener`)
  - **CaretListener** (`javax.swing.event.CaretListener`)
  - **ChangeListener** (`javax.swing.event.ChangeListener`)
  - **DocumentListener** (`javax.swing.event.DocumentListener`)
  - **UndoableEditListener** (`javax.swing.event.UndoableEditListener`)
  - **ItemListener** (`java.awt.event.ItemListener`)
  - **ListSelectionListener** (`javax.swing.event.ListSelectionListener`)
  - **WindowListener** (`java.awt.event.WindowListener`)
- A seguir, os mais populares *event listeners* serão apresentados.

# ActionListener

- É o tipo de Evento mais comum de ser implementado quando o usuário clica em um botão, escolhe um item de um menu, pressiona **<ENTER>** após a entrada de dados em um campo de texto.
- Para utilizar um **ActionListener** torna-se necessário:
  - 1. A classe que cria os componentes gráficos implementar esta interface.

```
public class MyClass implements ActionListener {
```

- 2. Adicionar este tipo de evento para um componente gráfico específico.

```
componente.addActionListener(instanciaMyClass);
```

- 3. Implementar o único método desta interface.

```
public void actionPerformed(ActionEvent e) {  
    // escrever código para tratar as ações }  
}
```

# ActionListener

- EXEMPLO

```
import java.awt.*;import java.awt.event.*;
import javax.swing.*;
public class JanelaFL extends JFrame implements
    ActionListener {
    public JanelaFL(int x, int y, String titulo)
    {
        super();
        this.setTitle(titulo);
        this.setSize(x,y);
        this.getContentPane().setLayout(new FlowLayout());
    }
    ...
}
```

# ActionListener

- EXEMPLO

```
public static void main(String args[]){
    JanelaFL janela = new JanelaFL(300,300,"Janela");
    JButton b1, b2;
    b1 = new JButton("Botão 1");
    b2 = new JButton("Botão 2");
    b1.addActionListener(janela);
    b2.addActionListener(janela);
    janela.getContentPane().add(b1);
    janela.getContentPane().add(b2);
    janela.setVisible(true);
}

public void actionPerformed(ActionEvent e){
    String comando = e.getActionCommand();
    System.out.println("-->" + comando);
}
}
```

# ItemListener

- É o tipo de Evento mais comum de ser implementado quando o usuário manipula check boxes, combo boxes, check menu items, etc.
- Para utilizar um **ItemListener** torna-se necessário:
  - 1. A classe que cria os componentes gráficos implementar esta interface.

```
public class MyClass implements ItemListener {
```

- 2. Adicionar este tipo de evento para um componente gráfico específico.

```
componente.addItemListener(instanciaMyClass);
```

- 3. Implementar o único método desta interface.

```
public void itemStateChanged(ItemEvent e) {  
    // escrever código para tratar as ações  
}
```

# ItemListener

- EXEMPLO

```
import java.awt.*;import java.awt.event.*;
import javax.swing.*;
public class Check extends JFrame implements
    ItemListener{
    JCheckBox cb1, cb2, cb3;
    public Check(int x, int y, String titulo)
    {
        super();
        this.getContentPane().setLayout(new FlowLayout());
        this.setTitle(titulo);
        this.setSize(x,y);
    }
    ...
}
```

# ItemListener

- EXEMPLO

```
public static void main(String args[]){
    Check janela = new Check(200,200,"janela");
    janela.cb1 = new JCheckBox("Checkbox 1");
    janela.cb2 = new JCheckBox("Checkbox 2");
    janela.cb3 = new JCheckBox("Checkbox 3");
    janela.getContentPane().add(janela.cb1);
    janela.getContentPane().add(janela.cb2);
    janela.getContentPane().add(janela.cb3);
    janela.cb1.addItemListener(janela);
    janela.cb2.addItemListener(janela);
    janela.cb3.addItemListener(janela);
    janela.setVisible(true);
}
...
```



# ItemListener

- EXEMPLO

```
public void itemStateChanged(ItemEvent event)
{
    if (event.getSource() == cb1)
        System.out.println("cb1");
    else if (event.getSource() == cb2)
        System.out.println("cb2");
    else
        System.out.println("cb3");
}
```

# WindowListener

- É o tipo de Evento mais comum de ser implementado quando o usuário manipula janelas, maximizando-as, minimizando-as, fechando-as, etc.
- Para utilizar um **WindowListener** torna-se necessário:
  - 1. A classe que cria os componentes gráficos implementar esta interface.

```
public class MyClass implements WindowListener {
```

- 2. Adicionar este tipo de evento para um componente gráfico específico.

```
janela.addWindowItemListener(instanciaMyClass);
```

- 3. Implementar os sete métodos desta interface.

```
public void windowOpened(WindowEvent e) {  
    // escrever código para tratar as ações }  
}
```

# WindowListener

- 3. Implementar os sete métodos desta interface.

```
public void windowOpened(WindowEvent e) {  
    // escrever código para tratar as ações }  
public void windowClosing(WindowEvent e) {  
    // escrever código para tratar as ações }  
public void windowClosed(WindowEvent e) {  
    // escrever código para tratar as ações }  
public void windowIconified(WindowEvent e) {  
    // escrever código para tratar as ações }  
public void windowDeiconified(WindowEvent e) {  
    // escrever código para tratar as ações }  
public void windowActivated(WindowEvent e) {  
    // escrever código para tratar as ações }  
public void windowDeactivated(WindowEvent e) {  
    // escrever código para tratar as ações }
```

# Classes Adaptadoras

- Em algumas situações pode ser muito cansativo implementar todos os métodos definidos em um *event listener*, tais como os oito métodos definidos na interface `WindowListener`.
- Para evitar este trabalho desnecessário, alguns *event listeners* possuem uma classe (abstrata) adaptadora correspondente.
- A classe adaptadora possui uma **implementação vazia ({})** para todos os métodos definidos em um determinado *event listener*. Dessa forma, pode-se criar uma subclasse da classe adaptadora e anular os métodos desejados, escrevendo a sua própria implementação.
- A seguir, são apresentados três exemplos de classes adaptadoras.

# Classes Adaptadoras

```
public abstract class WindowAdapter extends Object  
implements WindowListener,  
             WindowStateListener, WindowFocusListener
```

```
public abstract class KeyAdapter extends Object  
implements KeyListener
```

```
public abstract class MouseAdapter extends Object  
implements MouseListener,  
             MouseWheelListener, MouseMotionListener
```

# Exercícios

- 1) Implementar o tratamento de eventos nas classes **JTFApp** e **ButtonApp**.
- 2) Implementar o tratamento de eventos nas classes **JTFAppv2** e **ButtonAppv2** fazendo uso de Lambdas.



# Look and Feel

# Look And Feel (L&F)

- **"Look"** se refere à aparência dos componentes gráficos e **"Feel"** ao comportamento destes na ocorrência de eventos.
- Em geral, as plataformas tecnológicas possuem o seu próprio L&F, conforme a listagem da tabela abaixo:

Platform	Look and Feel
Solaris, Linux with GTK+ 2.2 or later	GTK+
Other Solaris, Linux	Motif
IBM UNIX	IBM*
HP UX	HP*
Classic Windows	Windows
Windows XP	Windows XP
Windows Vista	Windows Vista
Macintosh	Macintosh*

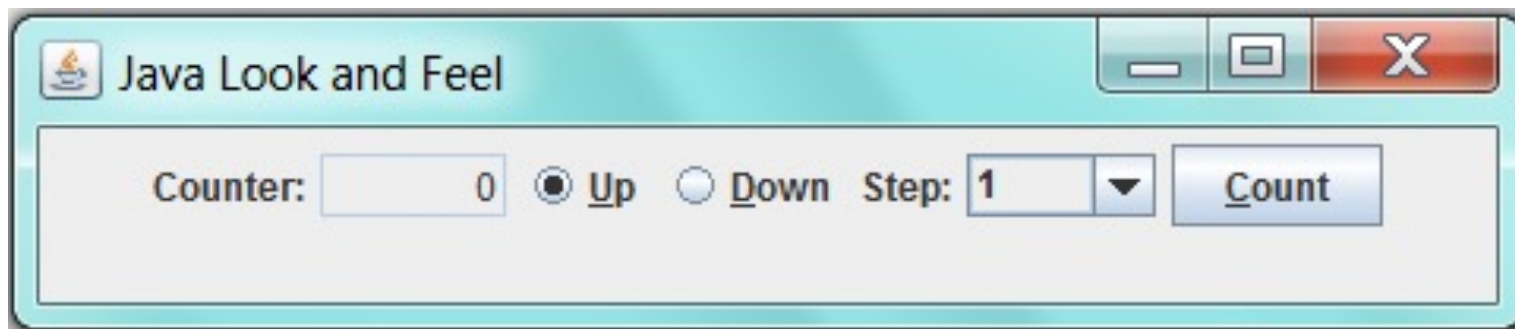
\* Supplied by the system vendor.



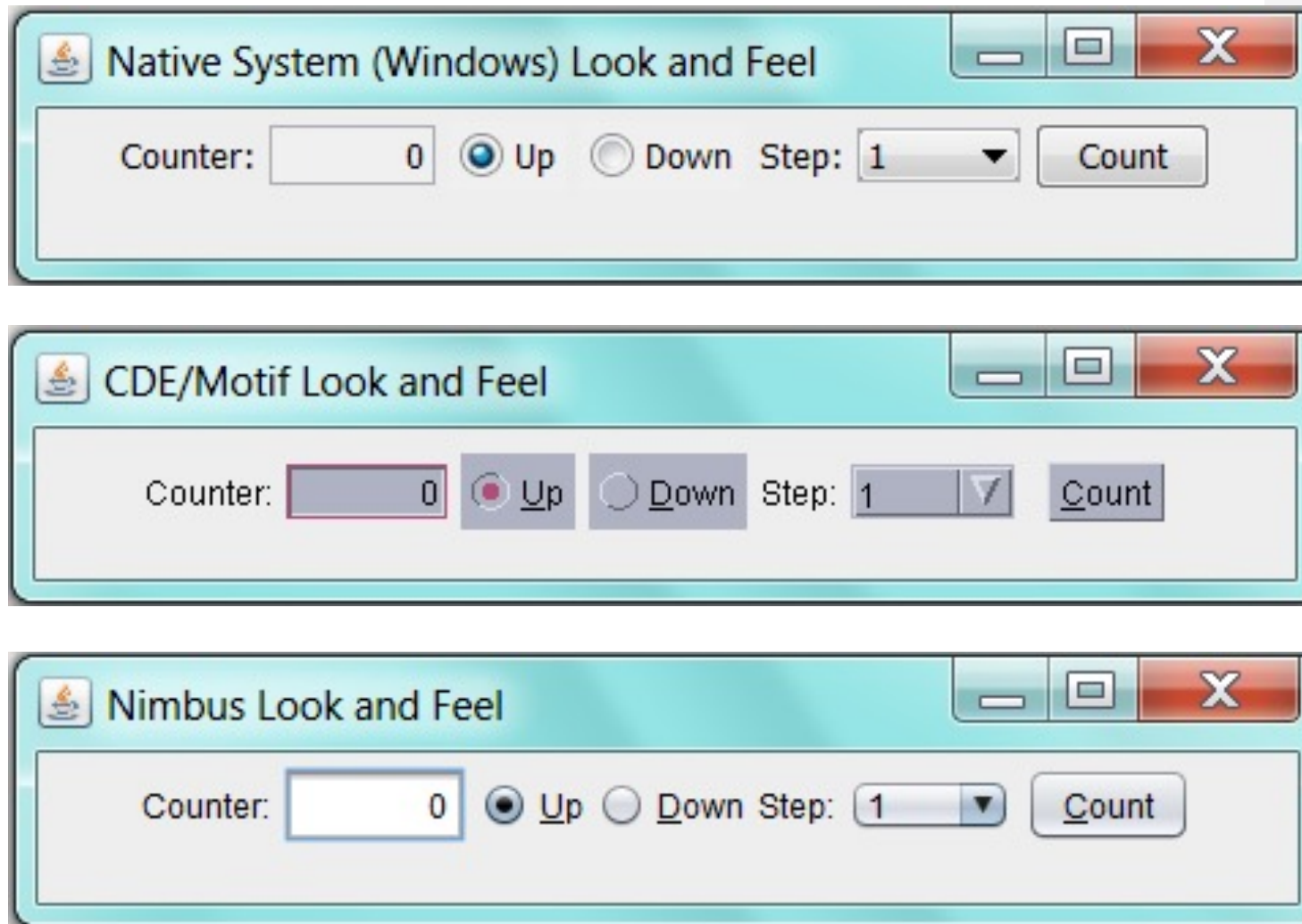
# Look And Feel (L&F)

- No Java SDK o L&F GTK+, Motif e Windows são definidos nas classes abaixo:

```
com.sun.java.swing.plaf.gtk.GTKLookAndFeel  
com.sun.java.swing.plaf.motif.MotifLookAndFeel  
com.sun.java.swing.plaf.windows.WindowsLookAndFeel
```



# Look And Feel (L&F)



# Look And Feel (L&F)

- A classe UIManager é que especifica qual o L&F a ser utilizado:

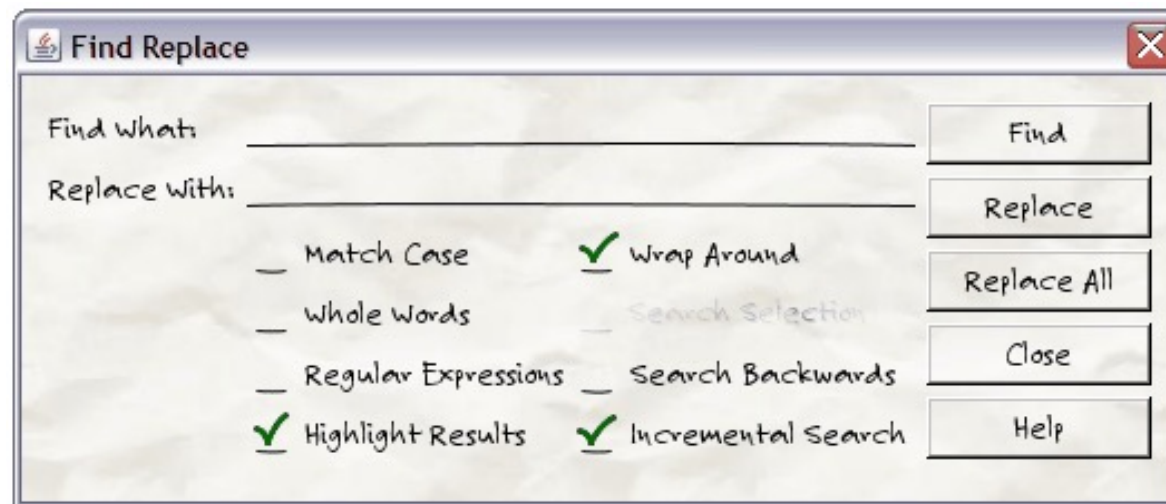
```
UIManager.setLookAndFeel("Windows");
```

- Esta classe pode assumir quatro estilos de L&F:
  - **Java L&F [0] (padrão Swing)**
  - **Motif [1] (CDE/Motif)**
  - **Windows [2]**
  - **Linux/GTK [3]**

```
UIManager.setLookAndFeel(looks[0].getClassName());
```

# Look And Feel (L&F)

- Além dos estilos de L&F disponíveis no Java SDK, é possível fazer uso de estilos de apresentação de interfaces gráficas de outros fabricantes (gratuitos ou pagos).



```
public static void main(String[] args) {  
    try {  
        UIManager.setLookAndFeel(new net.sourceforge.napkinlaf.NapkinLookAndFeel());  
    } catch (Exception unused) {  
        ; // Ignore exception because we can't do anything. Will use default.  
    }  
    ...  
}
```

Exemplo NapkinL&F ([napkinlaf.sourceforge.net](http://napkinlaf.sourceforge.net))

# Exercício

- 1) Alterar o L&F da classe **ButtonApp**.
  - Dica: utilizar o trecho de código abaixo.

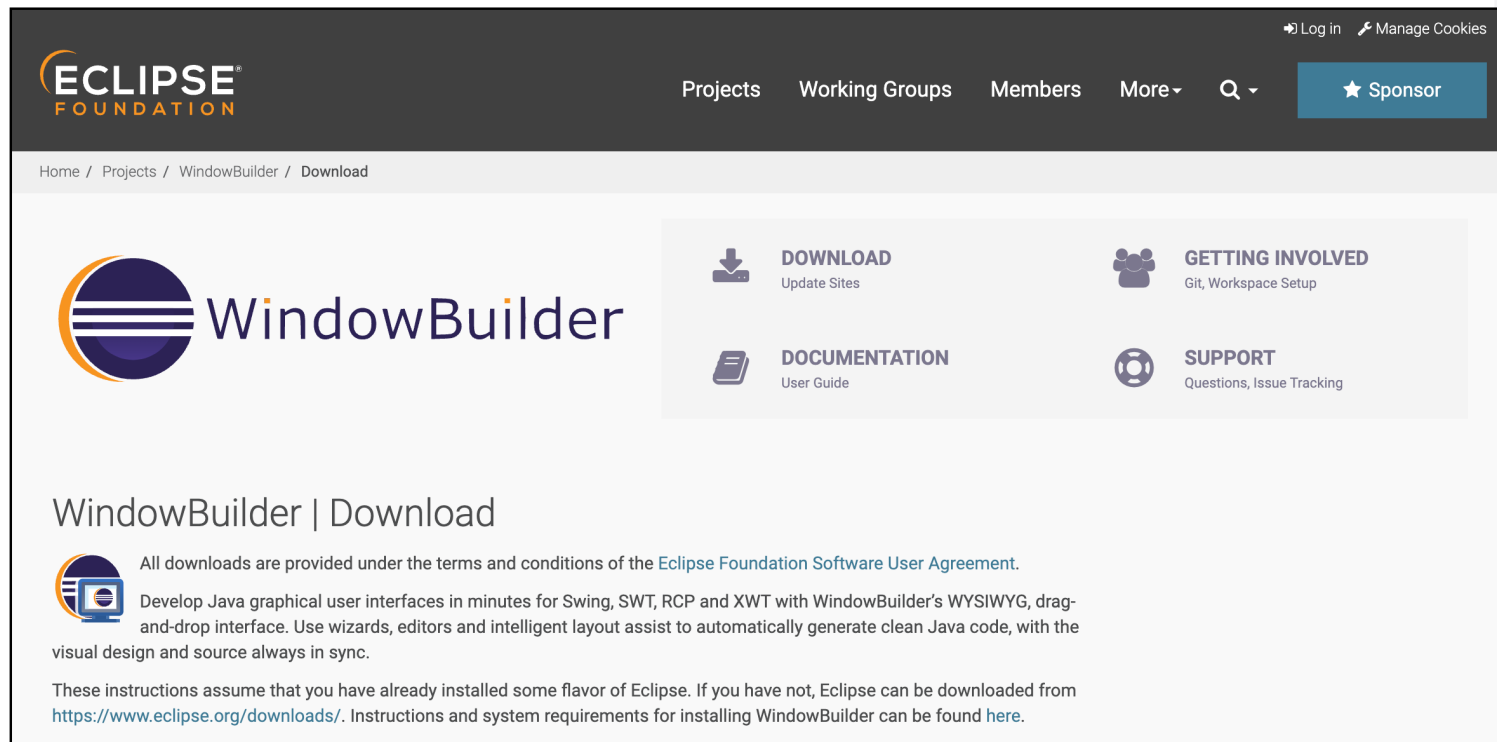
```
//novo atributo
public UIManager.LookAndFeelInfo looks[];
//código no construtor
looks = UIManager.getInstalledLookAndFeels();
try
{
    UIManager.setLookAndFeel(looks[0].getClassName());
}
catch (Exception e){...}
```



# Editor Visual

# O que é o WindowBuilder?

- O WindowBuilder é uma poderosa ferramenta de design de interfaces gráficas de usuário (GUI) para o ambiente de desenvolvimento Eclipse. Ele facilita a criação de interfaces visuais para aplicações Java, permitindo que os desenvolvedores criem, modifiquem e visualizem GUIs de forma intuitiva e eficiente.





The screenshot shows the Eclipse Foundation website's WindowBuilder download page. The header includes the Eclipse Foundation logo, navigation links (Projects, Working Groups, Members, More), a search icon, and a 'Sponsor' button. The breadcrumb trail reads 'Home / Projects / WindowBuilder / Download'. The main content area features the WindowBuilder logo and four action buttons: 'DOWNLOAD' (Update Sites), 'GETTING INVOLVED' (Git, Workspace Setup), 'DOCUMENTATION' (User Guide), and 'SUPPORT' (Questions, Issue Tracking). Below this, the section 'WindowBuilder | Download' contains a disclaimer about the Eclipse Foundation Software User Agreement, a brief description of WindowBuilder's capabilities (developing Java GUIs for Swing, SWT, RCP, and XWT), and a link to the Eclipse download page for users who haven't installed Eclipse yet.


ECLIPSE FOUNDATION


Projects Working Groups Members More Q Sponsor


Home / Projects / WindowBuilder / Download

 WindowBuilder


 **DOWNLOAD**  
Update Sites

 **GETTING INVOLVED**  
Git, Workspace Setup

 **DOCUMENTATION**  
User Guide

 **SUPPORT**  
Questions, Issue Tracking

## WindowBuilder | Download

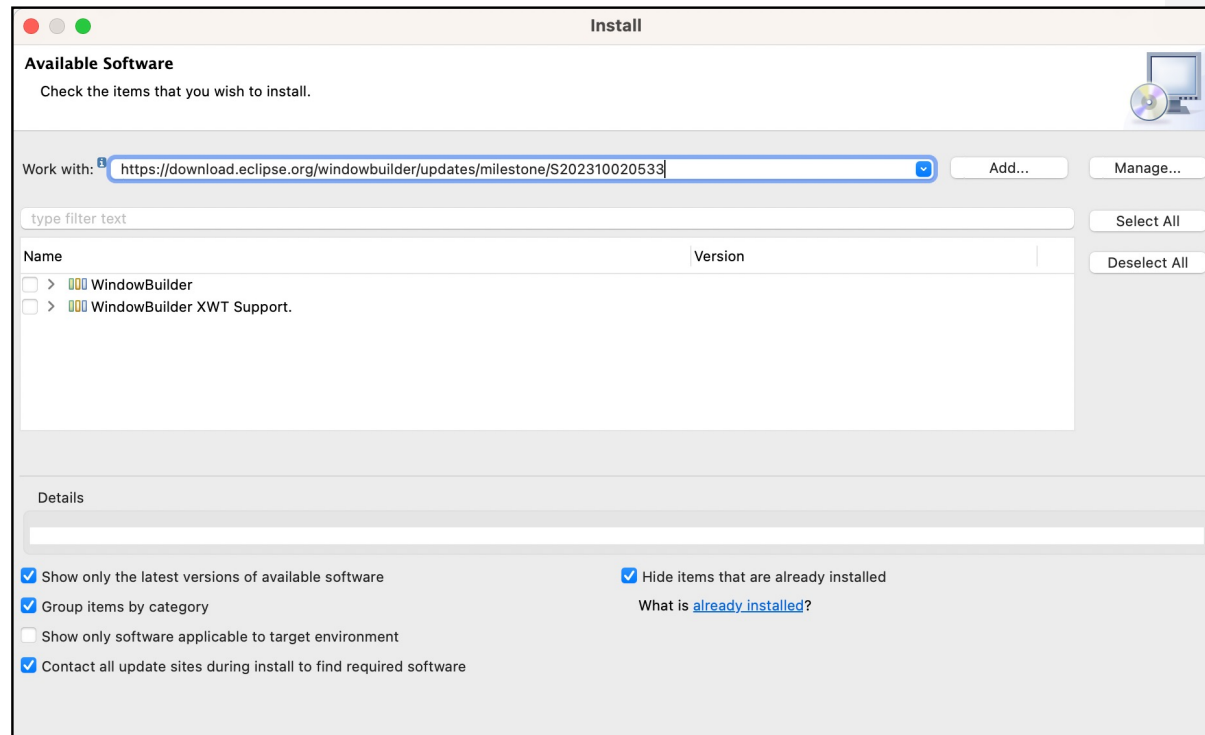
 All downloads are provided under the terms and conditions of the [Eclipse Foundation Software User Agreement](#).

Develop Java graphical user interfaces in minutes for Swing, SWT, RCP and XWT with WindowBuilder's WYSIWYG, drag-and-drop interface. Use wizards, editors and intelligent layout assist to automatically generate clean Java code, with the visual design and source always in sync.

These instructions assume that you have already installed some flavor of Eclipse. If you have not, Eclipse can be downloaded from <https://www.eclipse.org/downloads/>. Instructions and system requirements for installing WindowBuilder can be found [here](#).

# Instalação do WindowBuilder

- No Eclipse selecione o Menu “**Help**” e depois a opção “**Install New Software**”.

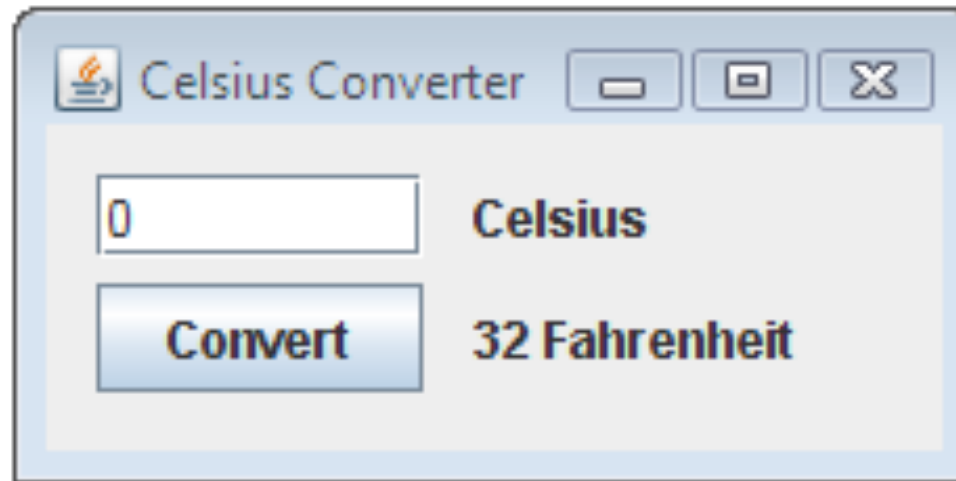


- Abaixo o endereço:
- <https://download.eclipse.org/windowbuilder/updates/milestone/S202310020533>



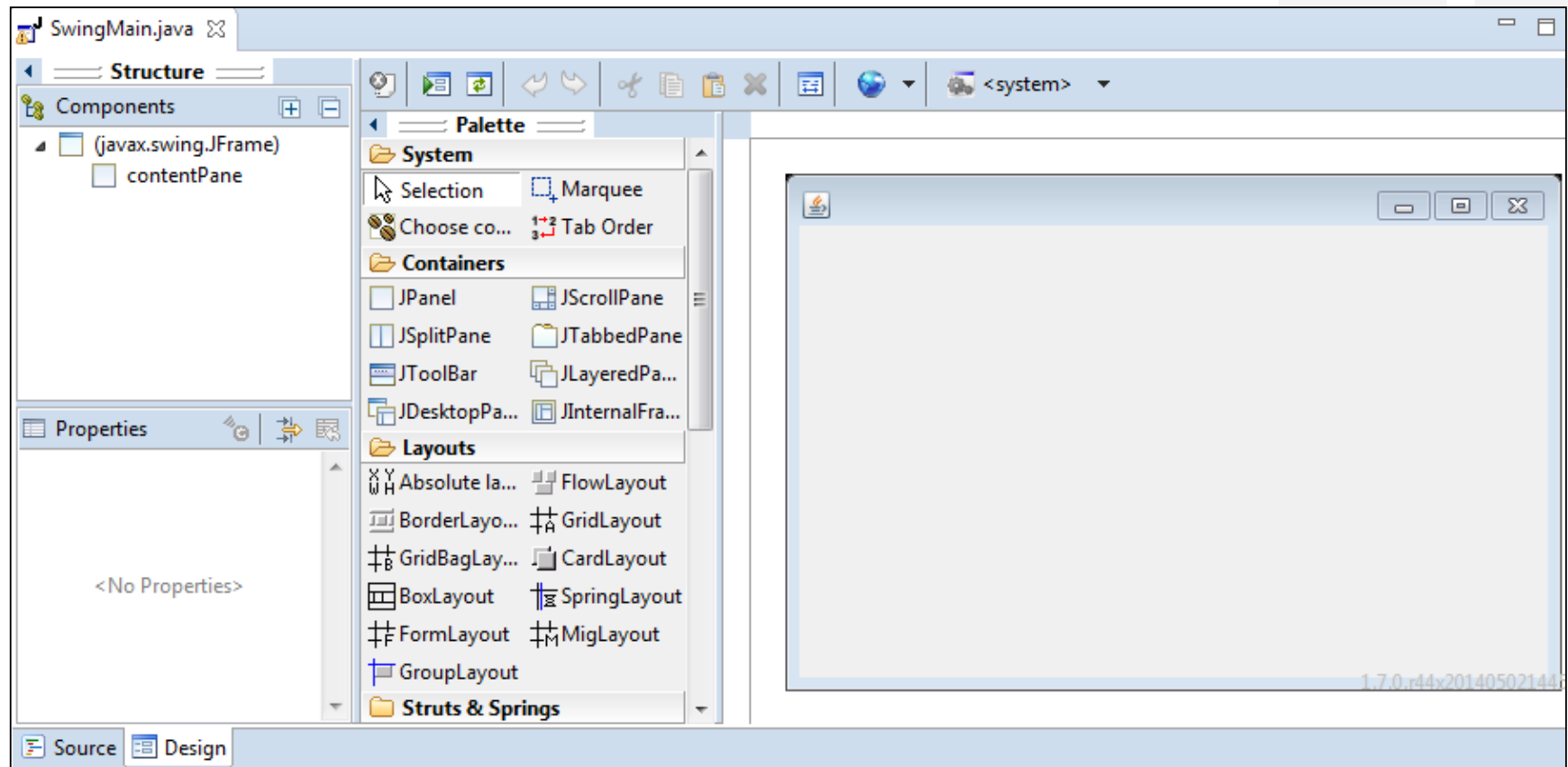
# Exemplo de Uso

- Implementar a classe **ConverterAppv2** que execute a conversão da temperatura em °C para °F.

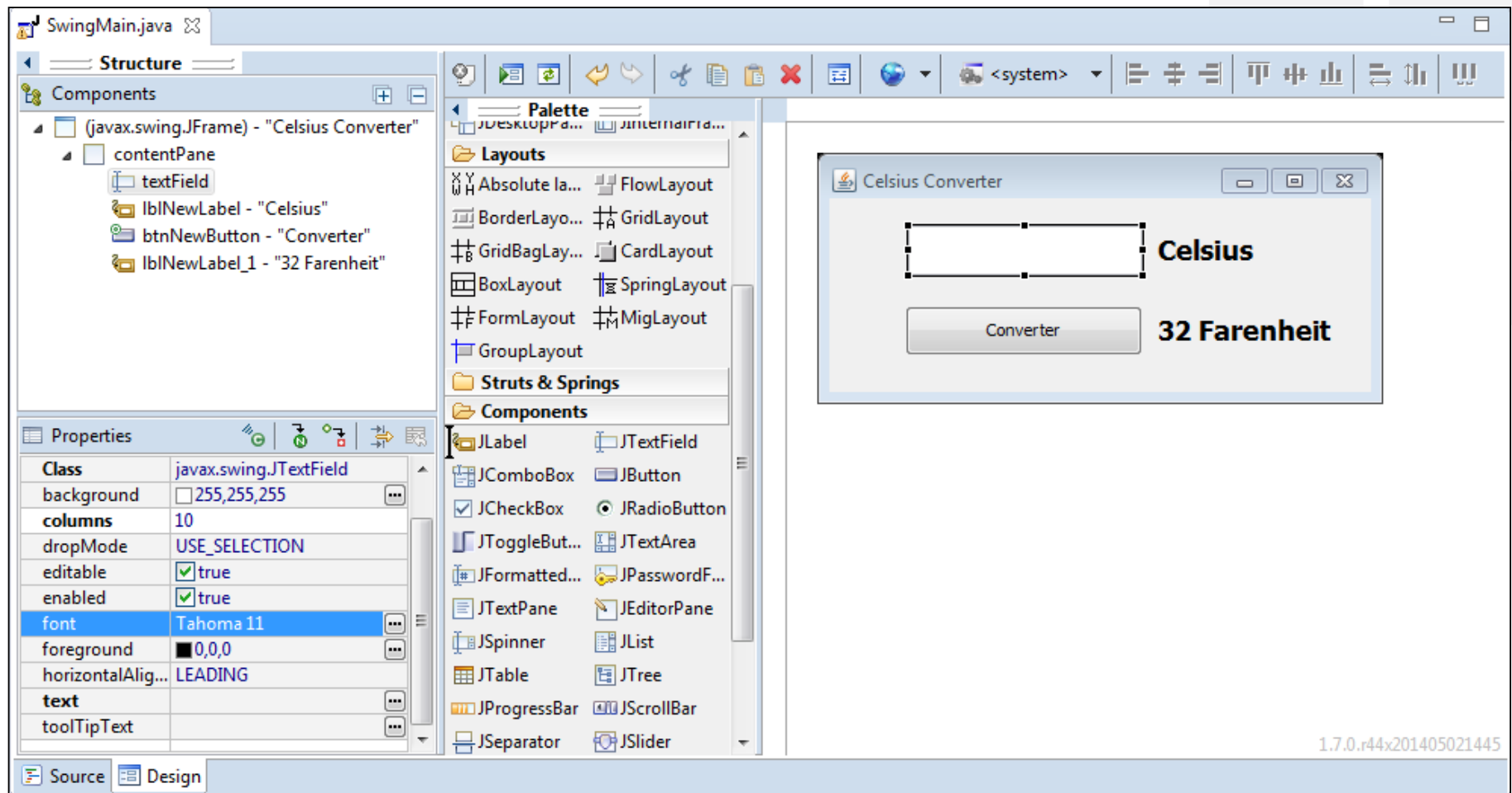


- Passos:**
  - 1° - Escolha o menu **"File"** ⇒ **"New"** ⇒ **"Other..."** ⇒ **"WindowBuilder"** ⇒ **"Swing Designer"** ⇒ **"JFrame"** ⇒ **"Next"**.
  - 2° - Na caixa de diálogo **"Create JFrame"** ⇒ Escreva **"ConverterAppv2"** no campo **"Name"** ⇒ **"Finish"**.
  - 3 – Selecione o painel **"Design"**.

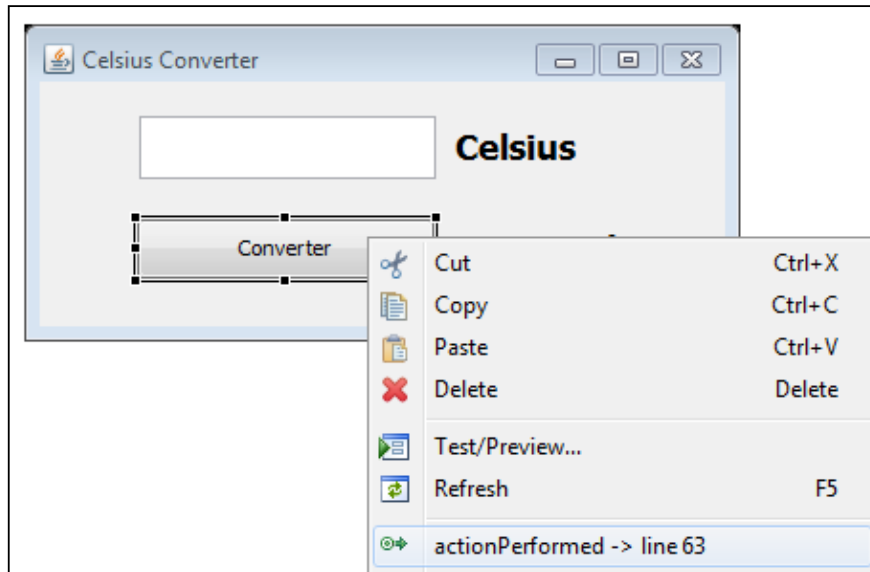
# Exemplo de Uso



# Exemplo de Uso



# Exemplo de Uso



```
btnNewButton.addActionListener(ev-> {  
    float cls = Float.parseFloat(celsius.getText());  
    float fht = (float) (9.0/5.0) * (cls) + 32;  
    celsius.setText(fht + "");  
});
```



# JavaFX

# Java FX

- Nas palavras da própria Oracle, JavaFX é “o próximo passo na evolução do Java como plataforma de desenvolvimento de aplicações cliente ricas”. A empresa também indica que a tendência é o JavaFX assumir o posto do Swing como principal biblioteca de interface do usuário da linguagem.
- A primeira versão oficial (1.0) só foi disponibilizada pela Sun em dezembro de 2008. Nas suas primeiras versões, antes da chegada da 2.0, os aplicativos JavaFX eram criados em uma linguagem de scripting específica, denominada JavaFX Script.

**//Exemplo de código JavaFX Script**

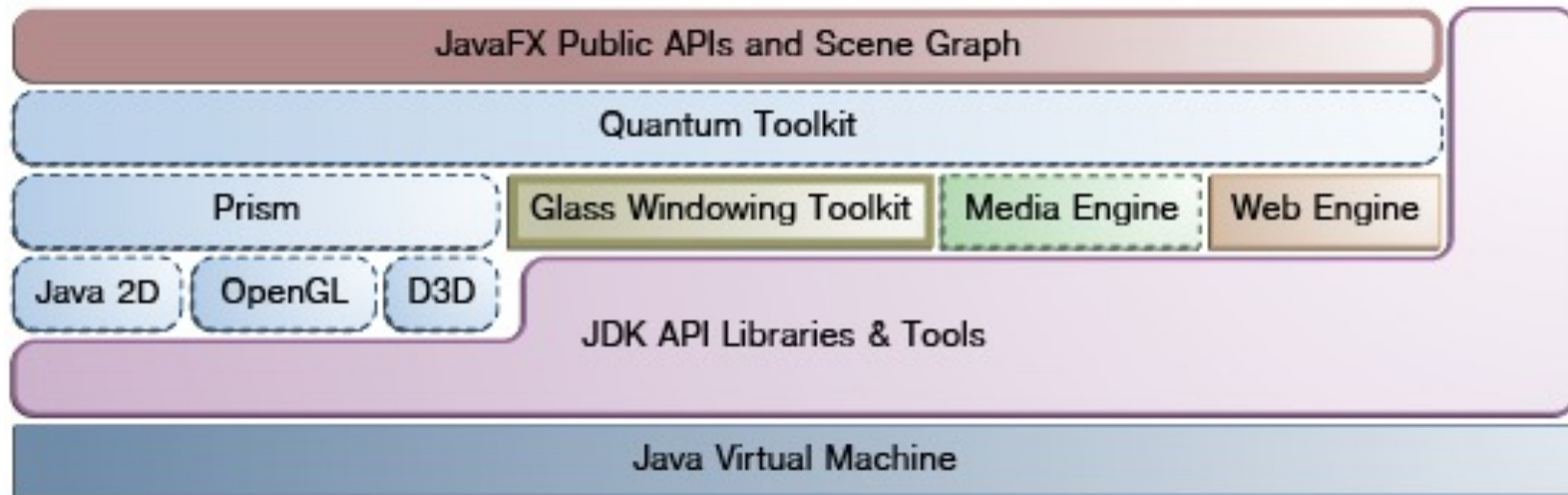
```
import java.lang.System;  
import java.util.Random;
```

```
var random = new Random(System.currentTimeMillis());  
println(random.nextInt(10));
```

# Java FX

- O lançamento da versão 2.0 marcou uma mudança significativa no JavaFX, já que a partir dela as aplicações puderam passar a ser escritas em código Java nativo.
- Agora, com o **JavaFX 8**, a solução é apresentada como a forma do programador Java “tradicional” reaproveitar seu conhecimento na linguagem para o desenvolvimento de aplicações ricas.
- O JavaFX não se resume somente a aplicações Java/desktop! Ele pode ser usado também em websites e aplicações móveis.
- Para utilizar o JavaFx no Eclipse é necessário a instalação do projeto e(fx)clipse disponível no site ([www.eclipse.org/efxclipse/index.html](http://www.eclipse.org/efxclipse/index.html)).

# Arquitetura Java FX



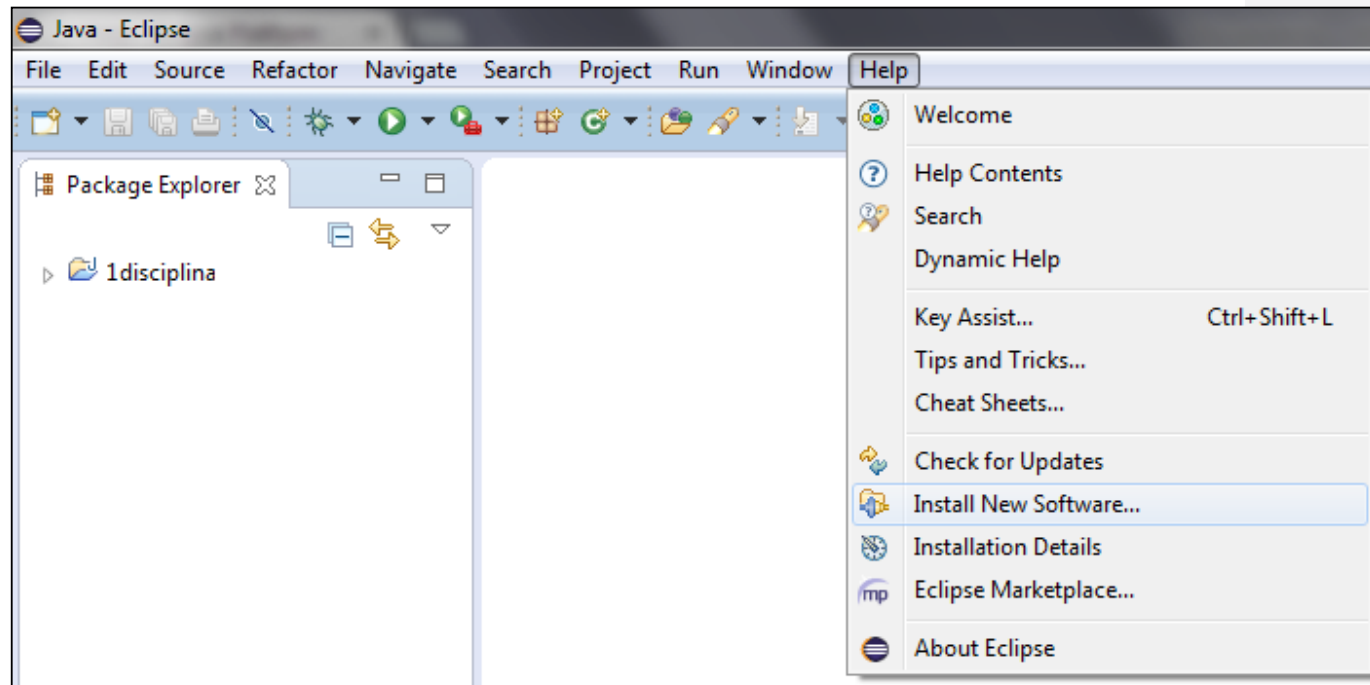
- O **projeto e(fx)clipse 1.0** oferece as seguintes ferramentas:
  - Integração com Eclipse 4.4;
  - Editor CSS para o JavaFX;
  - Editor FXML;
  - Editor FXGraph;
  - Suporte a mobilidade.





# Instalação do e(fx)clipse 1.0

- No Eclipse selecione o Menu “Help” e depois a opção “Install New Software”.

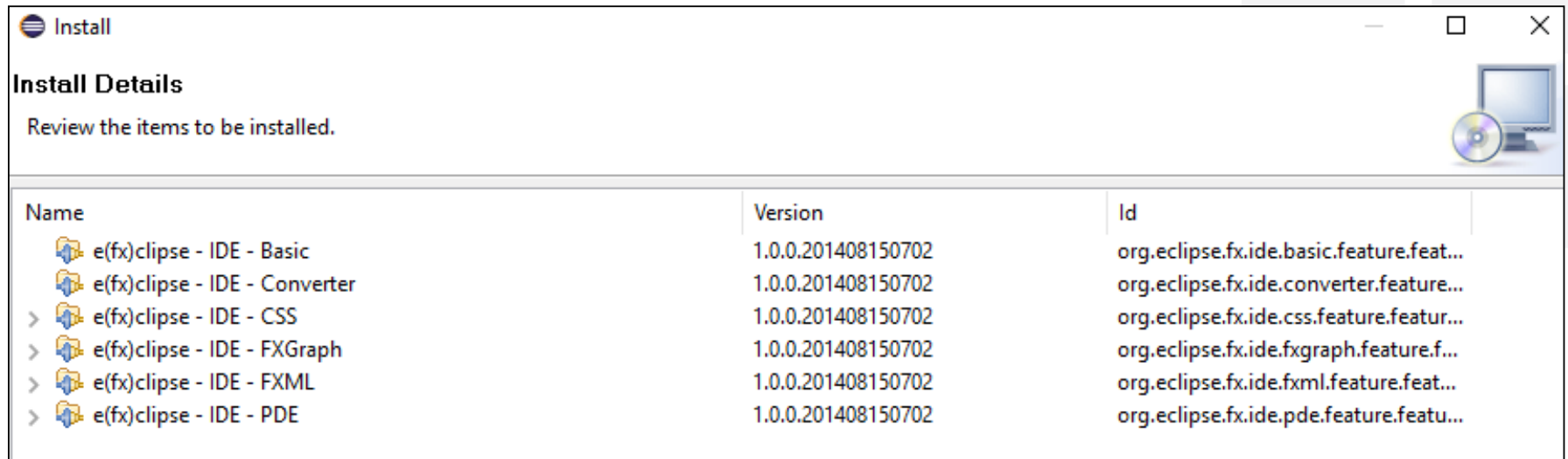


- Informe o endereço abaixo:

<http://download.eclipse.org/efxclipse/updates-released/1.0.0/site>

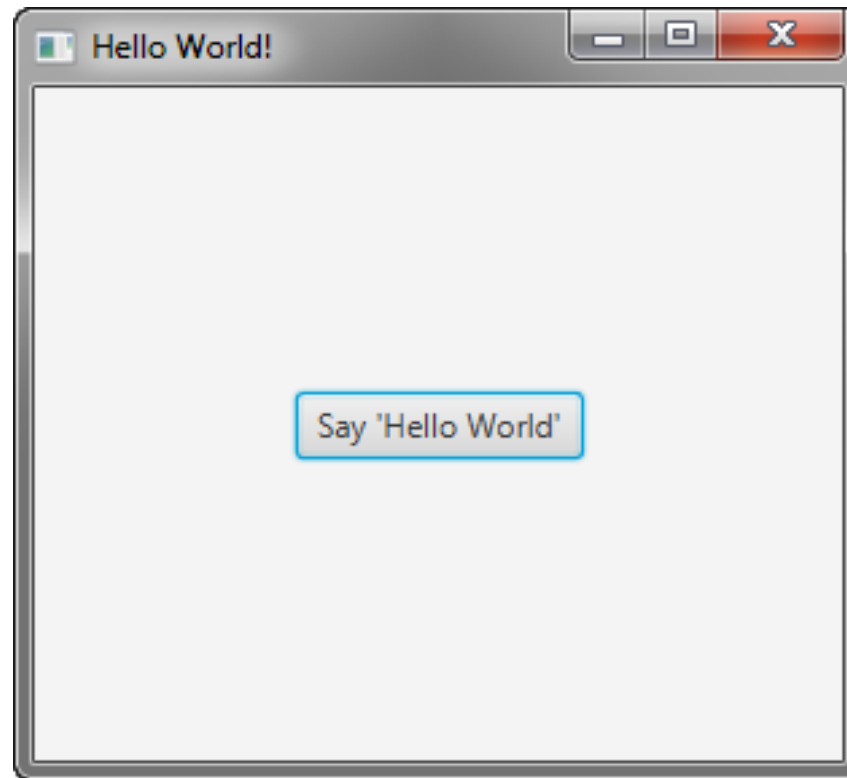
# Instalação do e(fx)clipse 1.0

- Selecionar as duas opção listadas.



# Exemplo de Uso

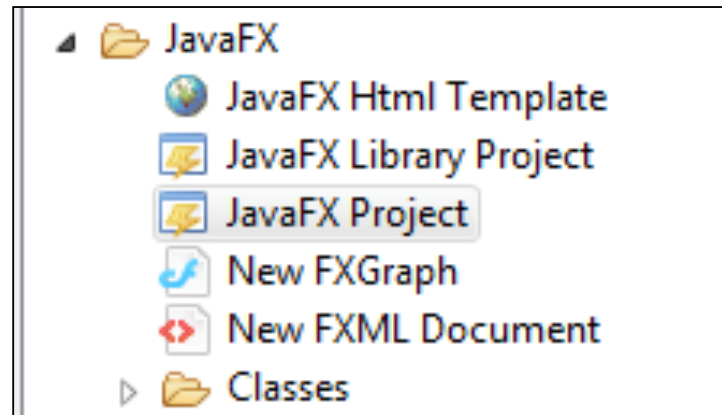
- Escrever a classe **Main.java** represente a interface gráfica abaixo e implemente o botão 'Say "Hello World"'.



# Exemplo de Uso

- **Passos:**

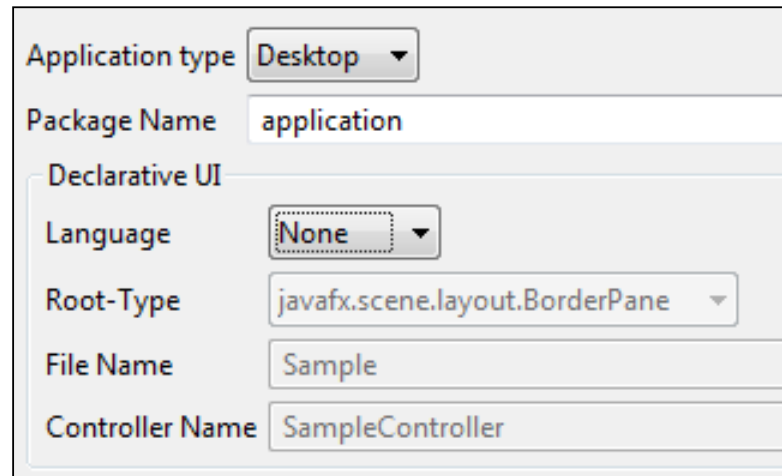
- 1° - Escolha o menu **"File" ⇒ "New Project" ⇒ "Other" ⇒ "JavaFxProject" ⇒ "Next"**.



- 2° - Na caixa de diálogo **"Application Type" ⇒ Manter "Desktop" ⇒ "Finish"**.

# Exemplo de Uso

- **Passos:**

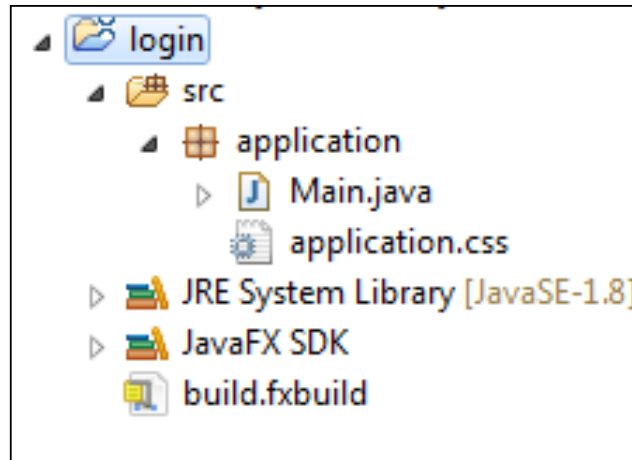


The screenshot shows a configuration window for a new application. The 'Application type' is set to 'Desktop'. The 'Package Name' is 'application'. Under the 'Declarative UI' section, the 'Language' is set to 'None'. The 'Root-Type' is set to 'javafx.scene.layout.BorderPane'. The 'File Name' is 'Sample' and the 'Controller Name' is 'SampleController'.

Application type	Desktop
Package Name	application
Declarative UI	
Language	None
Root-Type	javafx.scene.layout.BorderPane
File Name	Sample
Controller Name	SampleController

- 3° - Editar a classe **Main.java** que é criada automaticamente após a conclusão do passo 2.

# Exemplo de Uso



```
public class Main extends Application {  
    public void start(Stage primaryStage) {  
        try {  
            BorderPane root = new BorderPane();  
            Scene scene = new Scene(root, 400, 400);  
        } ...  
        public static void main(String[] args) {  
            launch(args);  
        }  
    }  
}
```

# Edição da Classe **Main.java**

```
public void start(Stage primaryStage) {
    try {
        StackPane root = new StackPane();
        primaryStage.setTitle("Aplicação JavaFx");
        Button btn = new Button();
        btn.setText("Say Hello World!");
        Label lb = new Label();
        btn.setOnAction(ev->lb.setText("\n\n\n Alo Pessoal"));
        root.getChildren().add(btn);
        root.getChildren().add(lb);
        Scene scene = new Scene(root,400,400);
        scene.getStylesheets().add(getClass().
            getResource("application.css").toExternalForm());
        primaryStage.setScene(scene);
        primaryStage.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}
```

# Edição do arquivo **application.css**

- **Passos:**

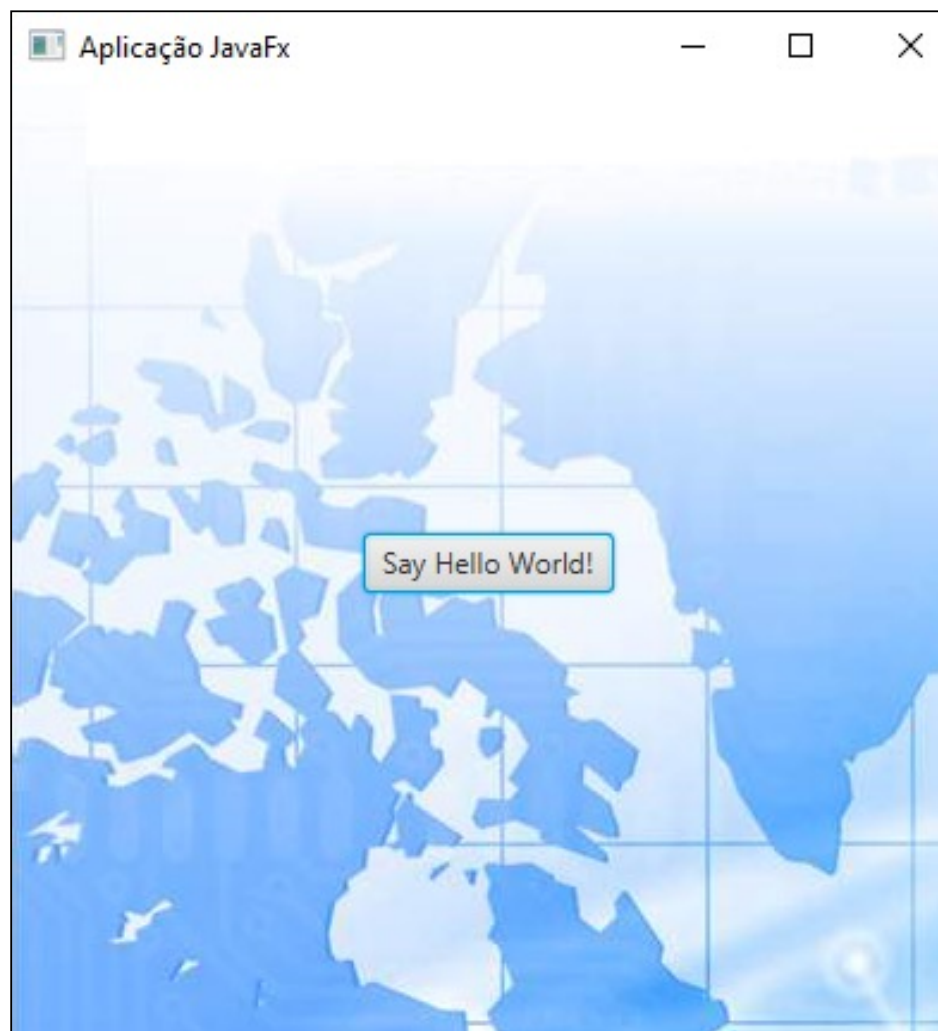
- 4° - Incluir um arquivo de imagem (**background.jpg**) no pacote application.
- 5° - Incluir o código abaixo no arquivo '**application.css**'.

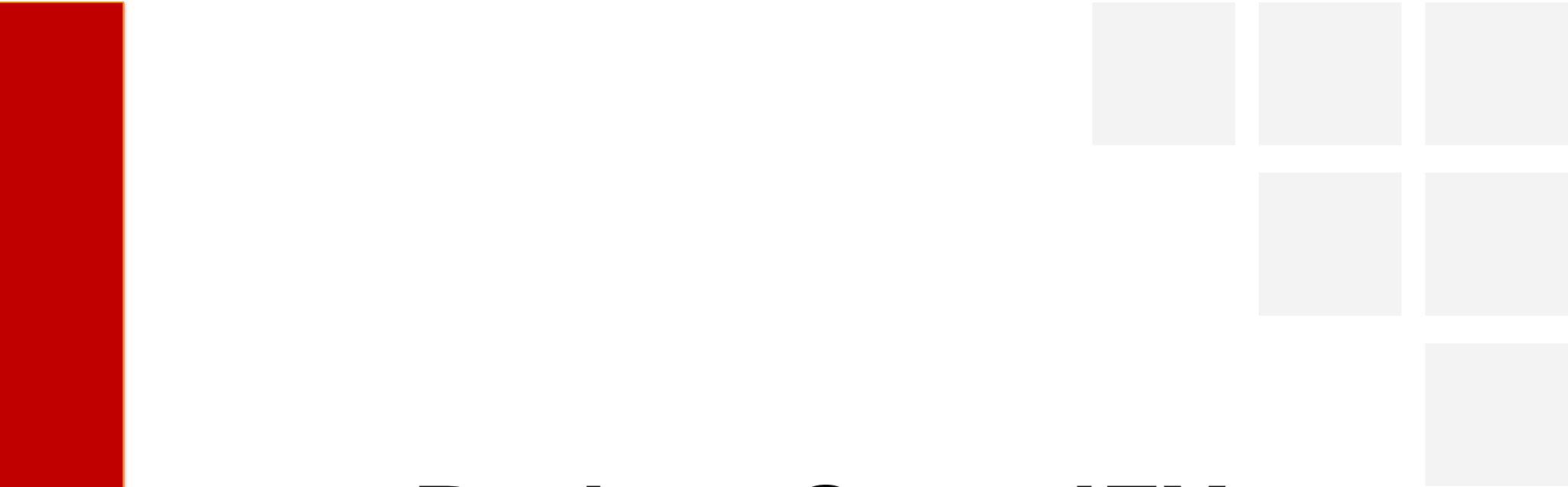
```
.root {  
    -fx-background-image: url("background.png");  
}
```

- 6 – Executar novamente a aplicação.



# Aplicação JavaFx

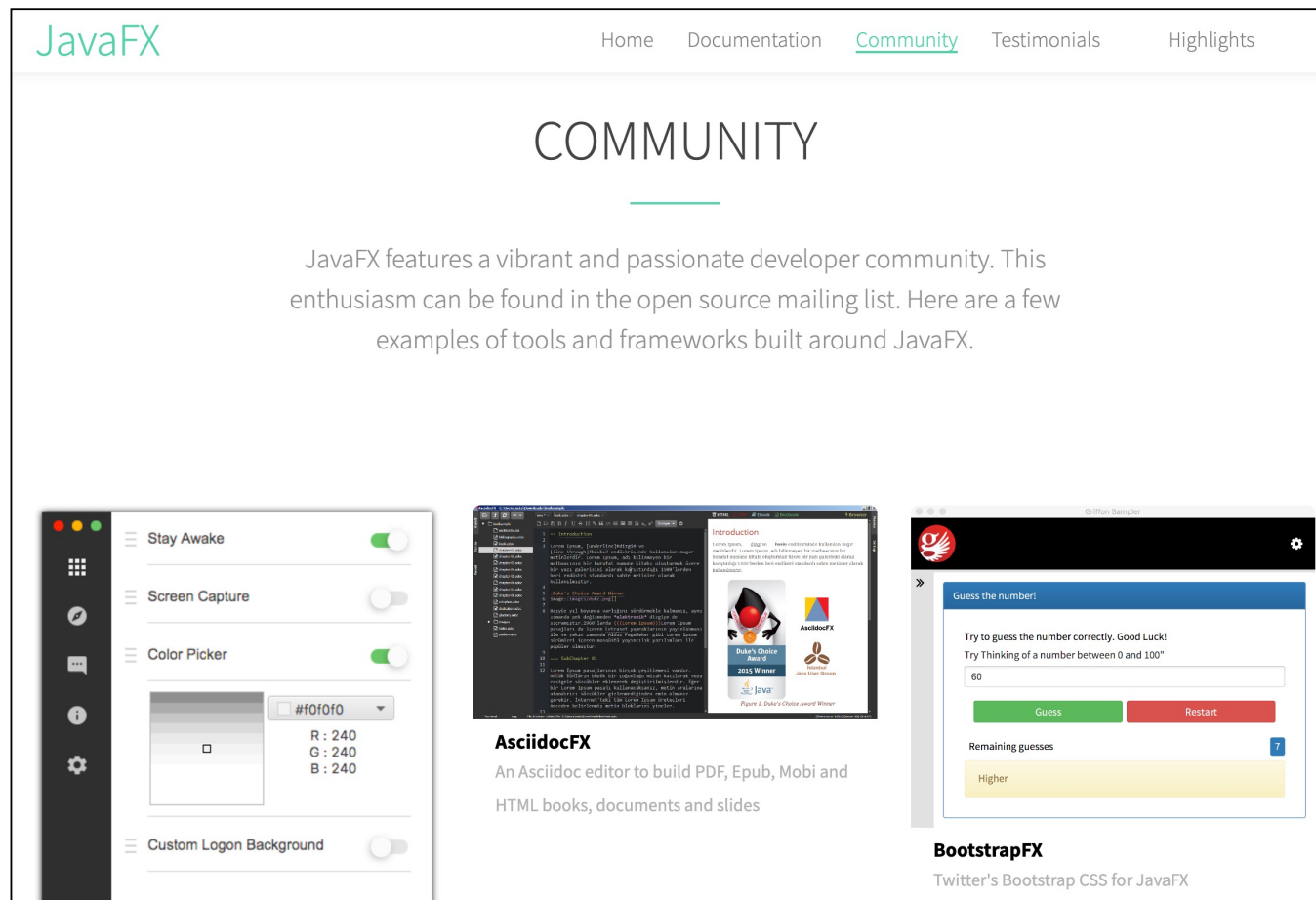




# Projeto OpenJFX

# Projeto OpenJFX

- A partir do Java 11, a Oracle tornou o JavaFX em um projeto open source (OpenJFX).



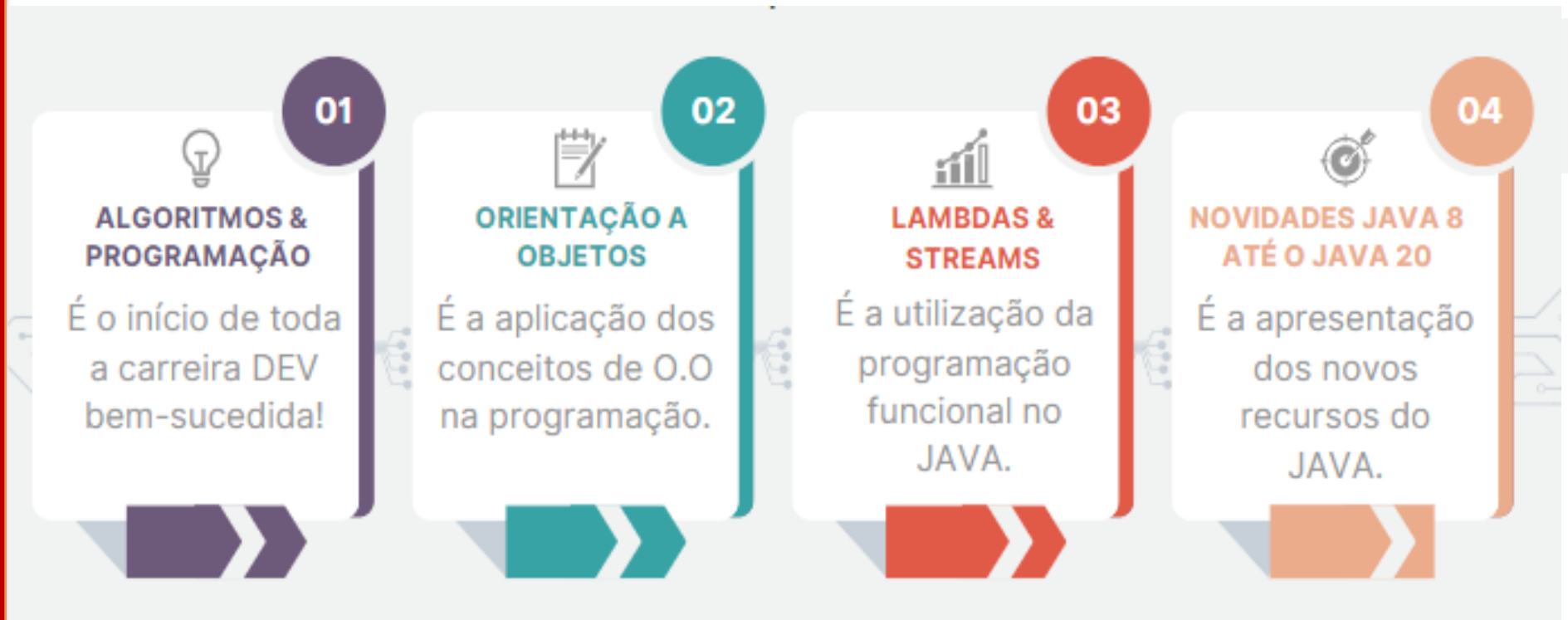
<https://openjfx.io/>

© ABC TREINAMENTOS | [www.abctreinamentos.com.br](http://www.abctreinamentos.com.br)



# Conclusão

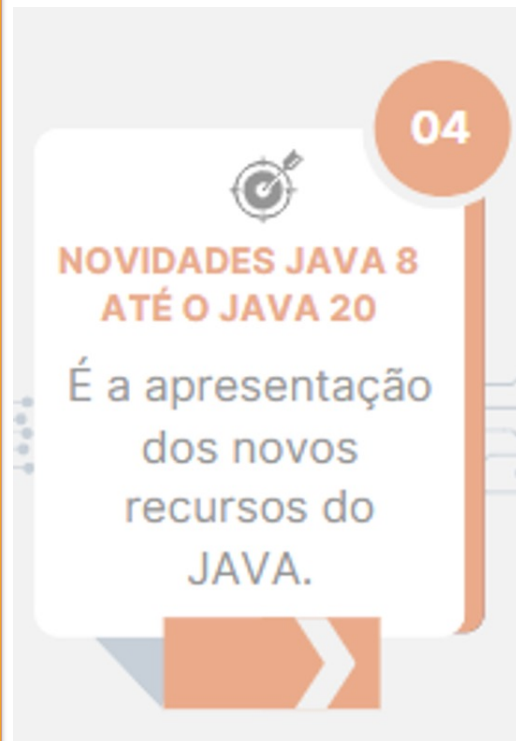
# A Nossa Trilha **DEV JAVA FULL STACK**



# A Nossa Trilha DEV JAVA FULL STACK



# A Nossa Trilha DEV JAVA FULL STACK



## Curso

### Domine as Inovações do Java com Spring Boot & MongoDB

As Mudanças mais Importantes do Java 8 até o Java 20



**Prof. Msc. Antonio B. C. Sampaio Jr**  
ENGENHEIRO DE SOFTWARE & PROFESSOR

@abctreinamentos  
@amazoncodebr

www.abctreinamentos.com.br  
www.amazoncode.com.br



# A Nossa Trilha DEV JAVA FULL STACK



07

  
**SPRING BOOT**

É o início de uma  
Carreira Vitoriosa  
como DEV JAVA  
FULL STACK.

Two red arrows pointing right.

Curso

## Aplicações JAVA com SPRING BOOT



**Prof. Msc. Antonio B. C. Sampaio Jr**  
ENGENHEIRO DE SOFTWARE & PROFESSOR

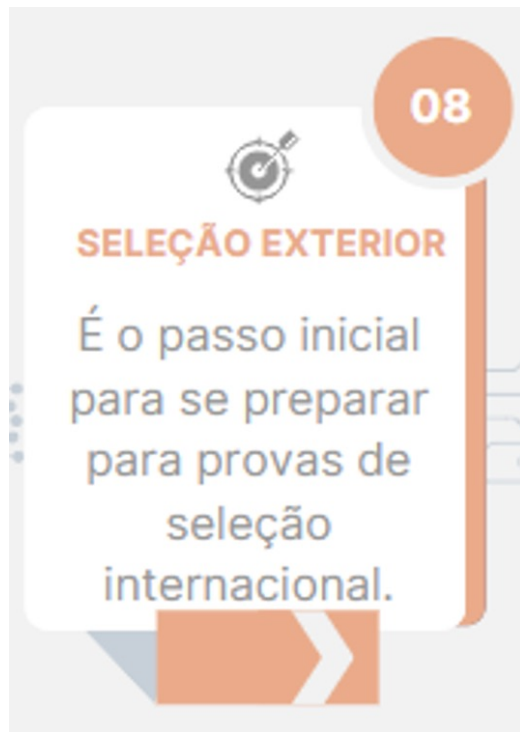
@abctreinamentos  
@amazoncodebr

www.abctreinamentos.com.br  
www.amazoncode.com.br





# A Nossa Trilha DEV JAVA FULL STACK



## Curso

### Aprenda a Resolver Questões Java para Seleção no Exterior

Resolução de 18 Questões de Java



**Prof. Msc. Antonio B. C. Sampaio Jr**  
ENGENHEIRO DE SOFTWARE & PROFESSOR

@abctreinamentos  
@amazoncodebr

www.abctreinamentos.com.br  
www.amazoncode.com.br

