

# Especialização em Desenvolvimento Java

## Programação de Interfaces Gráficas

Prof. Vinícius de Paula

<https://github.com/viniciusdepaula/aulas-pgui>



# **Objetivos desta aula**

# Objetivos desta aula

- Obter uma visão geral de componentes da API Swing;
- Conhecer alguns componentes: JFrame, JButton, JLabel, JTextField, JPanel, JCheckBox, JRadioButton;
- Entender como cooperar componentes com classes que recebem os eventos e atuam em regras de negócio;
- Apresentar uma visão geral sobre o padrão MVC;
- Entender como podemos aplicar o padrão MVC em uma aplicação Java utilizando Swing.

# Introdução

# Introdução

- Uma interface gráfica com o usuário apresenta uma interface visual para um programa.
- Em Java temos basicamente as APIs: Swing, AWT e SWT.
  - **Componentes AWT:** Label, Button, TextField, List, etc.
  - **Componentes Swing:** JLabel, JButton, JTextField, JList, etc.
  - **Componentes SWT:** Label, Button, Text, List, etc

# AWT (Abstract Window Toolkit)

- API padrão para criação de componentes GUI no início da plataforma Java (entre 1995 até 1998).
- Os objetos AWT são construídos sobre objetos de código nativo do Sistema Operacional em uso.
- Os componentes GUI originais do pacote `java.awt` estão diretamente associados com as capacidades de GUI da plataforma local.

# SWING

- API escrita puramente em Java.
- JLabel, JButton, JTextField, etc, são componentes GUI do pacote javax.swing. Os componentes do pacote Swing são escritos, manipulados e exibidos em Java.
- O aspecto e a funcionalidade de um programa Java (com GUI Swing) são conhecidos como “look and feel” do programa e podem ser modificados.
- Componentes podem ser estendidos ou modificados.

# Diagrama resumido dos componentes



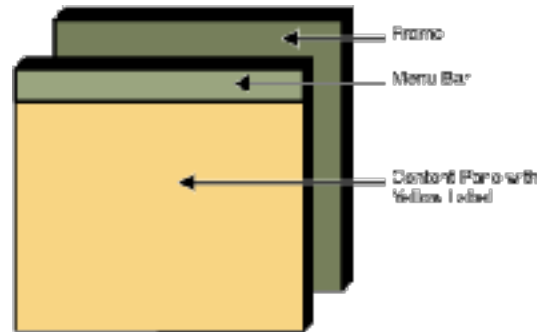


# SWING

- Componentes Swing: “pouca dependência da plataforma local”
- Componentes AWT: “dependência da plataforma local”
- Alguns componentes do Swing ainda são componentes com “dependência” (exemplo o JFrame).

# Frames ou JFrame

- Uma janela (ou quadro) de nível mais alto (que não fica contida dentro de outra janela) é um Frame ou, na versão Swing, um JFrame;
- Os quadros são contêineres.
  - Isso significa que um quadro pode conter outros componentes de interface com o usuário.



# **Primeira Aplicação utilizando a API Swing**

# Sobre o Netbeans IDE

- Ambiente integrado para o desenvolvimento de aplicações desktop, mobile e web.
  - <https://www.netbeans.org>
- Suporte ao desenvolvimento em diversas linguagens.
  - Java, HTML5, PHP e C++

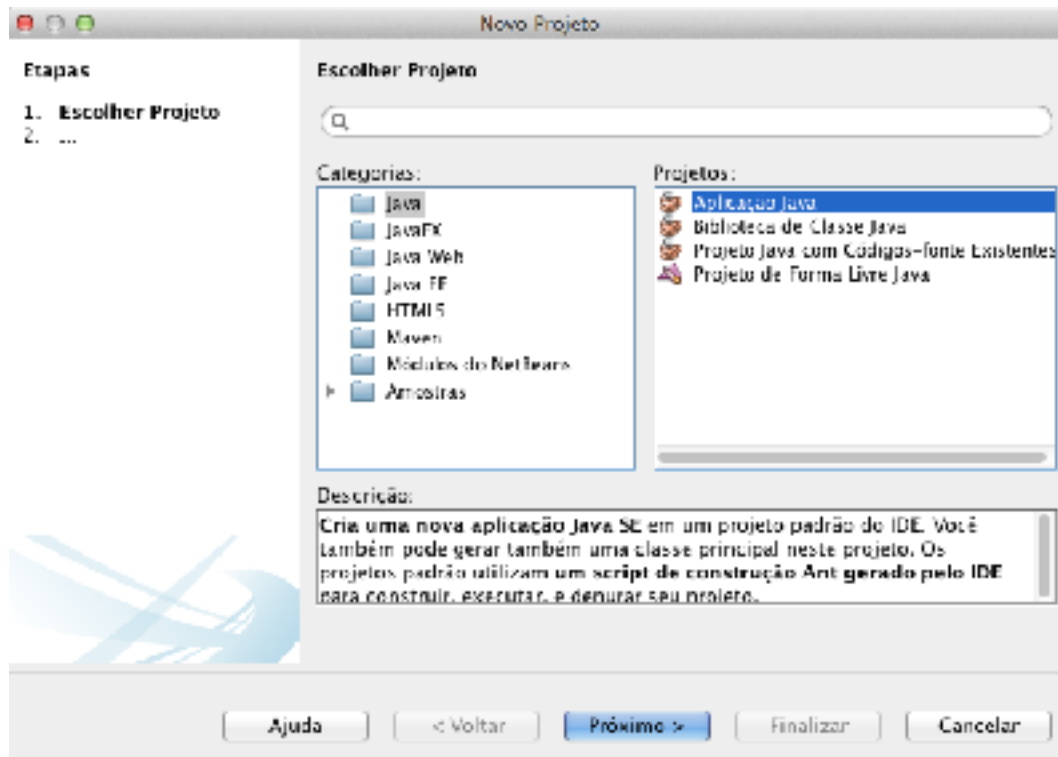
# Primeira Aplicação Swing

- Conversor de temperatura:
  - Celsius para Fahrenheit.

# Criação do projeto



# Definição do tipo de projeto



# Definição do nome e localização

**Novo Aplicação Java**

**Etapas**

1. Escolher Projeto
2. **Nome e Localização**

**Nome e Localização**

Nome do Projeto:

Localização do Projeto:

Pasta do Projeto:

☐ Usar Pasta Dedicada para Armazenar Bibliotecas

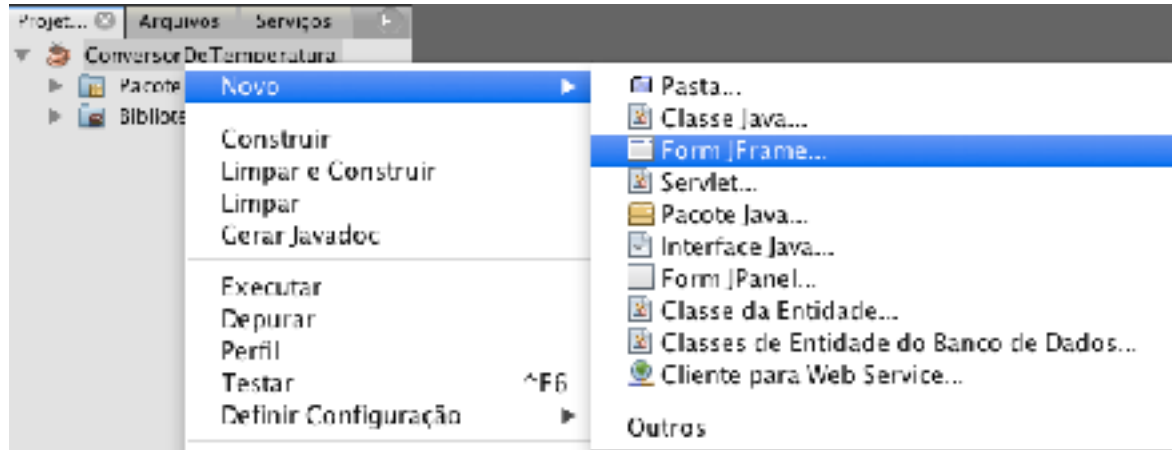
Pasta Bibliotecas:

Usuários e projetos diferentes podem compartilhar as mesmas bibliotecas de compilação (consulte a Ajuda para obter detalhes).

☒ Criar Classe Principal



# Adicionando o JFrame



JFrame é a classe da API Swing responsável pelo frame principal da aplicação.

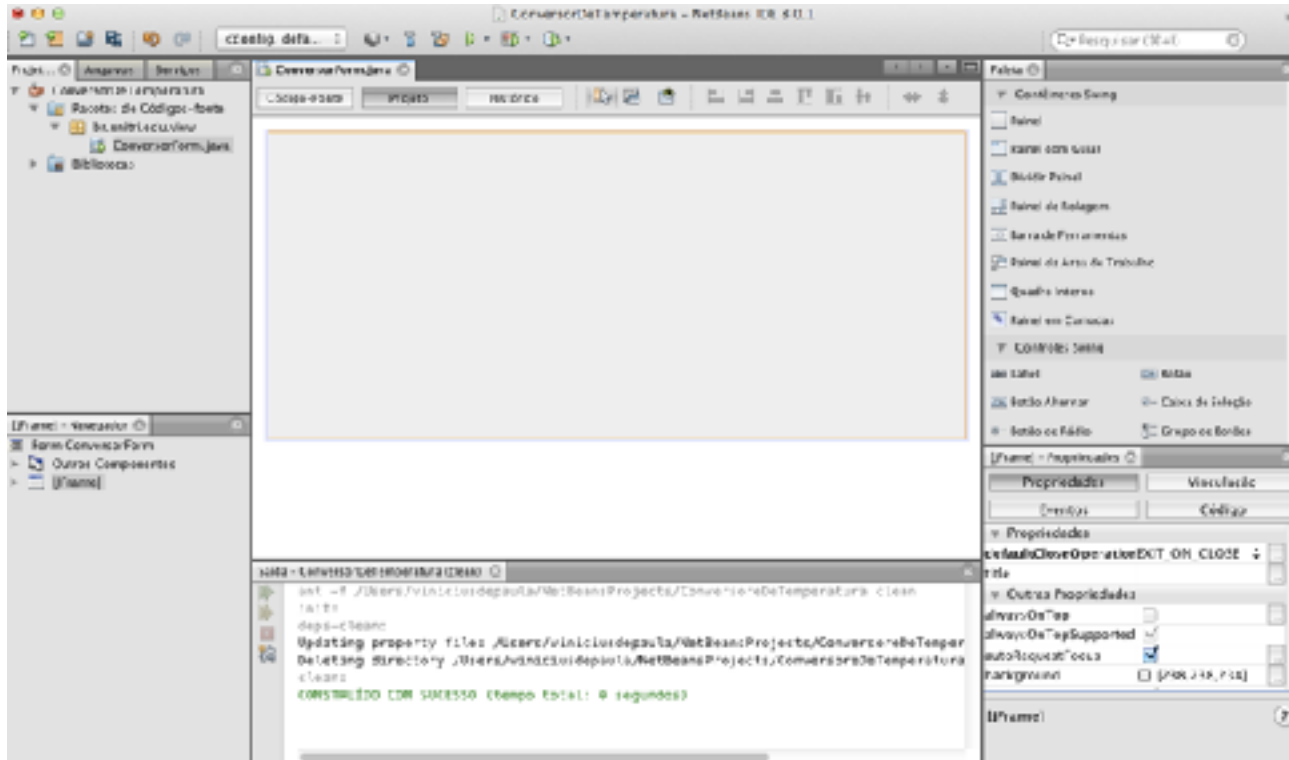
# Definição do nome e pacote da Classe

The image shows a 'New Form JFrame' dialog box with the following fields and values:

- Nome da Classe:** Conversor
- Projeto:** ConversorDeTemperatura
- Localização:** Pacotes de Códigos-fonte
- Pacote:** br.edu.unifri
- Arquivo Criado:** src\br\edu\unifri\Conversor.java

The 'Finalizar' button is highlighted in red.

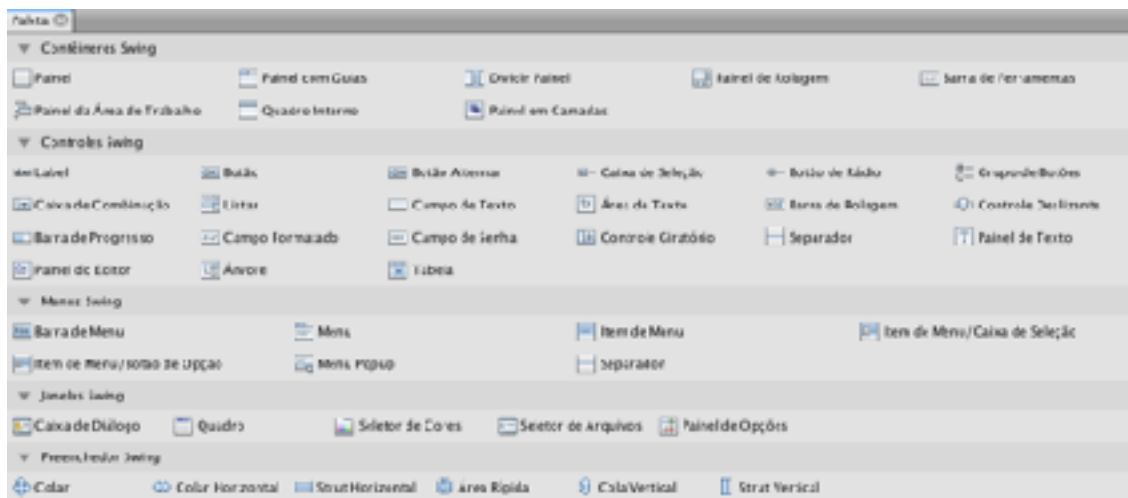
# Visão do projeto criado



# **Conceitos Fundamentais do NetBeans**

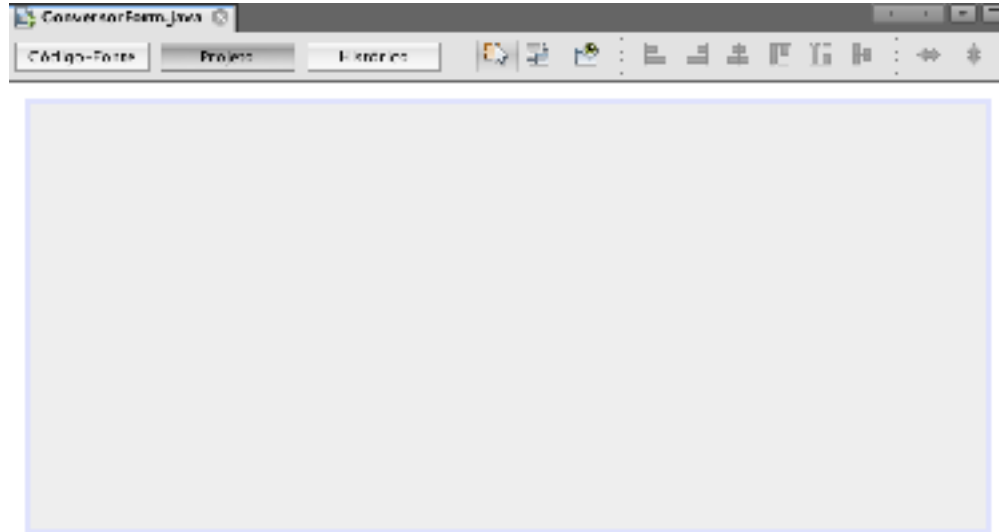
# Conceitos Fundamentais do NetBeans

- Paleta
  - Contém todos os componentes disponibilizados pela Swing API.



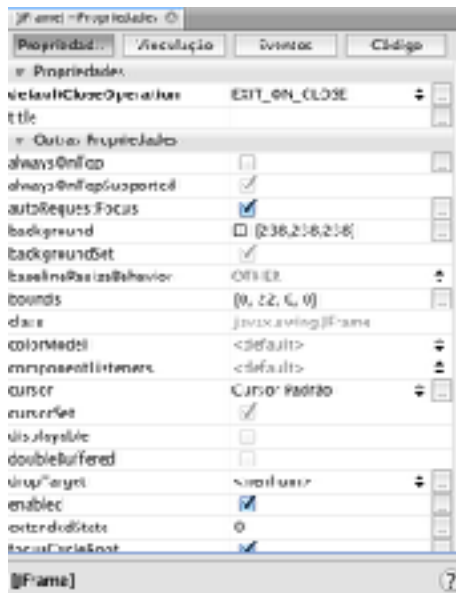
# Conceitos Fundamentais do NetBeans

- Área de Design
  - Ambiente visual para construção das telas
  - Possui duas visões: Projeto (Design) e Código Fonte



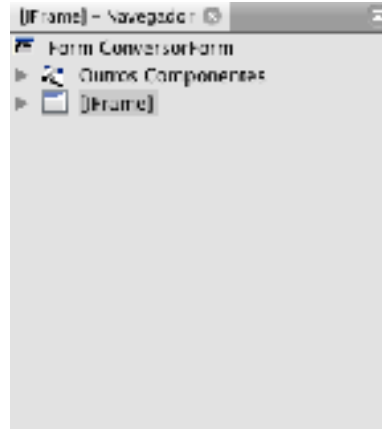
# Conceitos Fundamentais do NetBeans

- Editor de Propriedades
  - Permite a edição das propriedades dos componentes



# Conceitos Fundamentais do NetBeans

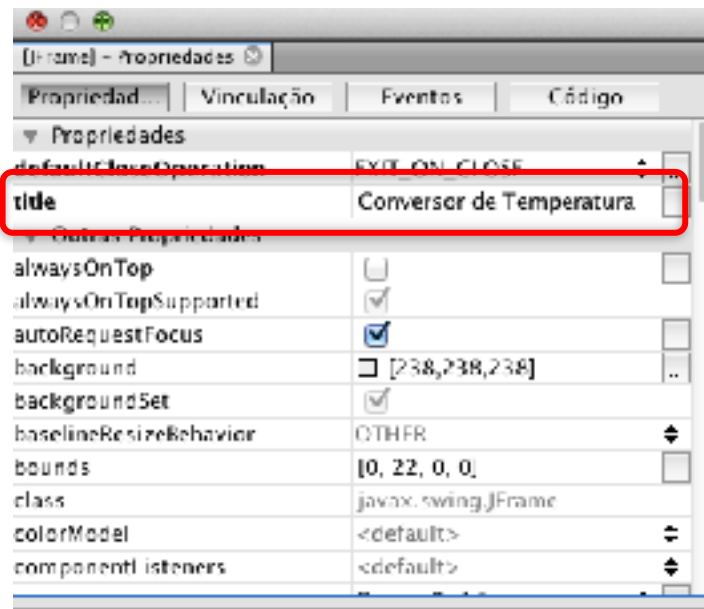
- Navegador
  - Representação hierárquica dos componentes da aplicação
  - Permite de forma simplificada a alteração dos nomes das variáveis



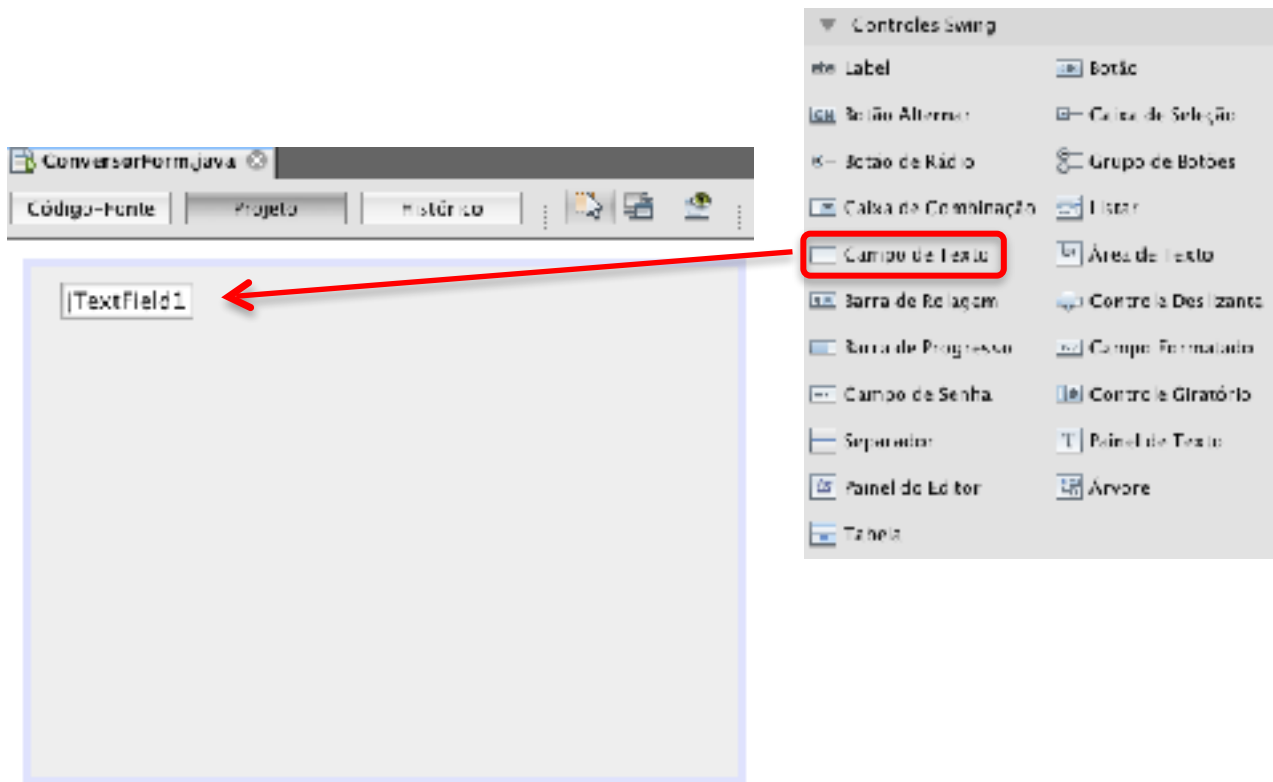


# **Aparência da Aplicação**

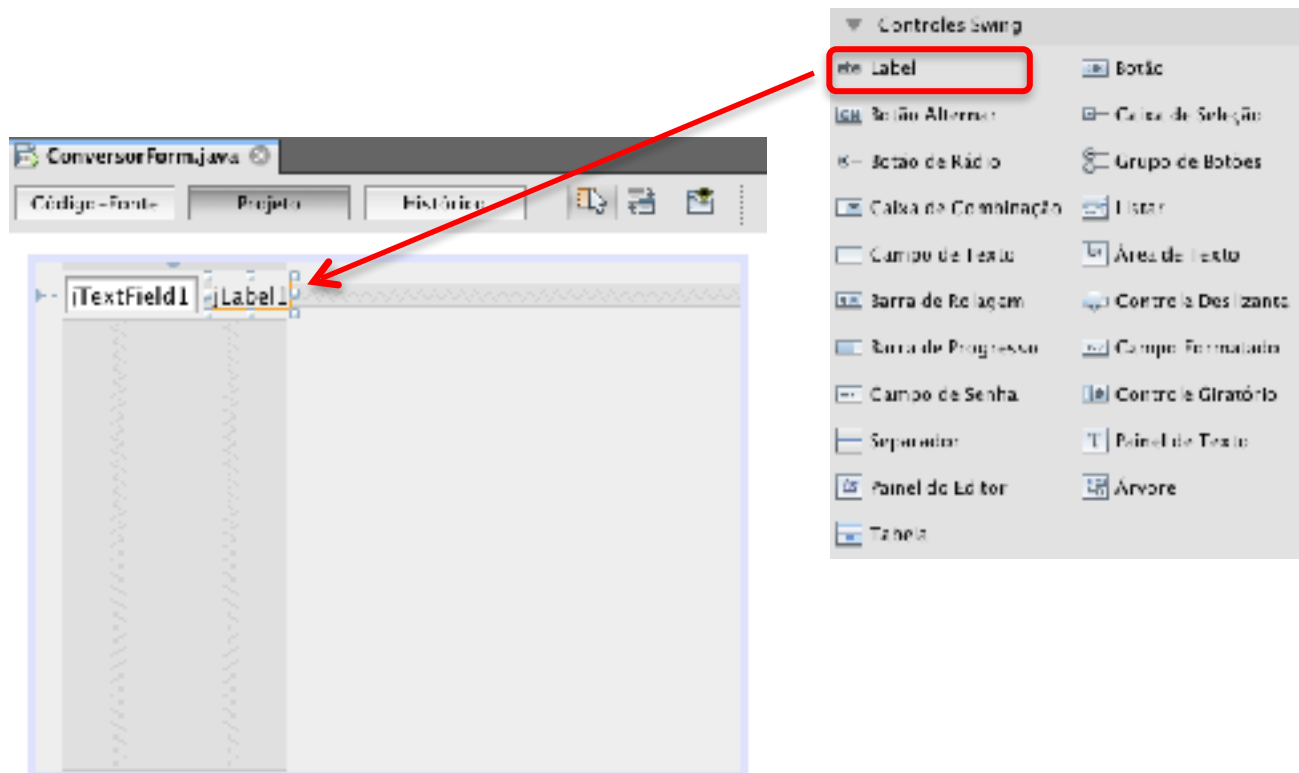
# Alteração do título da janela



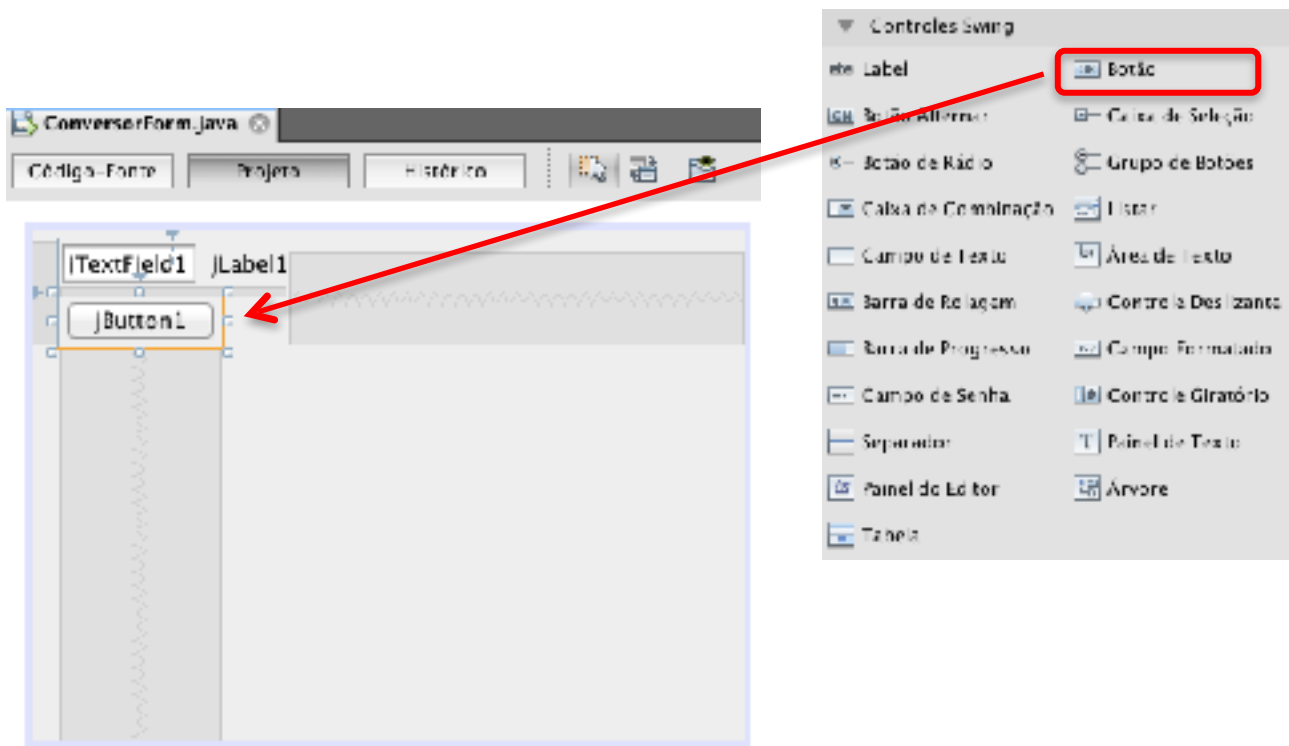
# Adicionando um JTextField



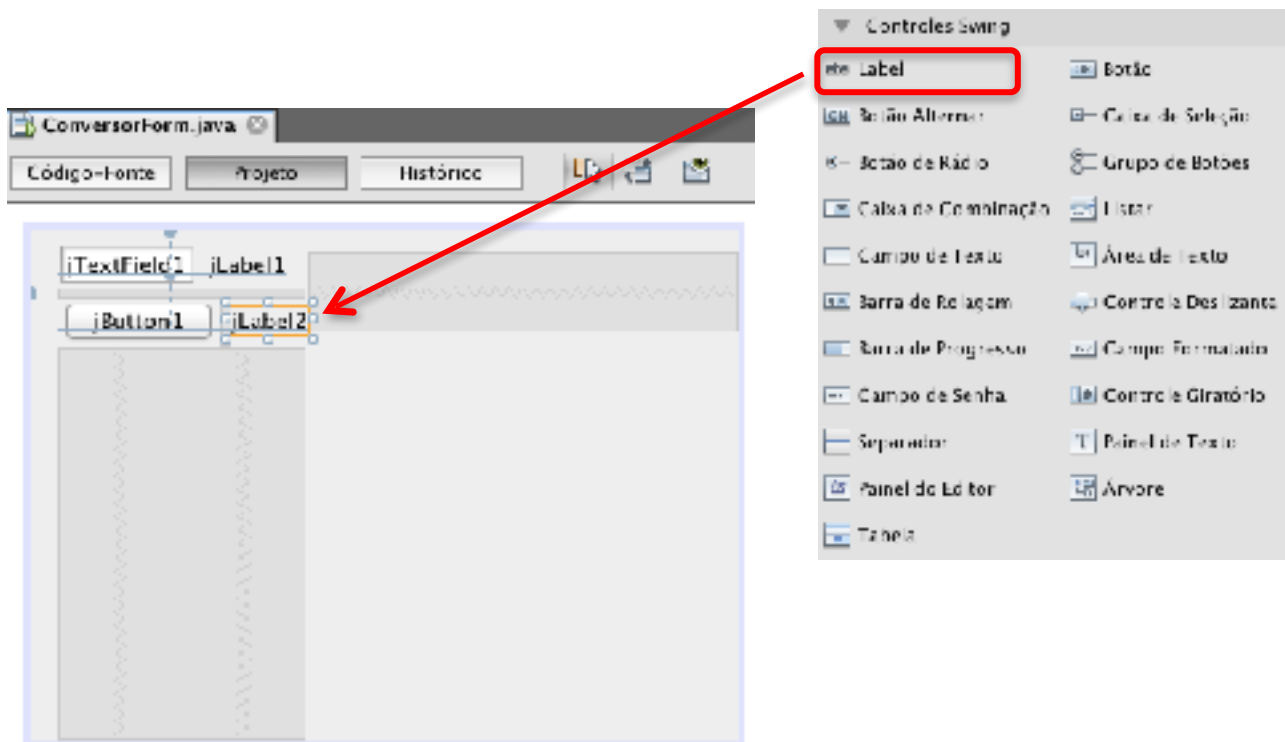
# Adicionando um JLabel



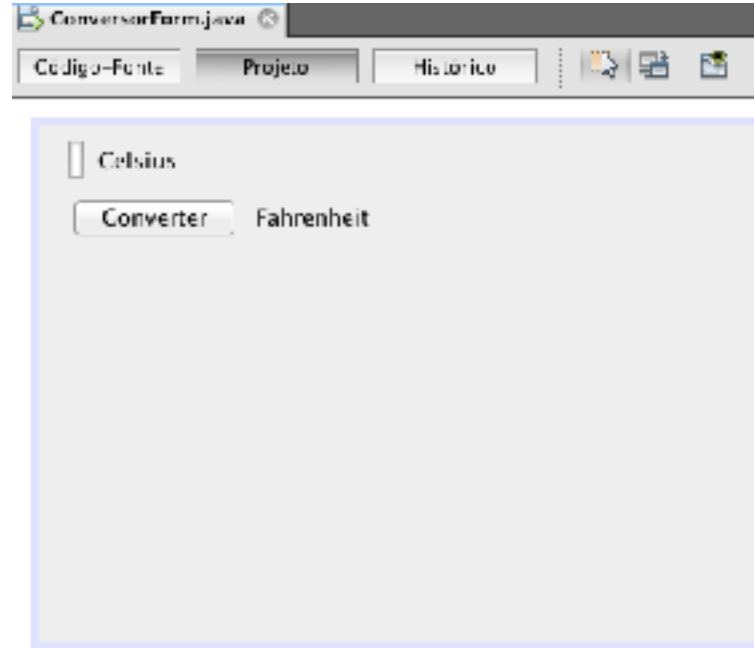
# Adicionando um JButton



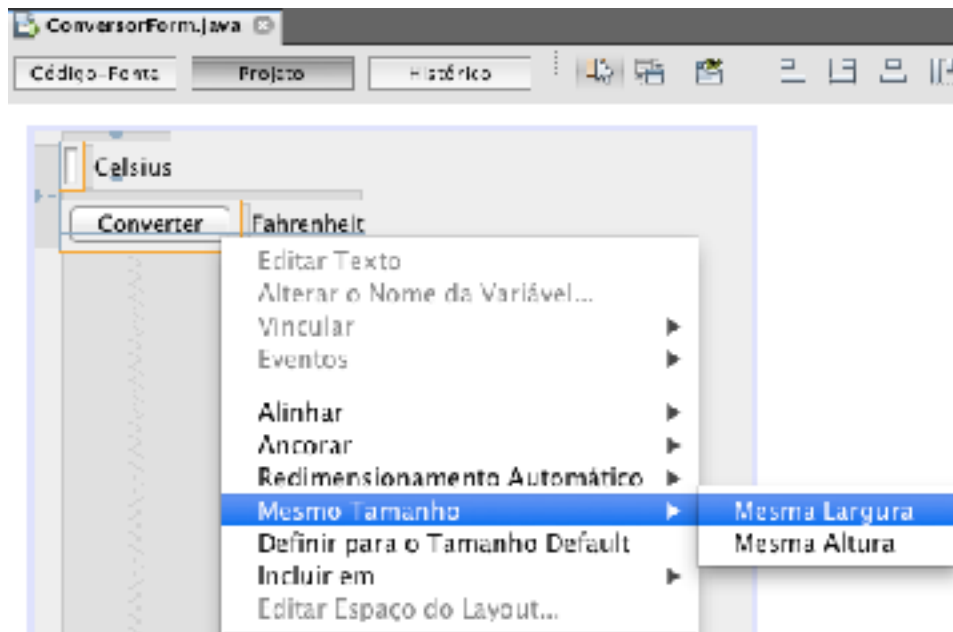
# Adicionando o segundo JLabel



# Alterar o texto dos componentes

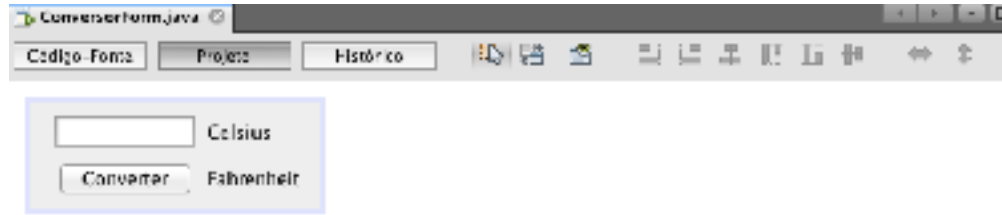


# Definir a largura dos componentes





# Eliminar os espaços extras do JFrame



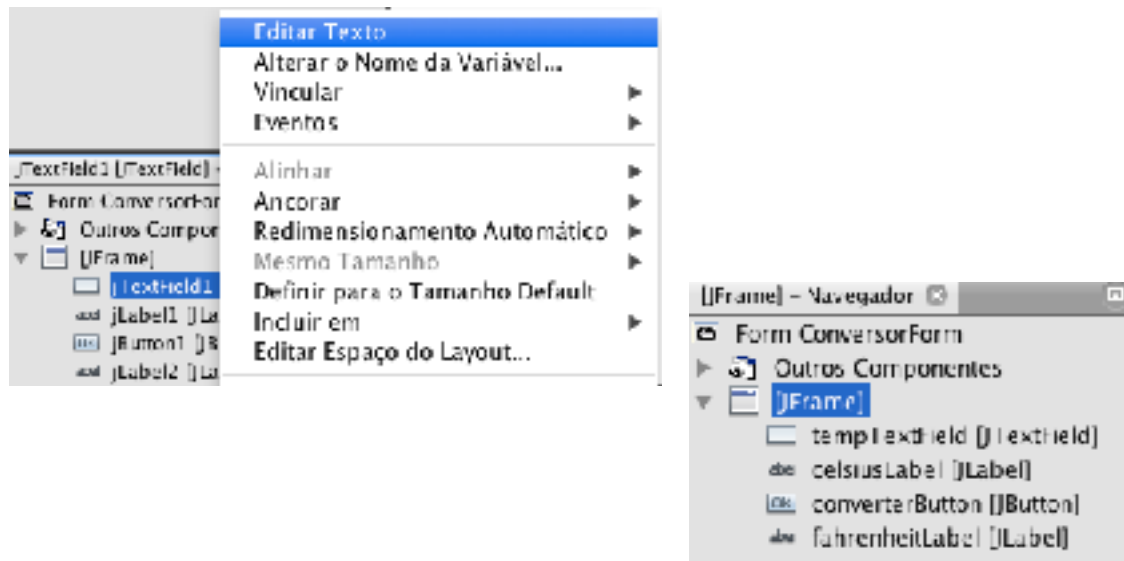
# Código gerado pelo NetBeans

```
25  ~|
26  @SuppressWarnings("inherited")
27  // @SuppressWarnings("unchecked")
28  private void initComponents() {
29
30      JTextField jTextF1 = new javax.swing.JTextField();
31      JLabel jLabel1 = new javax.swing.JLabel();
32      JButton jButton1 = new javax.swing.JButton();
33      JLabel jLabel2 = new javax.swing.JLabel();
34
35      setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
36      setTitle("Conversor de Temperatura");
37
38      JTextField jTextF1.addActionListener(new javax.swing.ActionListener() {
39          public void actionPerformed(java.awt.event.ActionEvent evt) {
40              jTextF1ActionPerformed(evt);
41          }
42      });
43
44      jLabel1.setText("Celsius");
45
46      jButton1.setText("Converter");
47
48      jLabel2.setText("Fahrenheit");
49
50      javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
51      getContentPane().setLayout(layout);
52      layout.setHorizontalGroup(
53          layout.createParallelGroup(Group)
54              .addGroup(layout.createSequentialGroup()
55                  .addGroup(layout.createParallelGroup(Group)
56                      .addComponent(jTextF1)
57                      .addComponent(jLabel1)
58                      .addComponent(jButton1)
59                      .addComponent(jLabel2))
60                  .addContainerGap())
61      );
62  }
```

# **Lógica da Aplicação**

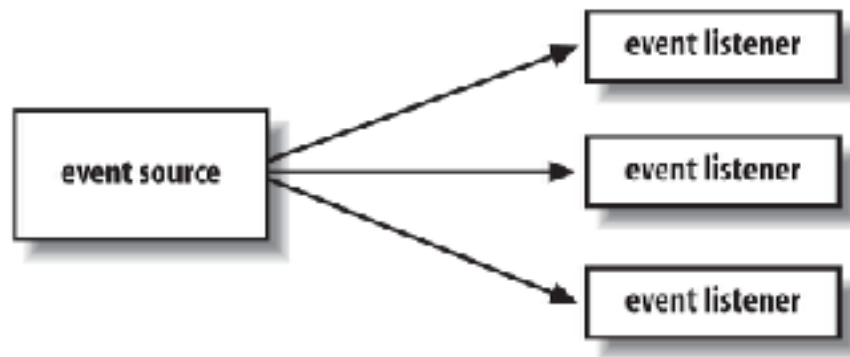
# Identificador padrão das variáveis

- A alteração do identificador padrão das variáveis pode ser feita utilizando o Navegador de componentes.

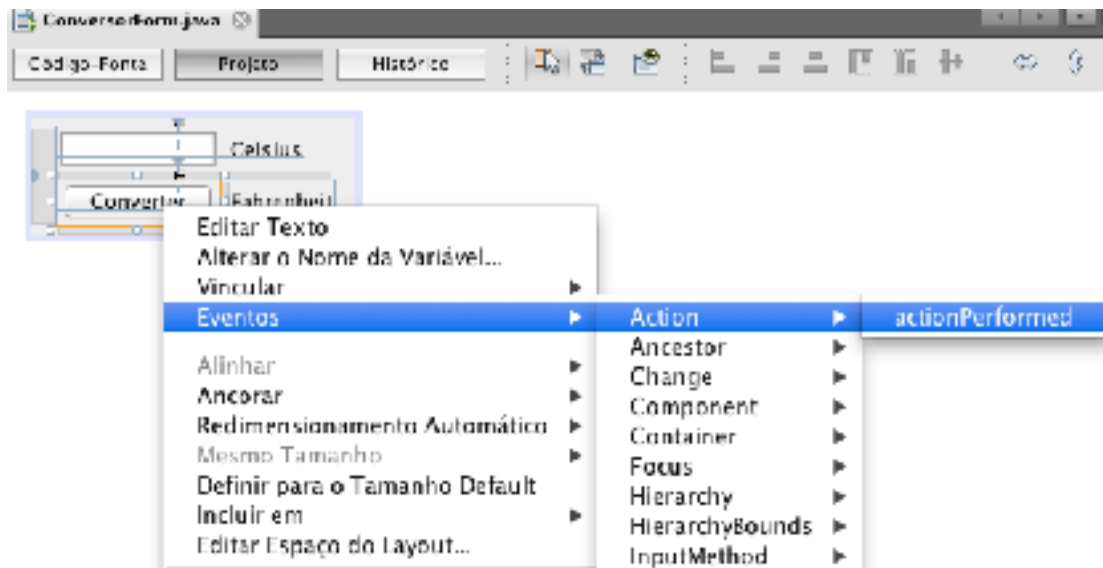


# Tratativa de eventos

- Qualquer tipo de interação do usuário com um componente gráfico da API Swing gerará um *event object*.
  - É possível manipular um *event object* adicionando *listeners* de acordo com o evento a ser tratado.

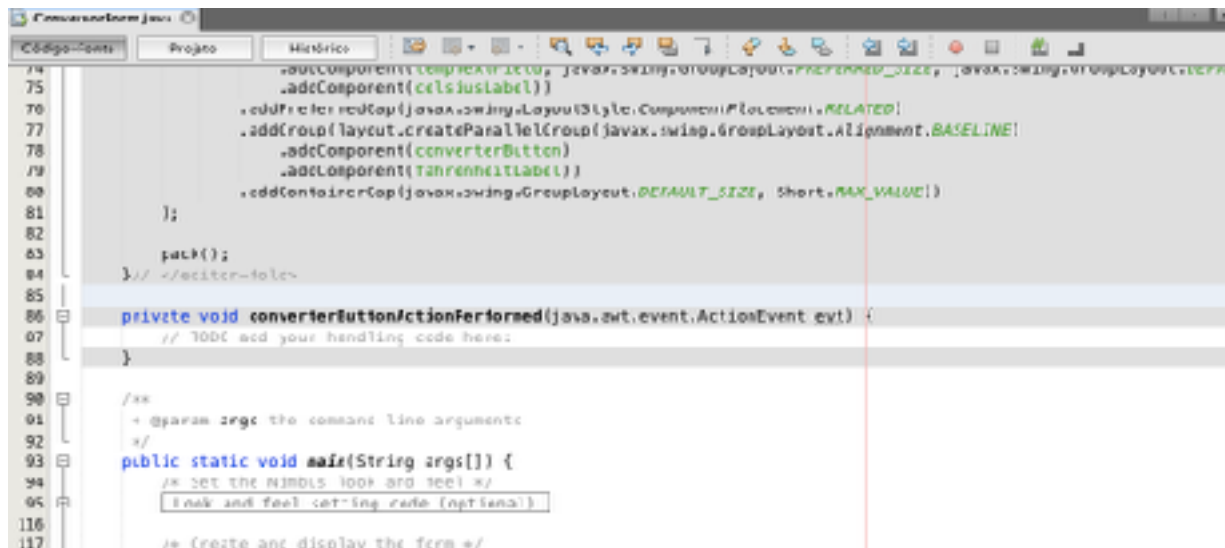


# Registrando um event listener



O método *actionPerformed(actionEvent)* é chamado após o usuário executar algum tipo de ação (ex.: clicar em um botão).

# Registrando um event listener



```
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
116  
117  
    .addComponent(celsiusLabel, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE)  
    .addComponent(fahrenheitLabel, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE)  
    .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)  
        .addComponent(converterButton)  
        .addComponent(fahrenheitLabel)  
    )  
    .addContainerLayout(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)  
    );  
    pack();  
} // </editor-fold>  
  
private void converterButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}  
  
/**  
 * @param args the command line arguments  
 */  
public static void main(String args[]) {  
    /* Set the Nimbus look and feel */  
    /* Look and feel set to Mac OS X (optional) */  
    /* Create and display the form */  
}
```

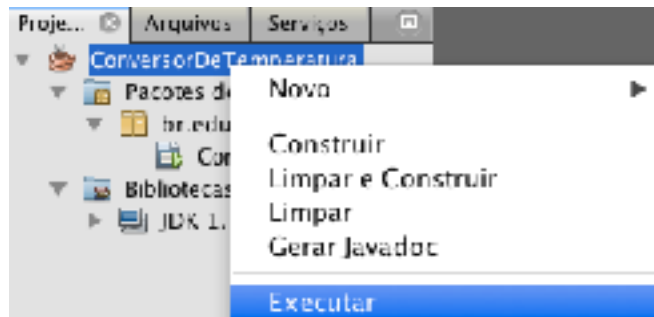
# Implementação da tratativa do evento

- Adicione o seguinte trecho de código no corpo do método *converterButtonActionPerformed*.

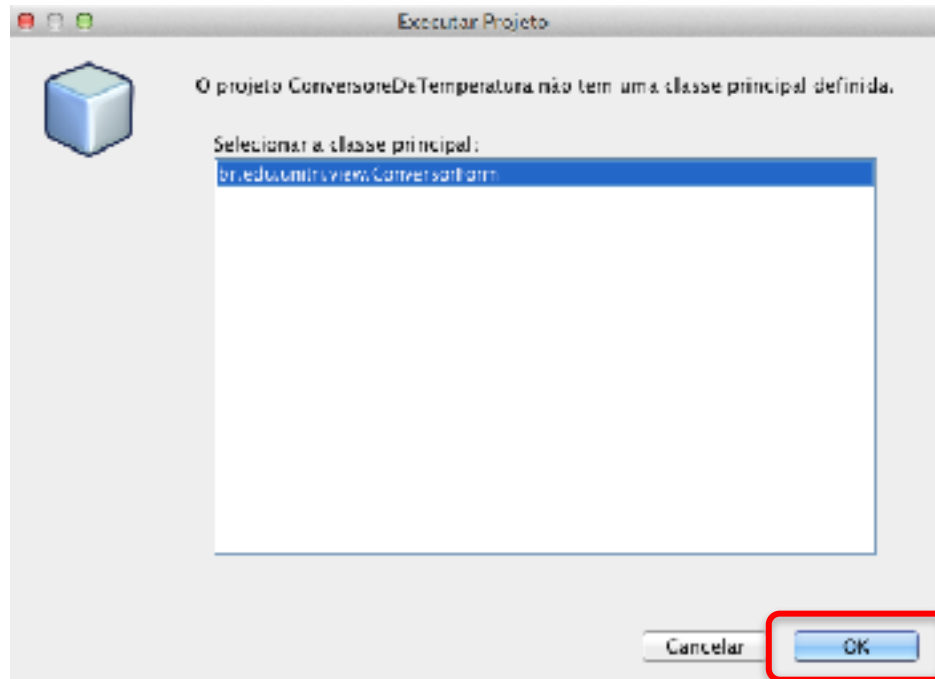
```
private void converterButtonActionPerformed(java.awt.event.ActionEvent evt) {  
     int tempFahr = (int) ((Double.parseDouble(tempTextField.getText()) * 1.8 + 32);  
    fahrenheitLabel.setText(tempFahr + " Fahrenheit");  
}
```



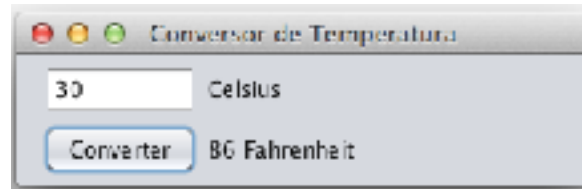
# Execução da aplicação



# Execução da aplicação



# Visual da aplicação criada

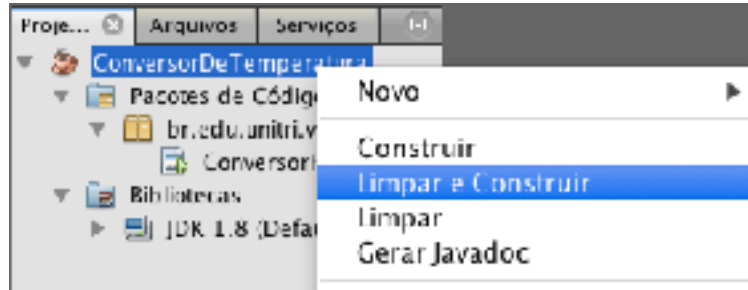


# **Criação do JAR da Aplicação**

# O quê é um JAR?

- O arquivo **jar** ou **Java AR**chive, possui todas as classes e os arquivos de configurações da aplicação em formato compactado.
  - Algumas características deste arquivo:
    - Similar a um arquivo zip;
    - Cross-platform;
    - Padrão aberto e escrito puramente em Java;
    - Especificação definida em [JAR File Specification](#).

# Criando um JAR com o NetBeans



O arquivo .jar será gerado dentro do diretório *dist* do projeto (ConversorDeTemperatura/dist/ConversorDeTemperatura.jar)

# Como executar um arquivo JAR?

- Abra o prompt de comando ou terminal e digite o seguinte comando:  
`java -jar Nome_Artefato.jar`

# Prática I



# Prática

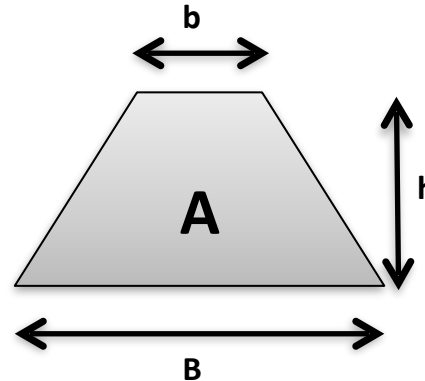


- 1) Desenvolva uma aplicação Swing para calcular a área de um trapézio. Sabendo-se que esta área pode ser calculada pela expressão:

$$A = (B + b) * h/2$$

Onde:

- B é a medida da base maior
- b é a medida da base menor
- h é a medida da altura



# Prática



- 2) Manoel contraiu um empréstimo de R\$1.730,00 a uma taxa de juros simples de 38% a.a. Sabendo que o empréstimo foi pago após 10 meses, desenvolva um programa que calcule qual foi o valor dos juros pagos por Manoel.

# Prática



- 3) O IMC – Índice de Massa Corporal é um critério da Organização Mundial de Saúde para dar uma indicação sobre a condição de peso de uma pessoa adulta. A fórmula para calcular o IMC é:

$$\text{IMC} = \text{peso} / (\text{altura})^2$$

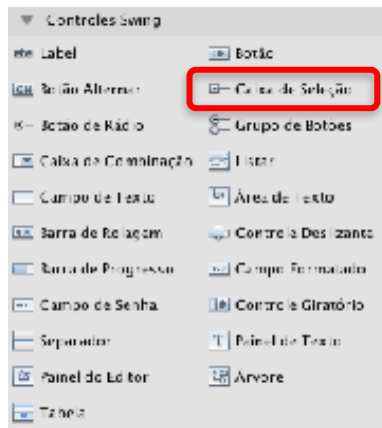
Elabore um algoritmo que leia o peso e a altura de um adulto e mostre sua condição de acordo com a tabela abaixo.

IMC	Classificação
Menor que 18,5	Abaixo do peso
Entre 18,5 e 24,9	Peso normal
Entre 25 e 29,9	Sobrepeso
Igual ou acima de 30	Obesidade

# **Outros Componentes da API Swing**

# JCheckBox

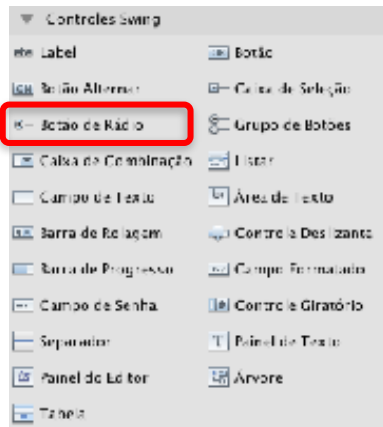
- Classe utilizada para a implementação das caixas de seleção.
  - Permite a seleção de mais de uma opção



O método `isSelected()` retornará true caso o checkbox seja selecionado.

# JRadioButton

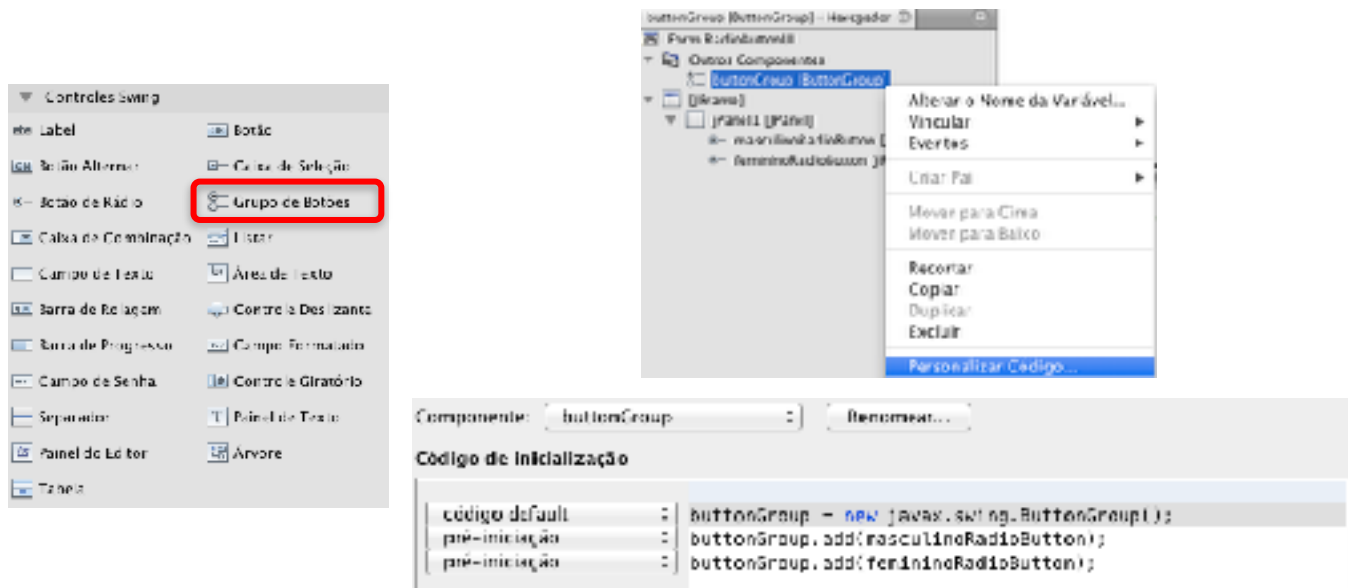
- Classe utilizada para a implementação dos botões de opção.
  - Permite a seleção de uma opção (com grupos de botões) ou mais de uma opção.



O método `isSelected()` retornará true caso o radio button seja selecionado.

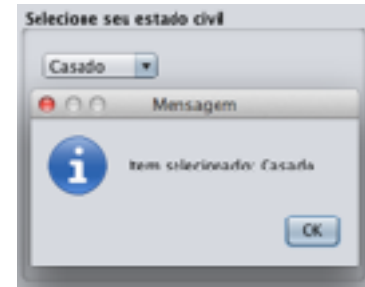
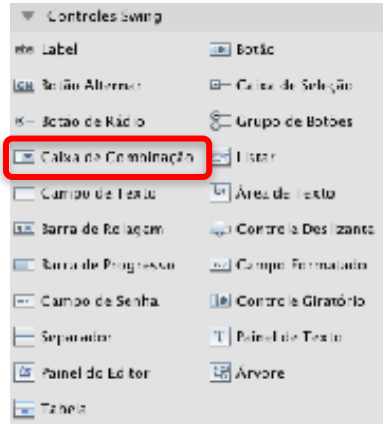
# JRadioButton

- Para agrupar os botões de seleção o componente *ButtonGroup* deve ser utilizado.



# JComboBox

- Classe utilizada para a implementação de combo boxes.

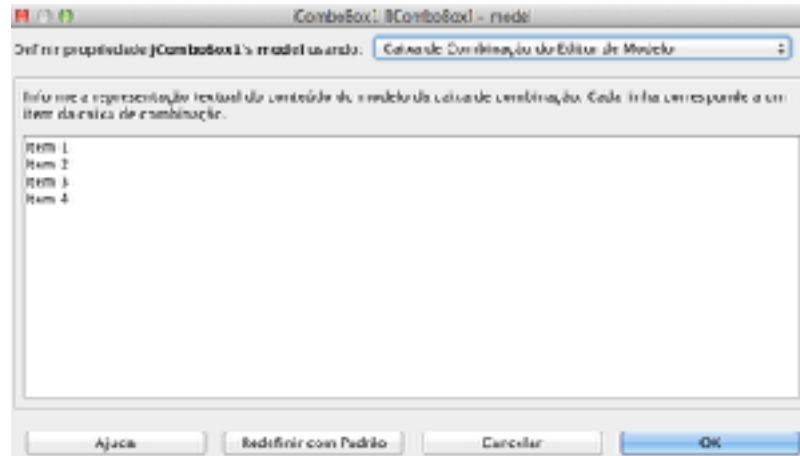
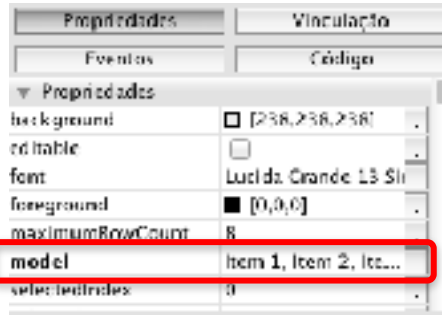


O método `getSelectedItem()` retorna o item que foi selecionado.



# JComboBox

- O combobox editor pode ser utilizado para edição da lista de itens.



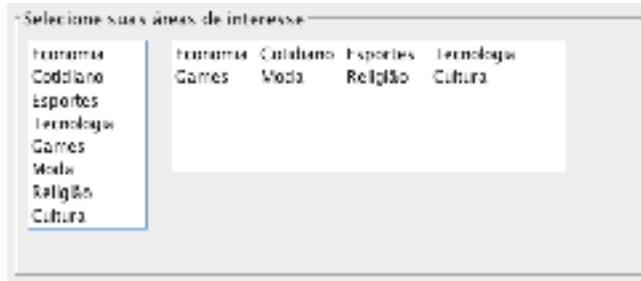
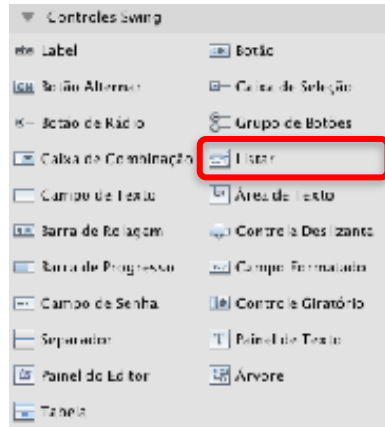
# JComboBox

- Os itens do combo podem ser preenchidos utilizando um Array.

```
public JComboBoxUI() {  
    initComponents();  
  
    String[] ArrayEstadoCivil = {"Casado", "Solteiro", "Divorciado", "Viúvo"};  
  
    for(int i=0; i<ArrayEstadoCivil.length; i++) {  
        estadoCivilComboBox.addItem(ArrayEstadoCivil[i]);  
    }  
}
```

# JList

- Classe responsável por implementar um grupo de itens que podem ser apresentados em uma ou mais colunas.



A propriedade `LayoutOrientation` define a orientação da lista. Já a propriedade `VisibleRowCount` define a quantidade de linhas a serem apresentadas.

# Prática II

# Prática



Selecione um ou mais itens

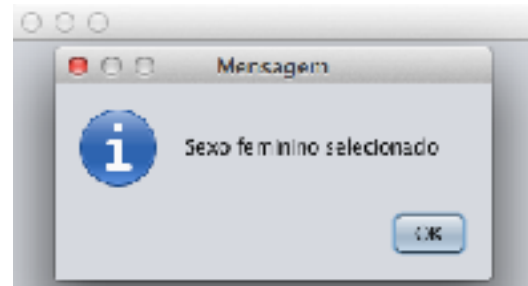
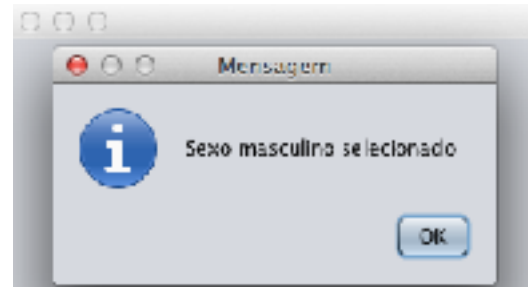
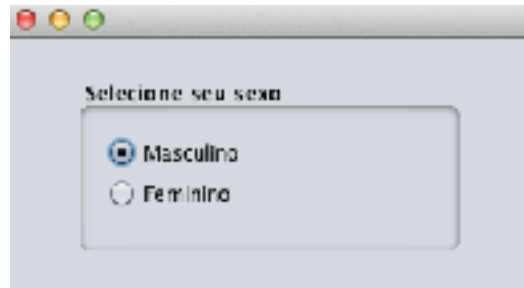
- ☐ Item 1
- ☐ Item 2
- ☐ Item 3

Selecione um ou mais itens

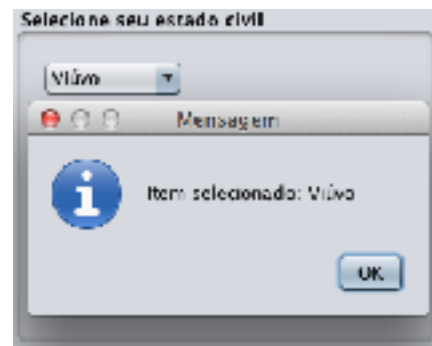
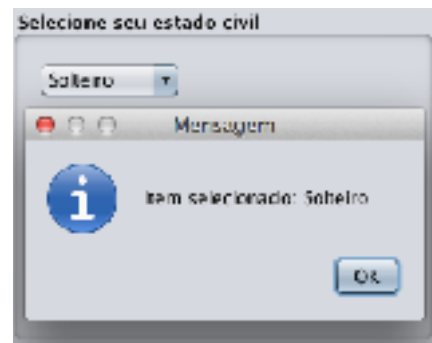
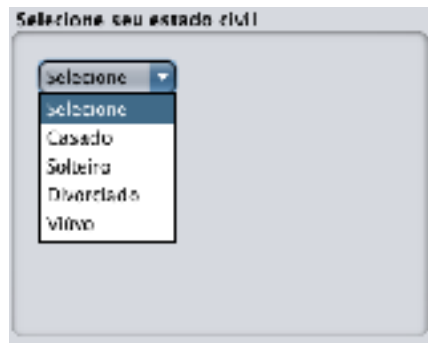
- ☒ Item 1
- ☒ Item 2
- ☒ Item 3

Item 1 selecionado  
Item 2 selecionado  
Item 3 selecionado

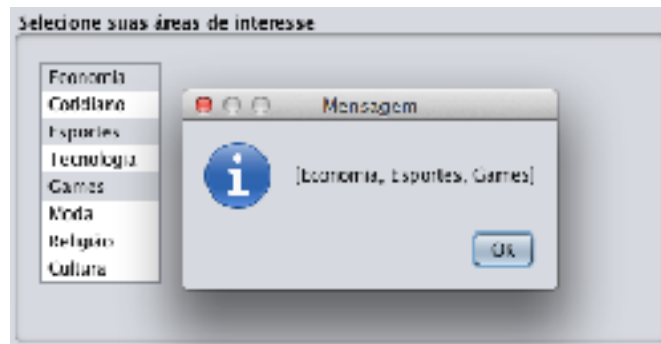
# Prática



# Prática



# Prática





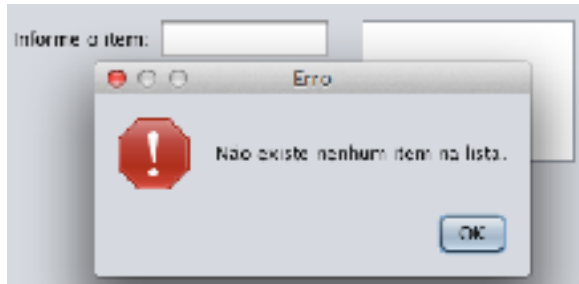
# Prática



Informe o item:

>>>

<<<



Informe o item:

>>>

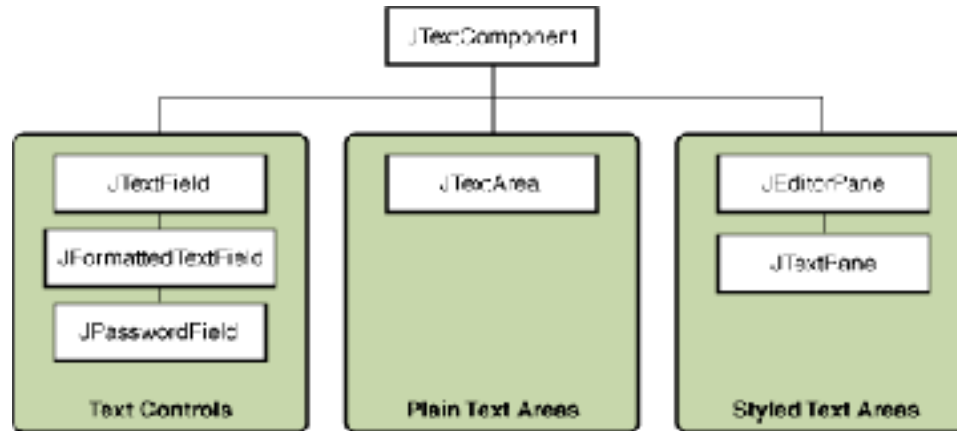
<<<

Item 1  
Item 2  
Item 3

# **Outros Componentes da API Swing**

# JTextComponent

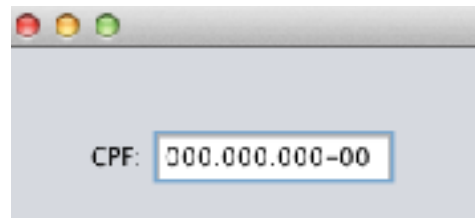
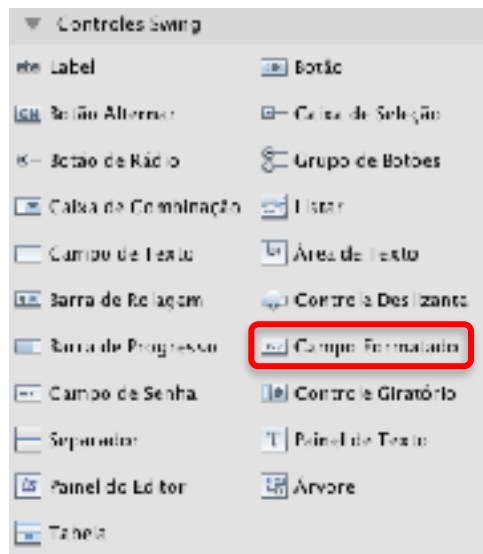
- Todos os componentes de texto do Swing herdam suas características desta superclasse.



**Hierarquia de JTextComponent**

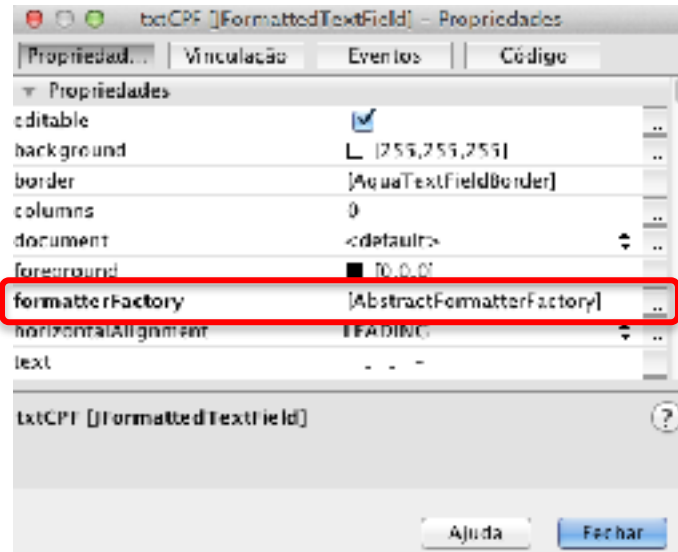
# JFormattedTextField

- Classe que herda as características de JTextField e implementa o mascaramento do *text field*.

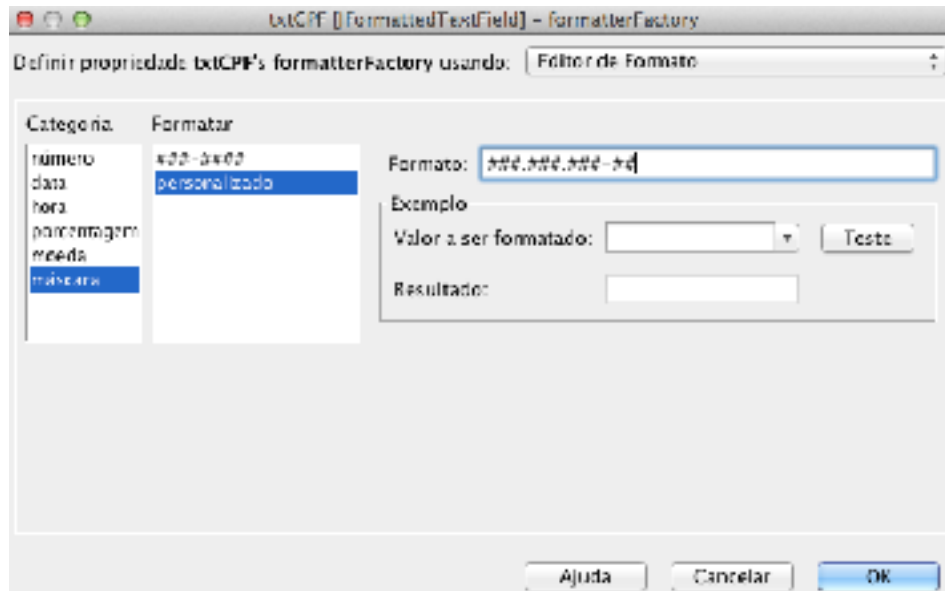


# JFormattedTextField

- Para formatar um *text field* é necessário definir um *formatterFactory*.



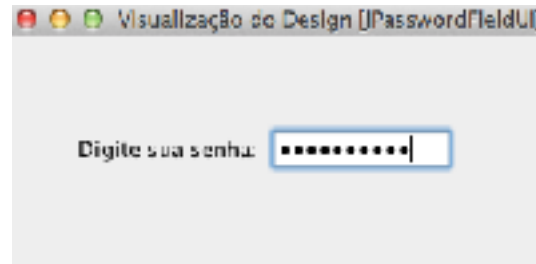
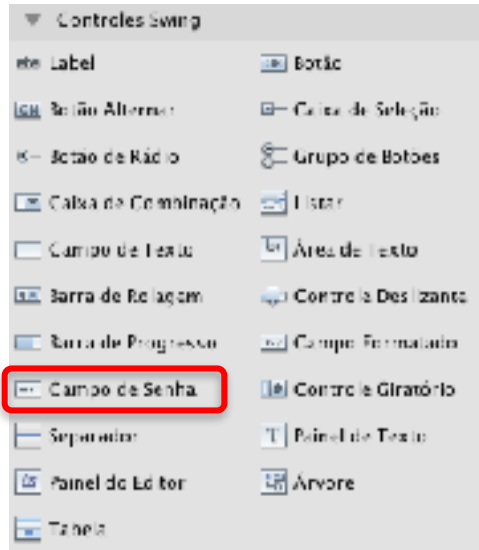
# JFormattedTextField



**Exemplo de definição de máscara para o CPF**

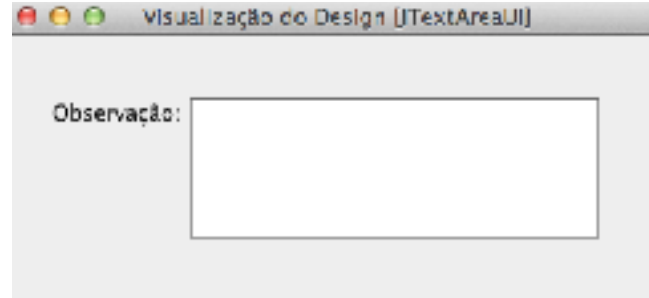
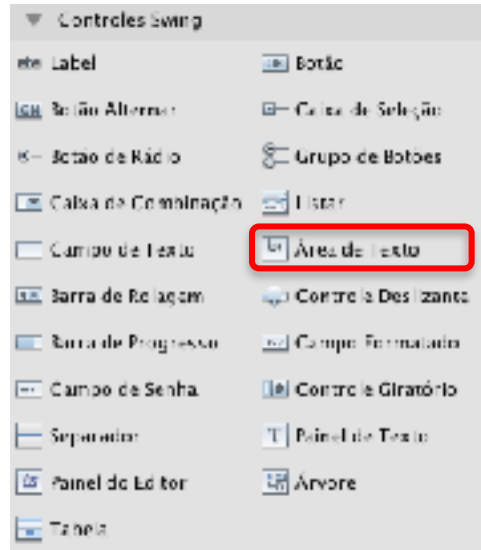
# JPasswordField

- Subclasse de JTextField que implementa o ocultamento dos caracteres digitados.



# JTextArea

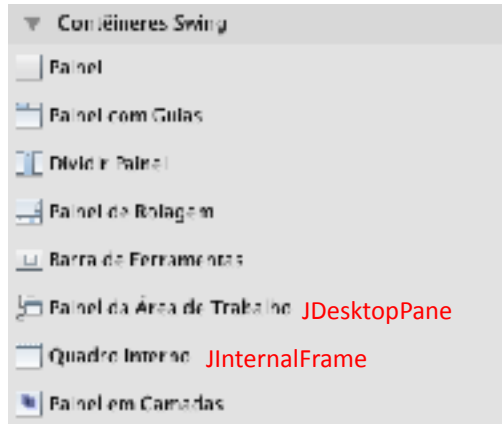
- Subclasse de JTextField que implementa um *text field* com múltiplas linhas para inclusão de texto.





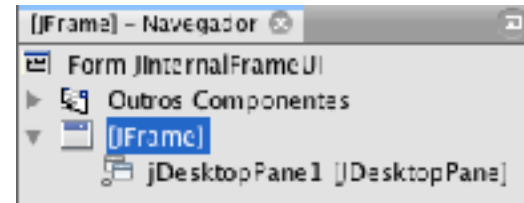
# JInternalFrame

- Classe que implementa a utilização de frames internos, um frame dentro de outro frame (principal).
  - Frames internos (JInternalFrame) devem ser adicionados em um *desktop pane* (instância de JDesktopPane).



# InternalFrame

- Quando utilizamos um *desktop pane* é necessário definir um tipo de layout, através do *Navegador* de componentes.

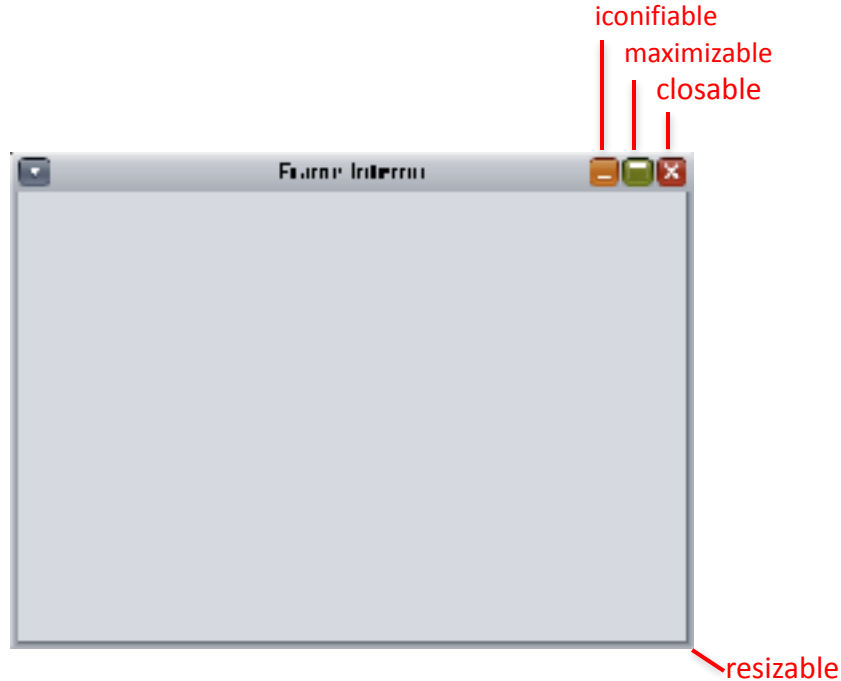


# InternalFrame



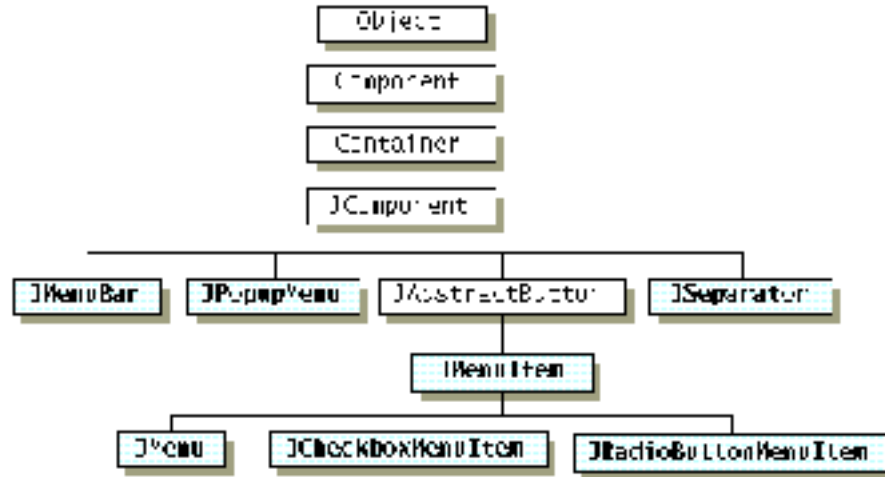
# JInternalFrame

- Algumas propriedades dos frames internos.



# Menu de Navegação Vertical

- O diagrama abaixo apresenta a hierarquia das classes relacionadas a implementação de menus.

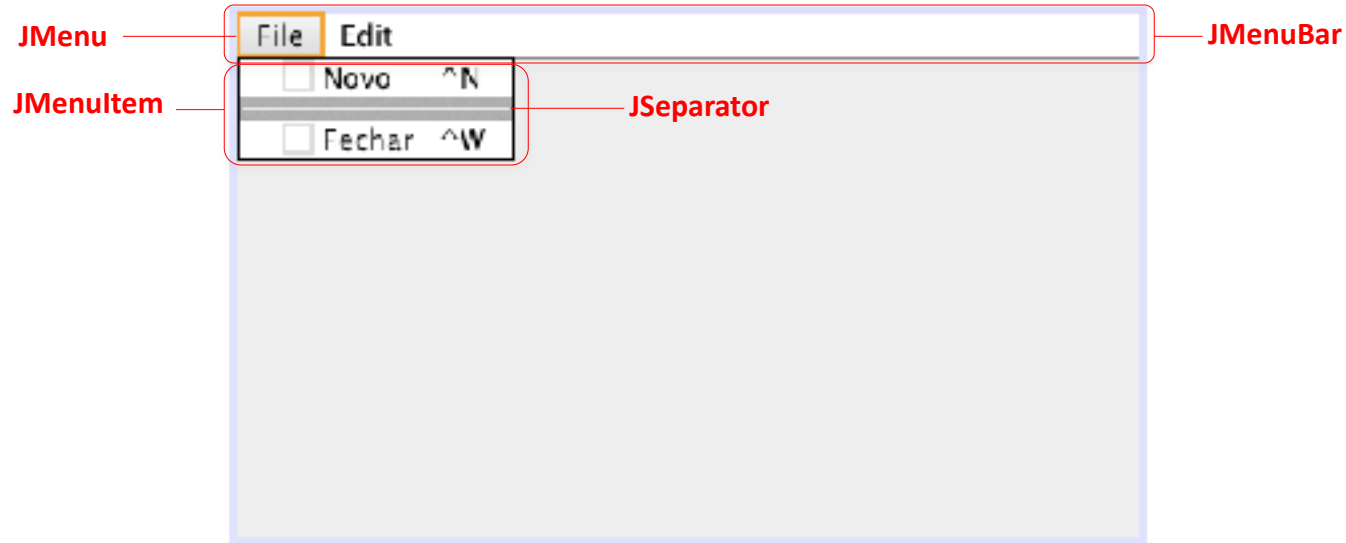


# Menu de Navegação Vertical

- Componentes Swing para construção de um menu:



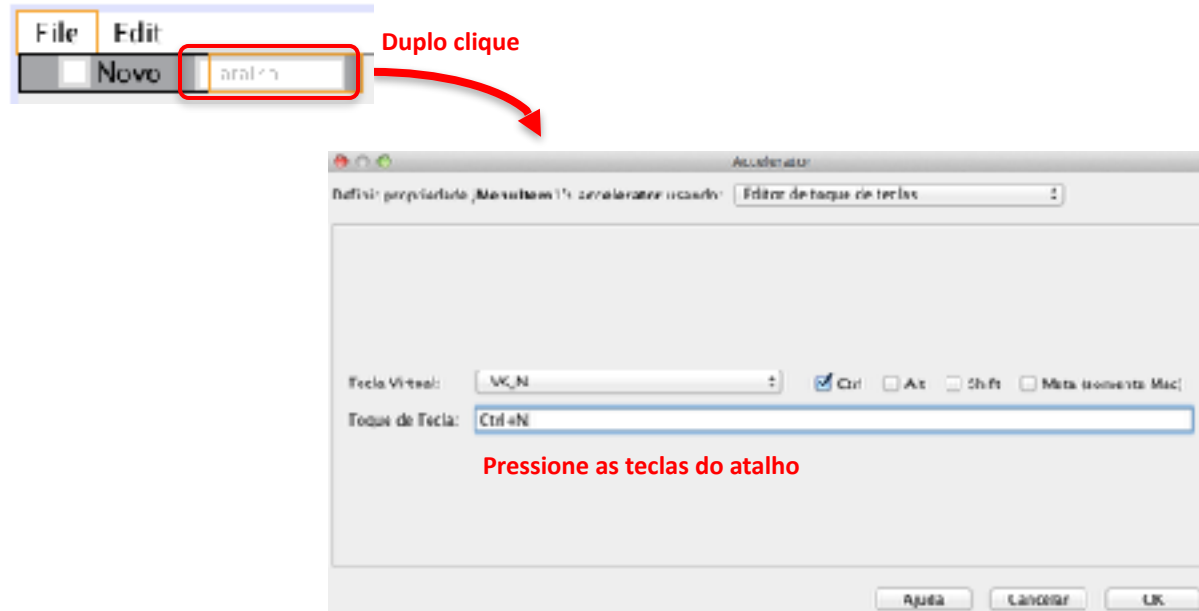
# Menu de Navegação Vertical



Componentes de um menu exibidos no modo gráfico

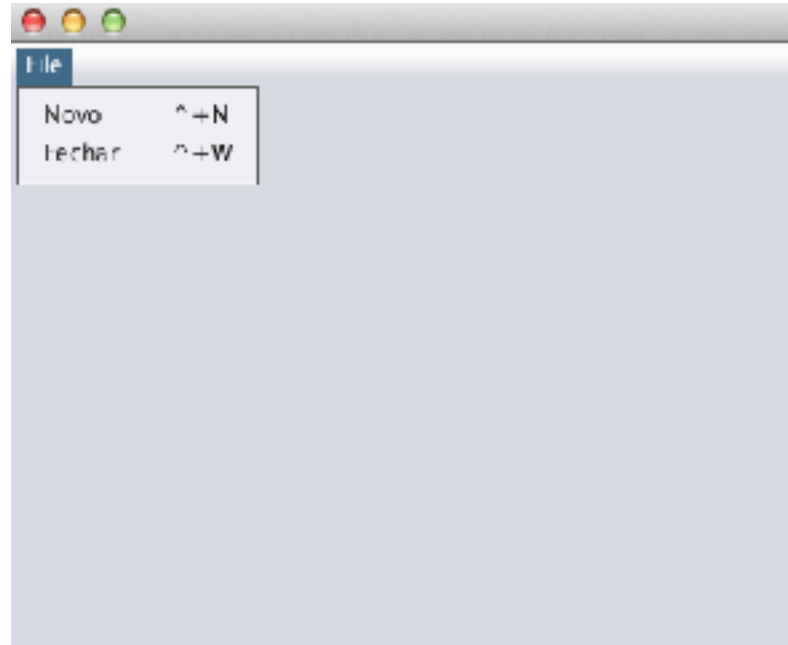
# Menu de Navegação Vertical

- Atalhos do teclado podem ser atribuídos aos itens do menu.



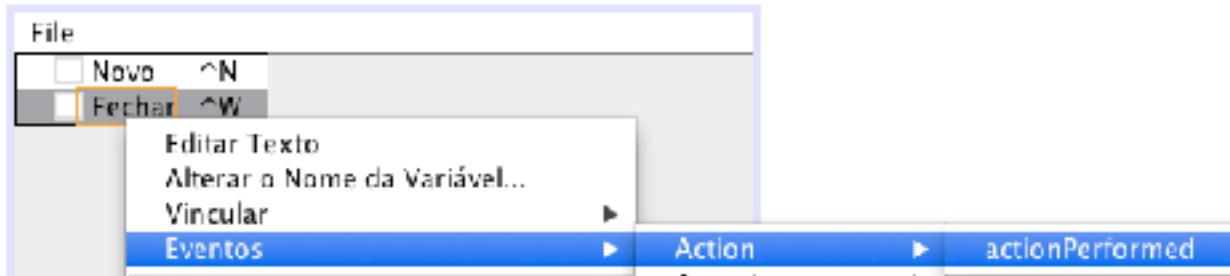


# Menu de Navegação Vertical



# Eventos no JMenuItem

- Podemos manipular eventos de um *JMenuItem* utilizando o método *ActionPerformed*.



# Eventos no JMenuItem

- Vamos adicionar o seguinte manipulador de eventos ao item de menu “Fechar”.

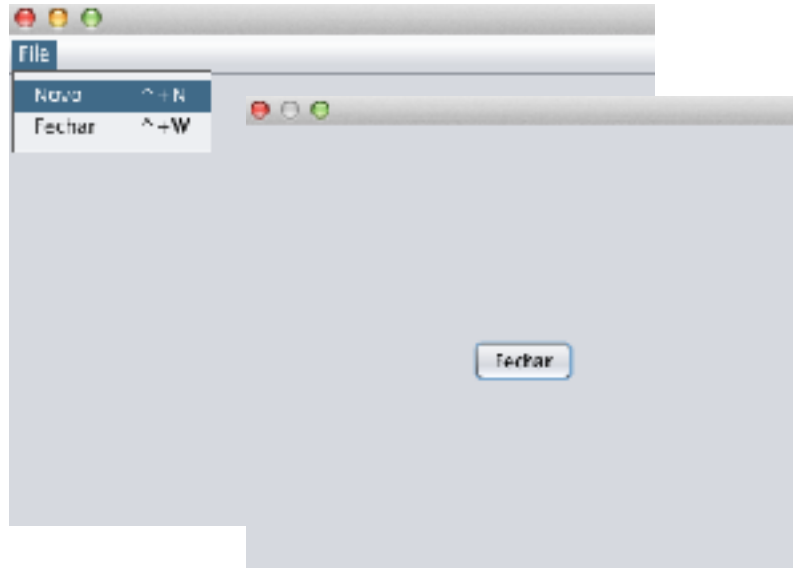
```
private void menuItemSairActionPerformed(java.awt.event.ActionEvent evt) {  
    this.dispatchEvent(new WindowEvent(this, WindowEvent.WINDOW_CLOSING));  
}
```



O nome da variável que representa o item de menu “Fechar” deve ser alterado de *jMenuItem2* para *menuItemFechar*

# Eventos no JMenuItem

- Eventos para abertura de novas janelas também podem ser adicionados a um item de menu.



# Eventos no JMenuItem

- Eventos para abertura de novas janelas também podem ser adicionados a um item de menu.

- Para tal:

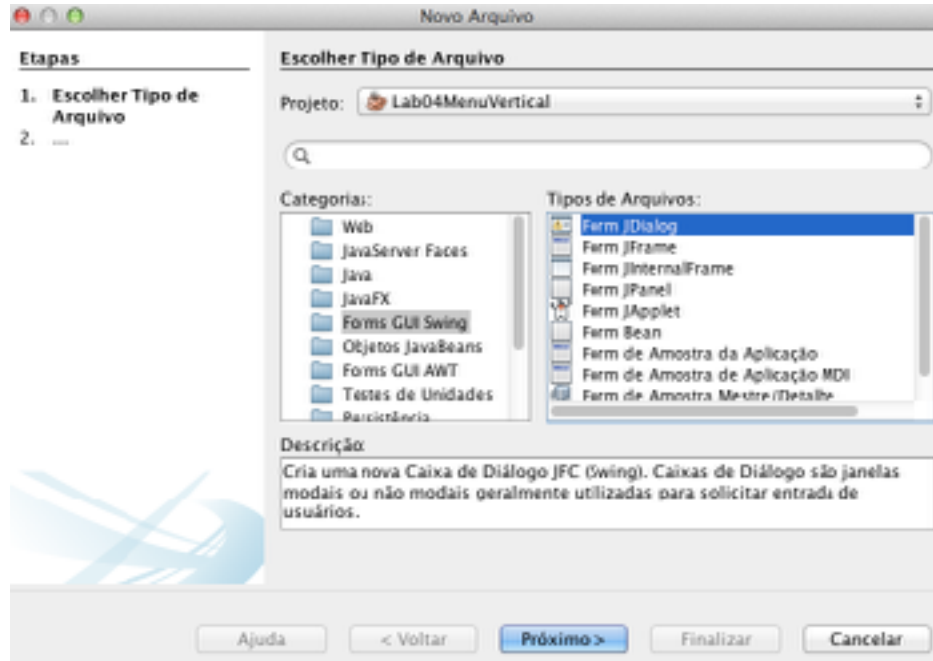
- 1) Temos que instanciar a nova janela a ser aberta no construtor da classe principal.

```
novFrame = new JDialogUI(this, true);
```

- 2) Alterar o status da visibilidade da janela para *true* no método que irá manipular o evento (*ActionPerformed*).

```
novFrame.setVisible(true);
```

# Eventos no JMenuitem



**Adicione um novo Form jDialog ao projeto**

# Eventos no JMenuItem

New Form JDialog

**Etapas**

1. Escolher Tipo de Arquivo
2. Nome e Localização

**Nome e Localização**

Nome da Classe: NovaJanelajDialog

Projeto: Lab04MenuVertical

Localização: Pacotes de Código-fonte

Pacote: br.edu.unitri

Arquivo Criado: jects/Lab04Menu/src/br/edu/unitri/NovaJanelajDialog.java

Ajuda < Voltar Próximo > Finalizar Cancelar

**Defina o nome da classe como NovaJanelajDialog**

# Eventos no JMenuItem



**Adicione um botão e altere o nome da variável para *btnFechar***



# Eventos no JMenuitem

- Adicione um manipulador de eventos do tipo *ActionPerformed* e implemente o seguinte código.

```
private void btnFecharActionPerformed(java.awt.event.ActionEvent evt) {  
    setVisible(false);  
}
```

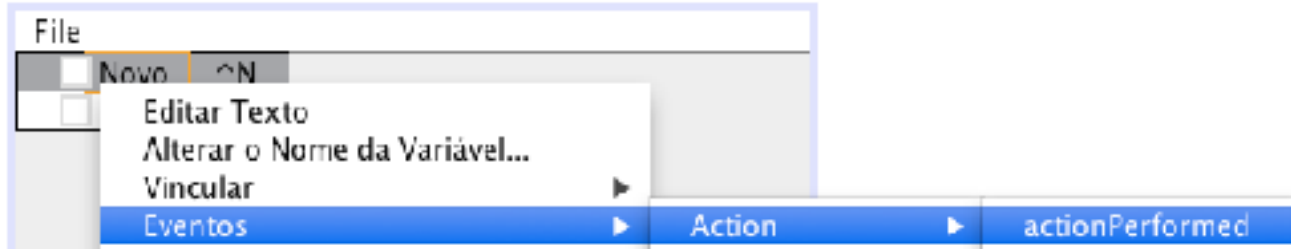
# Eventos no JMenuItem

- Uma vez criada a janela a ser aberta, temos que implementar o instanciamento da mesma e tratar o evento a ser acionado pelo item de menu.

```
public class MenuUI extends javax.swing.JFrame {  
    → private NovaJanelaJDialog dialog;  
  
    public MenuUI() {  
        initComponents();  
        → dialog = new NovaJanelaJDialog(this, true);  
    }  
}
```

**Adicione a seguinte implementação na classe principal do projeto**

# Eventos no JMenuItem

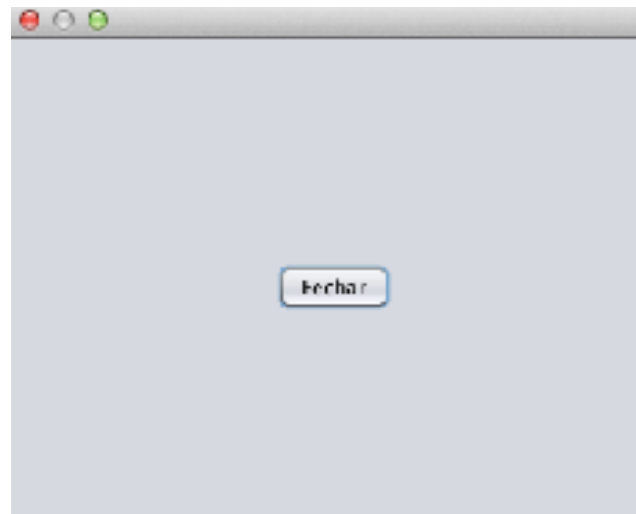
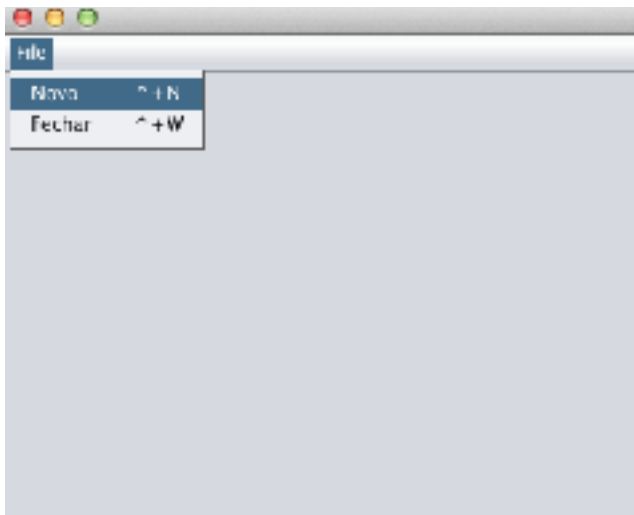


**Adicione um manipulador *ActionPerformed* no item de menu “Novo”**

```
private void menuItemNovoActionPerformed(java.awt.event.ActionEvent evt) {  
    dialog.setVisible(true);  
}
```

**Adicione a seguinte implementação no corpo do método criado**

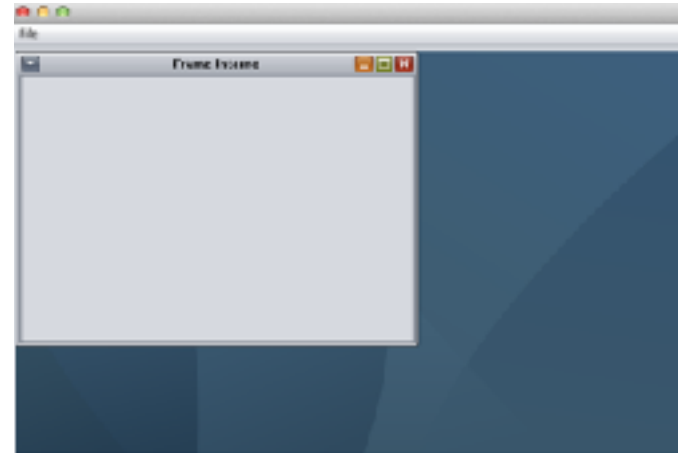
# Eventos no JMenuItem



# Desafio

- Como vimos anteriormente, podemos abrir novos frames dentro de um JFrame utilizando o conceito de frames internos (JInternalFrame).
  - Novos frames internos também podem ser abertos através dos itens de menu (JMenuItem).
  - Lembre-se que, um frame interno deve ser adicionado a um *desktop pane*.
- Faça a implementação da abertura de um frame interno a partir de um evento no item de menu.
  - Para lhe auxiliar no desafio proposto, será dado um pseudocódigo desta implementação.

# Desafio



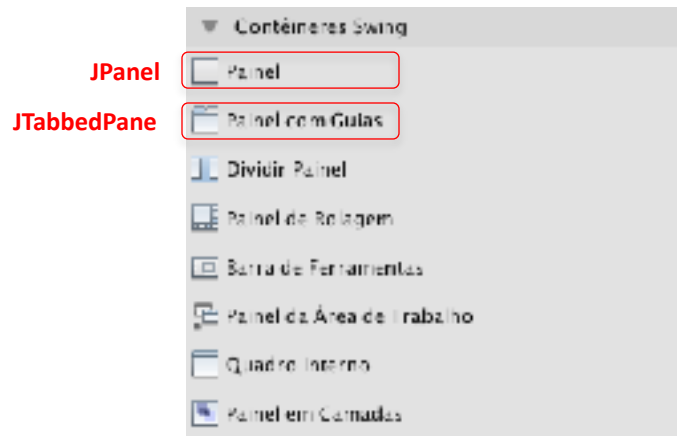
# Desafio

## Pseudocódigo

1. Defina um novo *desktop pane*.
2. Instancie um novo frame interno.
3. Altere a visibilidade do novo frame para *true*.
4. Adicione o frame interno instanciado ao *desktop pane* definido.
5. Altere a seleção (foco) do frame interno para *true*.

# Navegação através de Abas

- A classe que implementa a navegação através de abas (guias) é a *JTabbedPane*.



- Um *JTabbedPane* deve ser utilizado em conjunto com um *JPanel*.

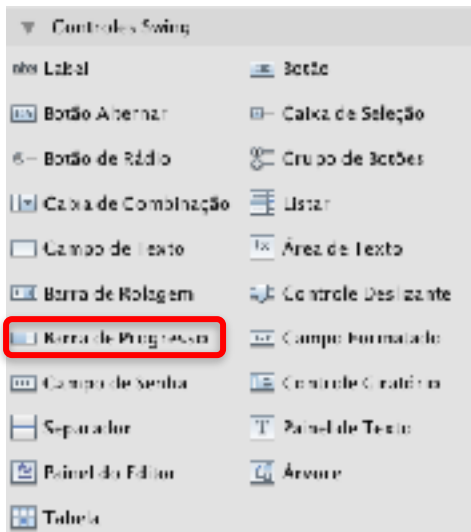


# Navegação através de Abas



# Barra de Progresso

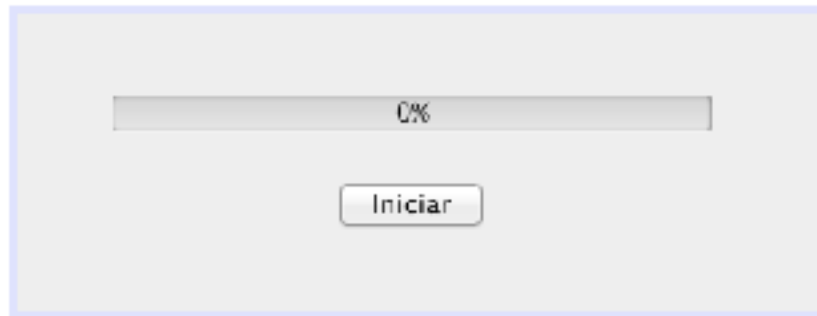
- A classe *JProgressBar* implementa uma barra de progresso.
  - Utilizamos uma barra de progresso quando necessitamos exibir o status de uma determinada tarefa.



# Barra de Progresso

- Para visualizarmos o funcionamento da barra de progresso criaremos uma Thread em Java.
  - O conceito de Thread está intimamente ligado ao conceito de um processo.
  - Ela possui um início, sequência de execução e um fim e em qualquer momento uma *thread* possui um único ponto de execução.
  - Contudo, uma thread não é um programa, ela não pode ser executada sozinha e inserida no contexto de uma aplicação, onde essa aplicação possuirá vários pontos de execuções distintos.

# Barra de Progresso



**Adicione uma barra de progresso e um botão ao JFrame**



O nome da variável que representa a barra de progresso deverá ser alterada para *“barra”*.

# Barra de Progresso

```
public static void main(String args[]) {  
    /* Set the Nimbus look and feel */  
    | Look and feel setting code (optional) |  
    //</editor-fold>  
  
    /* Create and display the form */  
    java.awt.EventQueue.invokeLater(new Runnable() {  
        public void run() {  
            new BarraProgressoUI().setVisible(true);  
        }  
    });  
}
```



```
// Variables declaration - do not modify  
private javax.swing.JProgressBar barra;  
private javax.swing.JButton btnIniciar;  
// End of variables declaration
```

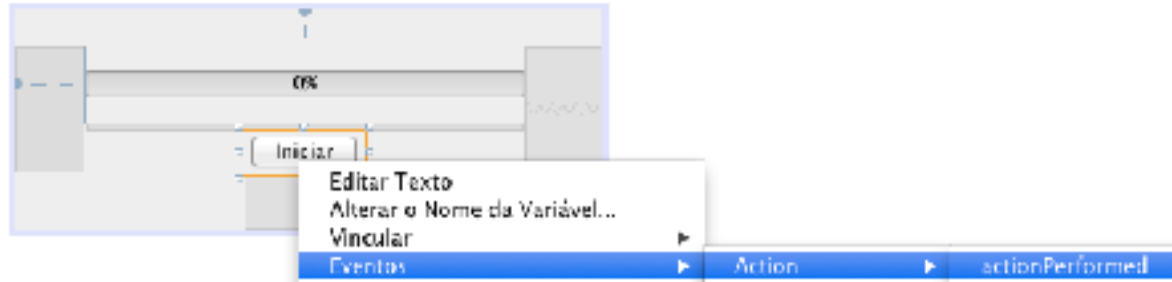
**Iremos adicionar no trecho de código destacado acima  
uma classe aninhada do tipo interna**

# Barra de Progresso

```
public class Temporizador implements Runnable {  
  
    @Override  
    public void run() {  
        for(int i=1; i <= 100; i++) {  
  
            barra.setValue(i);  
            barra.repaint();  
            try {  
                Thread.sleep(50);  
            } catch (InterruptedException ex) {  
                System.out.println(ex.toString());  
            }  
        }  
    }  
}
```

**Adicione a implementação acima no trecho de código destacado no slide anterior**

# Barra de Progresso

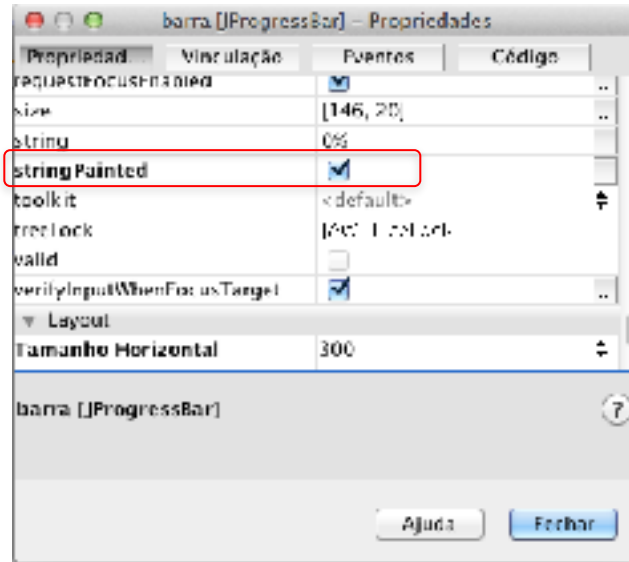


**Adicione um manipulador *ActionPerformed* no botão “Iniciar”**

```
private void btnIniciarActionPerformed(java.awt.event.ActionEvent evt) {  
    new Thread {new Temporizador()}.start();  
}
```

**Adicione a seguinte implementação no corpo do método criado**

# Barra de Progresso



**Para apresentar o contador do temporizador, selecione a propriedade *stringPainted* do progress bar**



# **Noções sobre o MVC**

# Histórico do MVC

- Padrão criado no final da década de 1970 por *Trygve Reenskaug*.
- Bastante difundido com o *Smalltalk* durante a década de 1980.
  - A documentação original da criação do MVC é disponibilizada pelo autor em seu site:
    - <http://folk.uio.no/trygver>

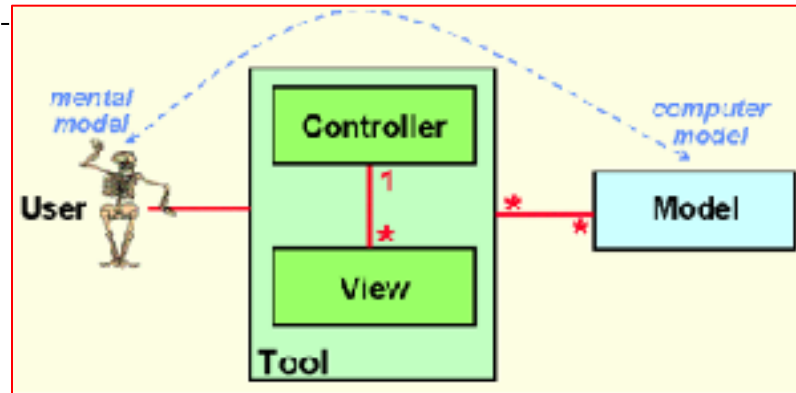
# Padrão MVC

## Model, View, Controller

- Tem como objetivo primordial a organização da aplicação em três camadas visando aumentar a flexibilidade e o reuso.

# Origens do Padrão MVC

“The essential purpose of MVC is to bridge the gap between the human user's mental model and the digital model that exists in the computer. The ideal MVC solution supports the user illusion of seeing and manipulating the domain information directly. The structure is useful if **the user needs to see the same model element simultaneously in different contexts** and/or from different viewpoints.”



Texto e figura extraídos do site <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>

# Arquitetura do MVC

- MVC possui três tipos de objetos:
  - **Model:** objeto do domínio da aplicação.
  - **View:** objeto da apresentação.
  - **Controller:** objetos que definem como a interface do usuário reage a uma entrada do usuário.
- O MVC desacopla estes objetos para aumentar flexibilidade e reuso.



# Arquitetura do MVC

## MODELO

- Os objetos desta camada contêm os dados do negócio e ditam como acessar e modificar os dados de maneira consistente.
- Encapsula os dados e os comportamentos do negócio e persiste os mesmos sem se preocupar em como serão mostrados.

# Arquitetura do MVC

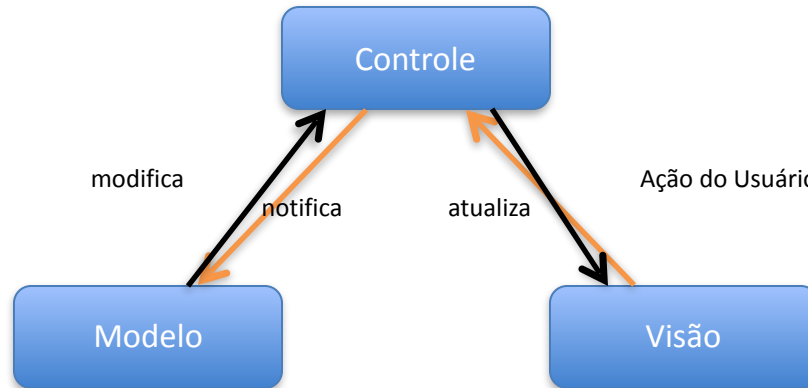
## VISÃO

- Deve garantir que sua aparência reflita o estado do modelo.
- Toda vez que o modelo muda, notifica as visões que dependem dele. Assim, cada visão pode atualizar-se.
  - Isto permite que múltiplas visões sejam conectadas ao modelo para prover diferentes apresentações.
  - Novas visões podem ser criadas para o modelo sem a necessidade de reescrevê-lo.

# Arquitetura do MVC

## CONTROLE

- Comanda o fluxo de apresentação das informações fazendo a intermediação entre as camadas de visão e de modelo.





# Vantagens do MVC

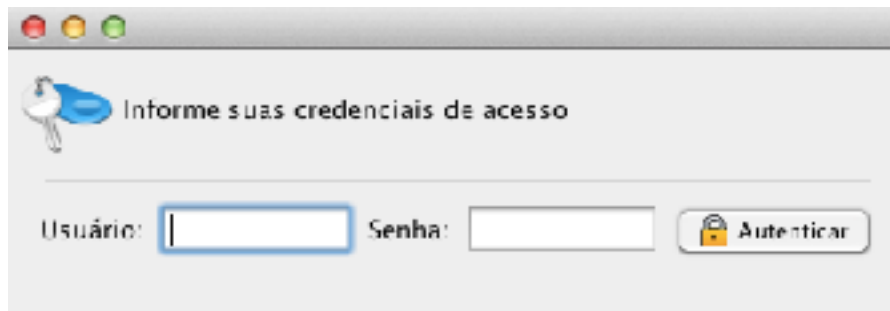
- Facilita a manutenção do software.
- Simplifica a inclusão de um novo elemento de visão.
- Melhora a escalabilidade.
- Possibilita desenvolvimento das camadas em paralelo, se estas forem bem definidas.

# Desvantagens do MVC

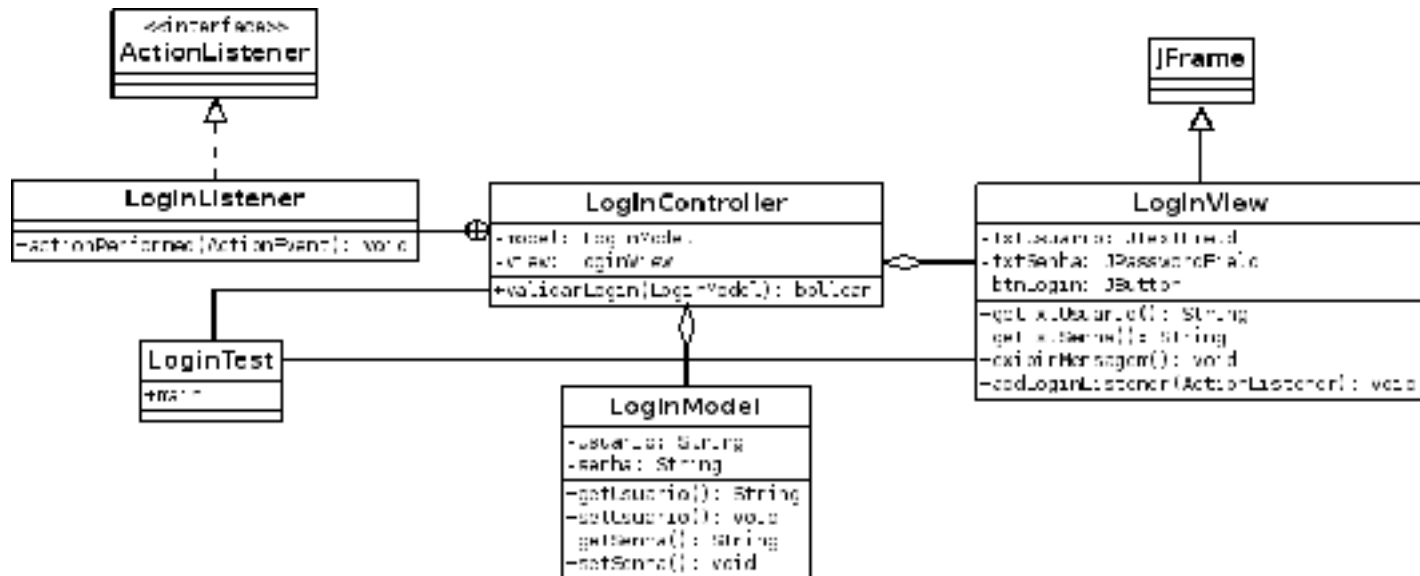
- Requer análise mais aprofundada da arquitetura da aplicação (mais tempo).
- Não aconselhável para aplicações pequenas (custo x benefício não compensa).

# **Aplicação Swing com MVC**

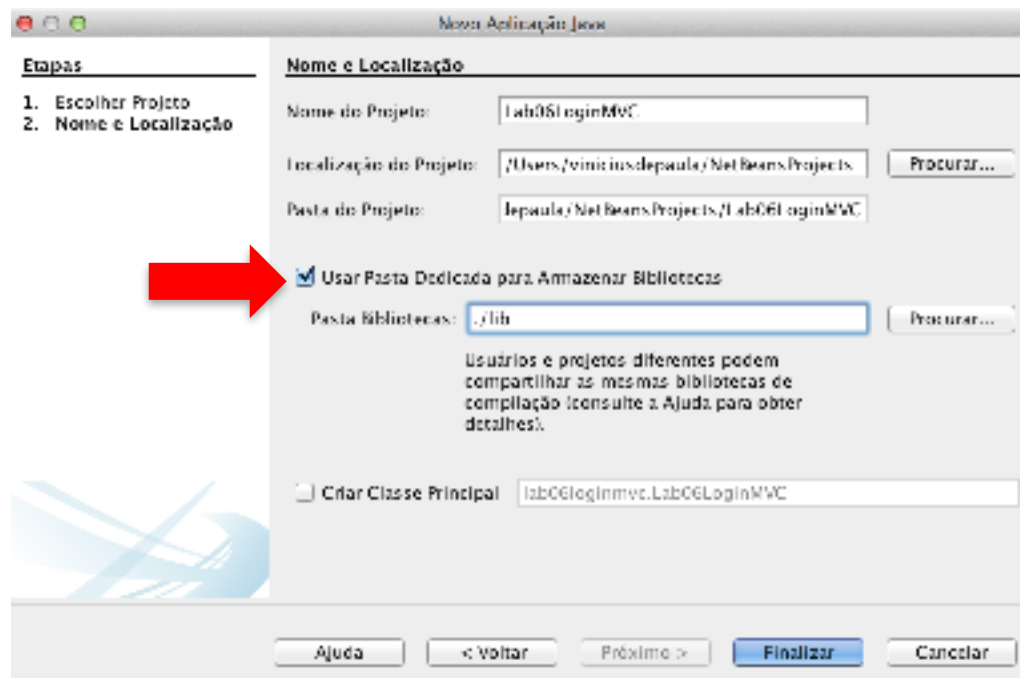
# Visão da Interface da Aplicação



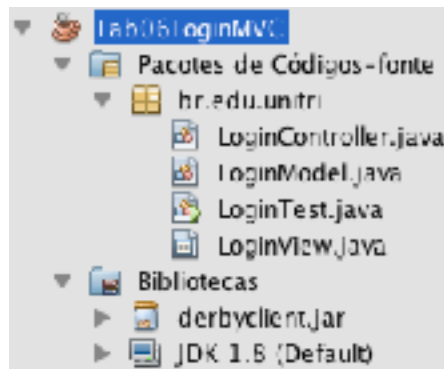
# Diagrama de Classes



# Criação do Projeto



# Estrutura do Projeto



# **Implementação do Modelo**



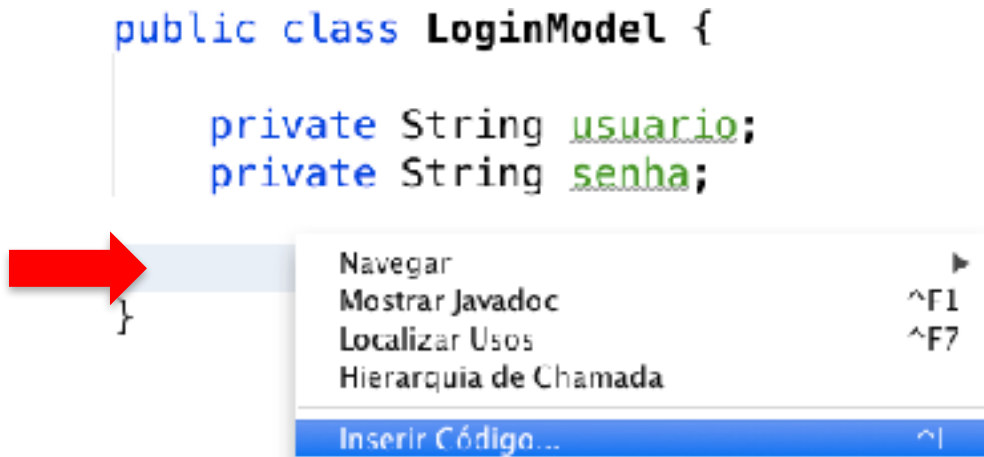
# Implementação do Modelo

Crie uma classe Java conforme o modelo apresentado

```
public class LoginModel {  
    private String usuario;  
    private String senha;  
  
}
```

Iremos adicionar um construtor e os métodos Getters e Setters para os dois atributos

# Implementação do Modelo



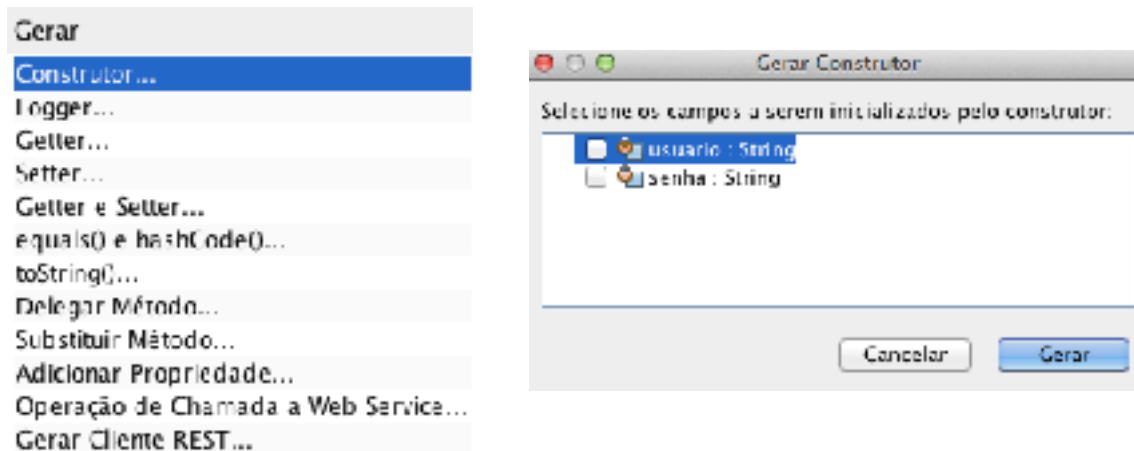
**Clique com o botão direito na linha destacada acima e em seguida clique em “Inserir Código”**

# Implementação do Modelo

## Construtor

- Iremos definir dois tipos de construtores para a classe que tratará do modelo da aplicação (*LoginModel*).
  - Um construtor **default**, que não receberá nenhum argumento e o corpo dele será vazio.
  - E outro que receberá dois argumentos, e será responsável por **inicializar** as informações do modelo (usuário e senha).

# Implementação do Modelo

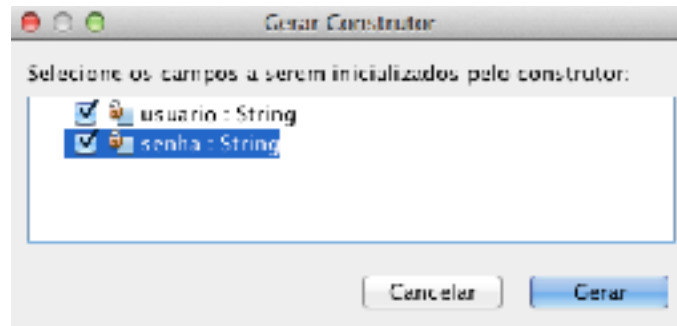
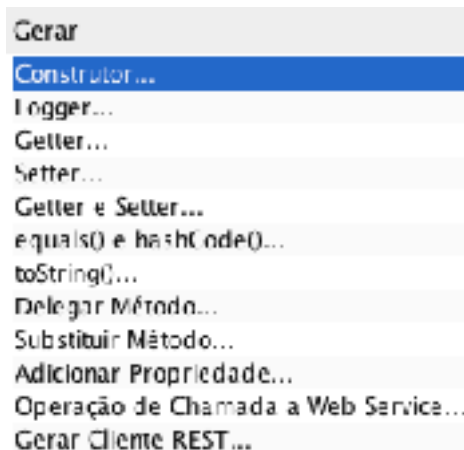


Clique em “Construtor” e em seguida clique no botão “Gerar”

```
public LoginModel() {  
}
```

Código do Construtor default gerado automaticamente pelo NetBeans

# Implementação do Modelo

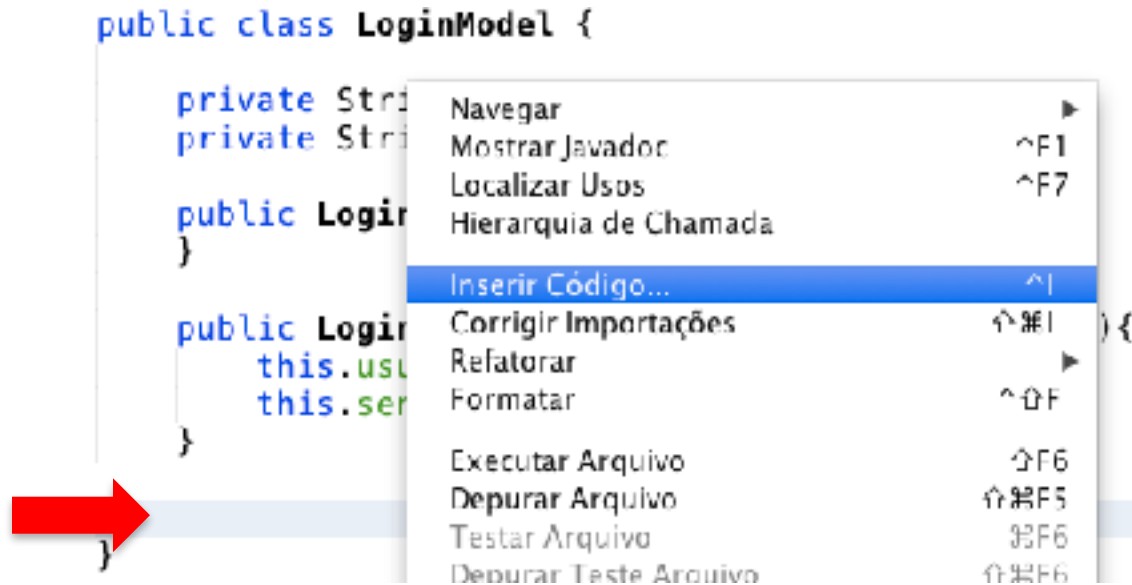


Clique em "Construtor" e selecione os dois atributos

```
public LoginModel(String usuario, String senha) {  
    this.usuario = usuario;  
    this.senha = senha;  
}
```

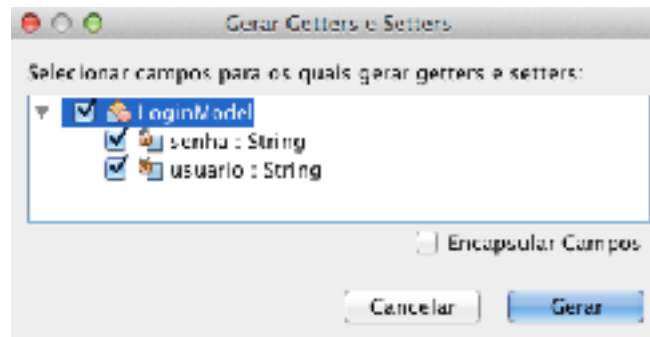
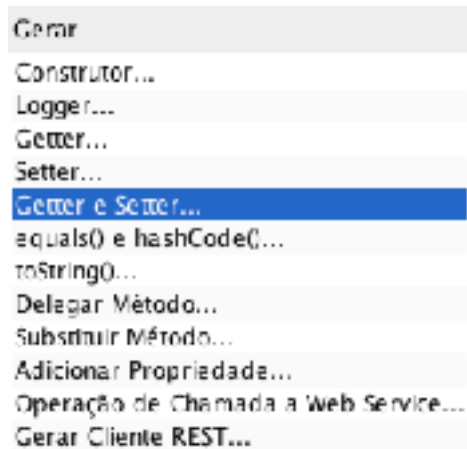
Código do Construtor gerado automaticamente pelo NetBeans

# Implementação do Modelo



**Clique com o botão direito na linha abaixo do código gerado para o Construtor em seguida clique em “Inserir Código”**

# Implementação do Modelo



**Clique em “Getter e Setter...” e selecione os dois atributos**

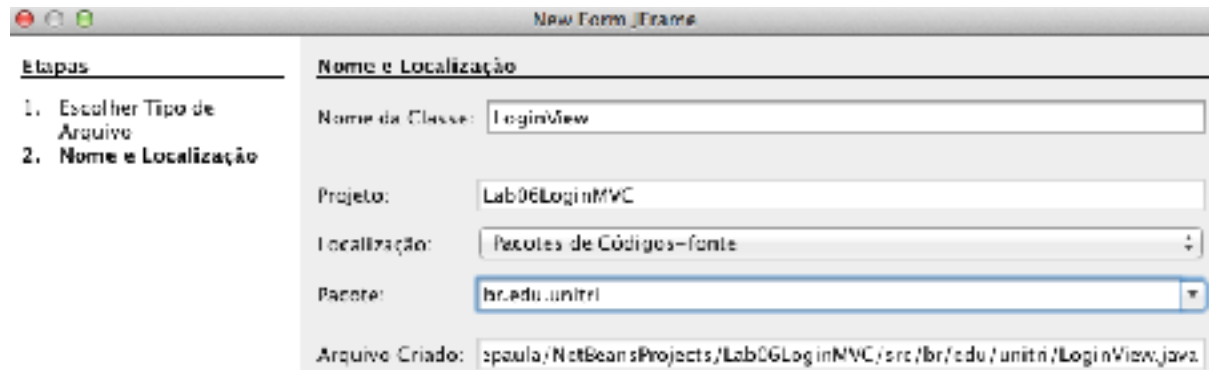
# **Implementação da Visão**



# Implementação da Visão



Adicione um JFrame ao projeto



Nomenclatura da classe e localização do pacote

# Implementação da Visão



```
public static void main(String args[]) {  
    /* Set the Nimbus look and feel */  
    Look and feel setting code (optional)  
  
    /* Create and display the form */  
    java.awt.EventQueue.invokeLater(new Runnable() {  
        public void run() {  
            new LoginView().setVisible(true);  
        }  
    });  
}
```

Remova o método `main()` da classe `LoginView`

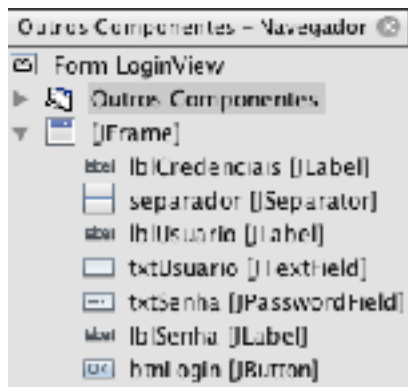


A classe `LoginTest` será responsável por inicializar a aplicação. Dessa forma, criaremos posteriormente o método `main()` dentro desta classe.

# Implementação da Visão



Visão da interface gráfica da aplicação a ser construída



Nomenclatura dos componentes

# Implementação da Visão

```
@SuppressWarnings("unchecked")  
Generated Code
```



```
public String getTxtUsuario() {  
    return txtUsuario.getText();  
}
```



```
public String getTxtSenha() {  
    return new String(txtSenha.getPassword());  
}
```



```
public void exibirMensagem(String msg) {  
    JOptionPane.showMessageDialog(this, msg);  
}
```



```
void addLoginListener(ActionListener listener) {  
    btnLogin.addActionListener(listener);  
}
```

**Implemente os seguintes métodos dentro da classe *LoginView***

# **Implementação do Controlador**

# Implementação do Controlador

- A classe *LoginController* que criaremos a seguir coordenará as interações entre a classe *LoginModel* e a classe *LoginView*.

# Implementação do Controlador

Crie uma nova classe Java chamada *LoginController* e adicione um Construtor conforme a implementação abaixo

```
public class LoginController {  
  
    private LoginView view;  
    private LoginModel model;  
  
    public LoginController(LoginView view, LoginModel model) {  
  
        this.view = view;  
        this.model = model;  
        this.view.addLoginListener(new LoginListener());  
    }  
}
```

# Implementação do Controlador

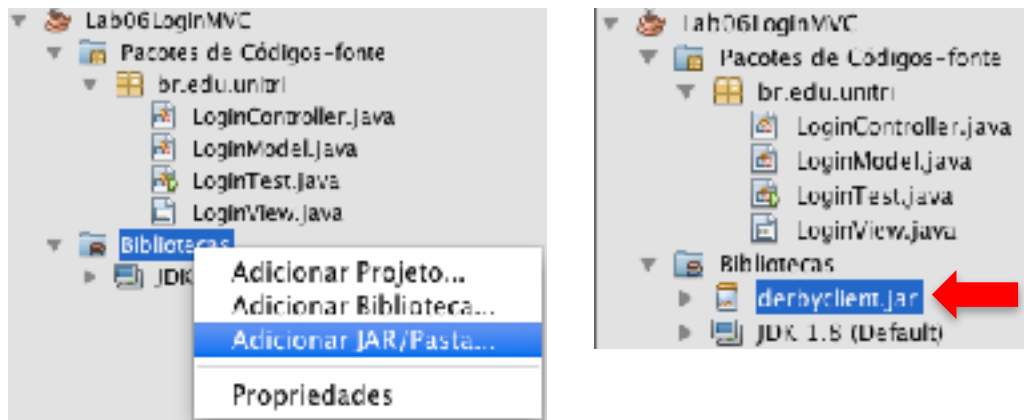
## Validação das Credenciais

- As credenciais informadas na aplicação serão validadas utilizando uma tabela do banco de dados.
- Por praticidade utilizaremos o [Apache Derby](#), SGBDR (sistema gerenciador de banco de dados relacional) que vem integrado ao NetBeans, construído em Java e com suporte a SQL.



# Implementação do Controlador

- Para que possamos interagir com o Apache Derby através do código Java, temos que adicionar o driver JDBC correspondente ao banco de dados no projeto.



# Implementação do Controlador


Implemente o método *validarLogin()* logo abaixo do Construtor da classe *LoginController*

```
public boolean validarLogin(LoginModel model) throws Exception {  
  
    String dbUrl = "jdbc:derby://localhost:1527/db_test;user=user_test;password=123mudar";  
    String dbClass = "org.apache.derby.jdbc.ClientDriver";  
    String query = "SELECT * FROM tbl_usuario WHERE ds_login = ? AND ds_senha = ?";  
  
    try {  
        Class.forName(dbClass);  
        Connection con = DriverManager.getConnection(dbUrl);  
        PreparedStatement prepStmt = con.prepareStatement(query);  
        prepStmt.setString(1, model.getUsuario());  
        prepStmt.setString(2, model.getSenha());  
        ResultSet rs = prepStmt.executeQuery();  
  
        if (rs.next()) {  
            return true;  
        }  
        con.close();  
    } catch (Exception e) {  
        throw e;  
    }  
    return false;  
}
```

# Implementação do Controlador

Implemente a seguinte classe aninhada à classe *LoginController*

```
class LoginListener implements ActionListener {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        try {  
            model = new LoginModel(view.getTxtUsuario(), view.getTxtSenha());  
            if(validarLogin(model)){  
                view.exibirMensagem("Login realizado com sucesso!");  
            }else{  
                view.exibirMensagem("Usuário e/ou senha inválidos!");  
            }  
        } catch (Exception ex) {  
            view.exibirMensagem(ex.toString());  
        }  
    }  
}
```

 Fim da classe *LoginController*

# **Implementação da Classe Principal**

# Implementação da Classe Principal

Crie uma nova classe Java chamada *LoginTest* e adicione um método *main()*

```
public class LoginTest {  
    public static void main(String[] args) {  
        LoginView view = new LoginView();  
        LoginModel model = new LoginModel();  
  
        LoginController controller = new LoginController(view, model);  
  
        view.setVisible(true);  
    }  
}
```

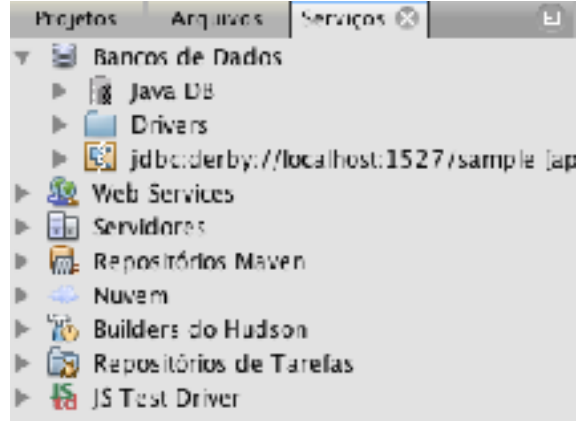
# Resumo

- ✓ Implementação do Modelo
- ✓ Implementação da Visão
- ✓ Implementação do Controlador
- ✓ Implementação da Classe Principal
- ☐ Criação do Banco de Dados
- ☐ Criação da Tabela
- ☐ Inclusão dos Registros

# **Criação do Banco de Dados**

# Criação do Banco de Dados

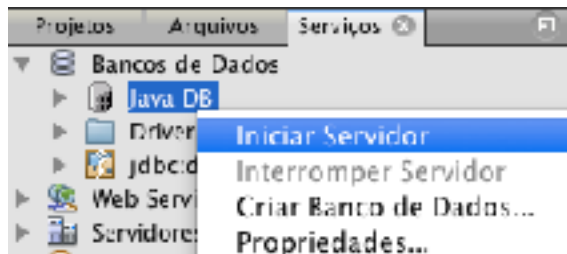
- O painel *Serviços > Bancos de Dados* apresenta a visualização das conexões existentes.





# Criação do Banco de Dados

- *Java DB* representa o Apache Derby. Para iniciar o serviço, clique com o botão direito e em seguida “Iniciar Servidor”.



- Caso a inicialização ocorra com sucesso, a seguinte mensagem deverá ser apresentada:

Apache Derby Servidor de Rede - 10.9.1.0 - (1344872) iniciado e pronto para aceitar conexões na porta 1527

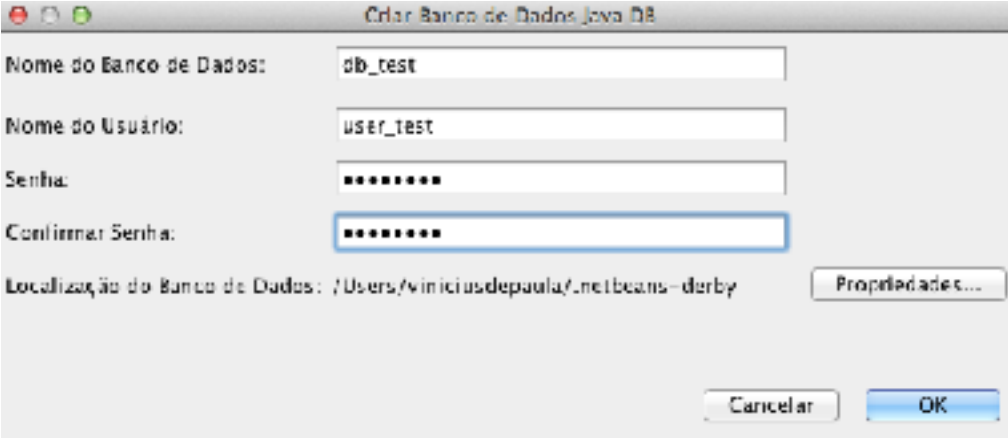
# Criação do Banco de Dados

- Para criar o banco de dados, clique com o botão direito em *Java DB* > *Criar Banco de Dados...*



# Criação do Banco de Dados

- Defina o seguintes valores para o banco de dados a ser criado:

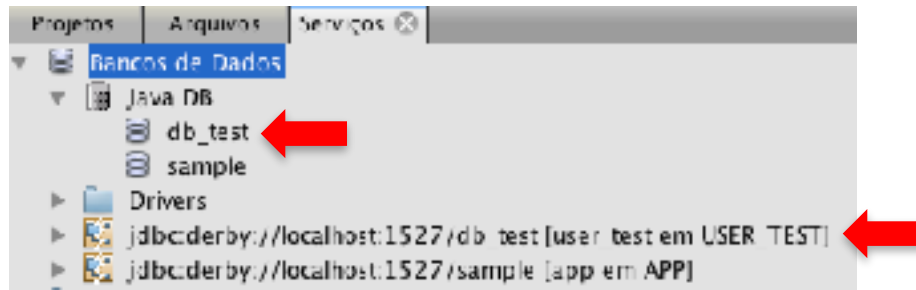


The screenshot shows a dialog box titled "Criar Banco de Dados Java DB". It contains the following fields and controls:

- Nome do Banco de Dados:** A text field containing "db\_test".
- Nome do Usuário:** A text field containing "user\_test".
- Senha:** A password field containing seven asterisks "\*\*\*\*\*".
- Confirmar Senha:** A password field containing seven asterisks "\*\*\*\*\*".
- Localização do Banco de Dados:** A text field containing the path "/Users/viniciusdepaula/.netbeans-derby".
- Propriedades...:** A button located to the right of the location field.
- Cancelar:** A button at the bottom right.
- OK:** A button at the bottom right, highlighted in blue.

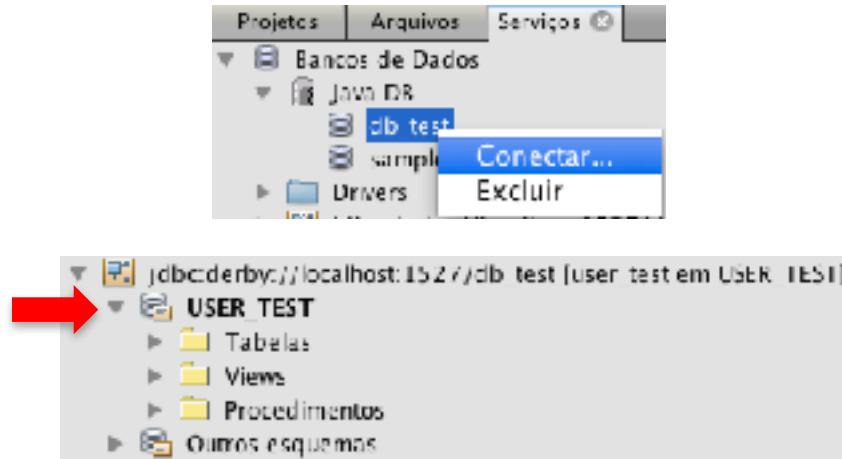
# Criação do Banco de Dados

- As seguintes entradas serão adicionadas:



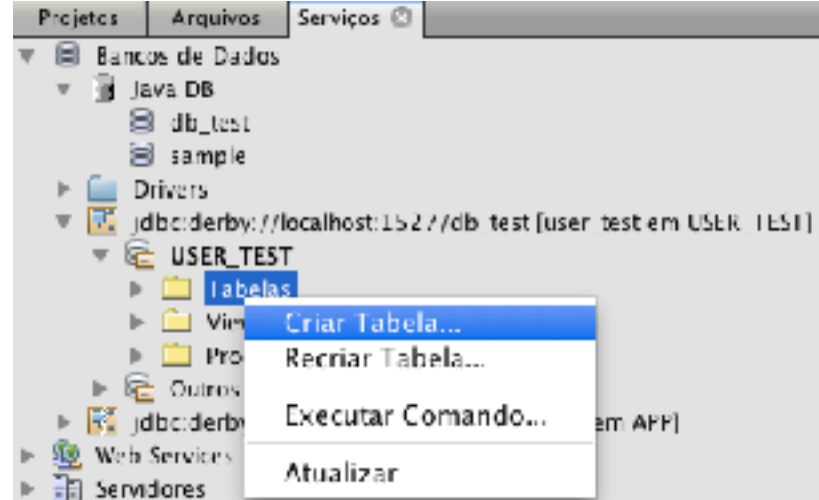
# Criação do Banco de Dados

- Antes de criarmos a tabela, é necessário se conectar ao banco de dados criado anteriormente. Clique com o botão direito em *db\_test* > *Conectar...*



# Criação da Tabela

- A criação da tabela pode ser realizada clicando com o botão direito em *Tabelas* > *Criar Tabela...*



# Criação da Tabela

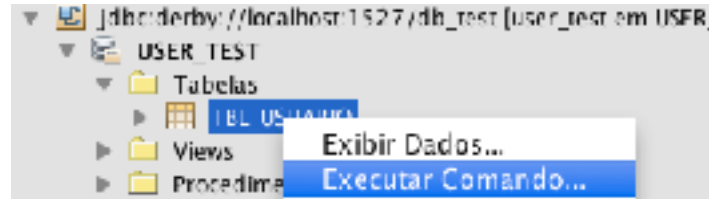
- Defina as seguintes parametrizações para a tabela.

Nome da tabela:

Chave	Índice	Única	Exclusivo	Nome da coluna	Tipo de dados	Tamanho
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	cd_usuario	INTEGER	4
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	ds_login	VARCHAR	40
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	ds_senha	VARCHAR	16

# Adicionando Registros na Tabela

- Vamos adicionar três registros na tabela criada. Para isso, clique com o botão direito em *TBL\_USUARIO* > *Executar Comando...*

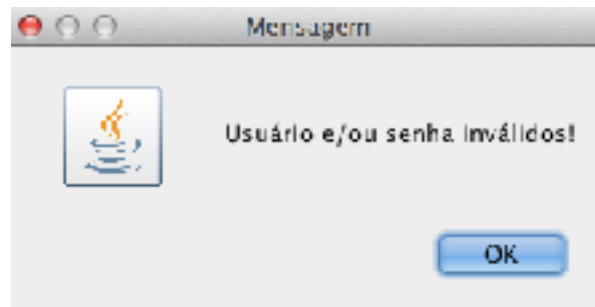
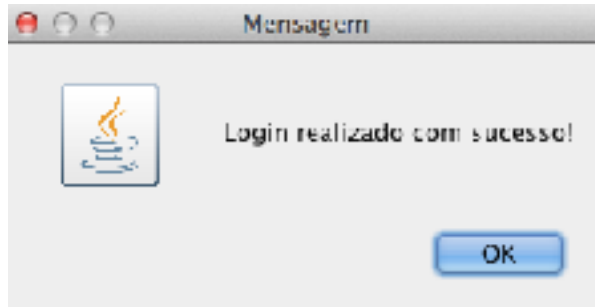
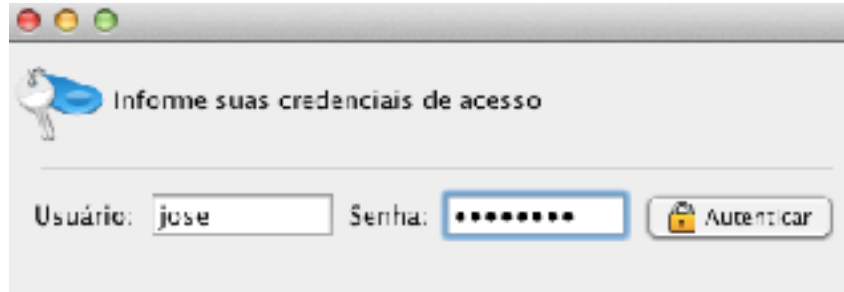


```
insert into tbl_usuario values (1, 'jose', '123mudar');  
insert into tbl_usuario values (2, 'joao', '123mudar');  
insert into tbl_usuario values (3, 'maria', '123mudar');
```

**Execute os seguintes comandos**

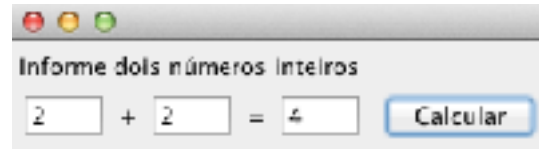


# Executando a Aplicação



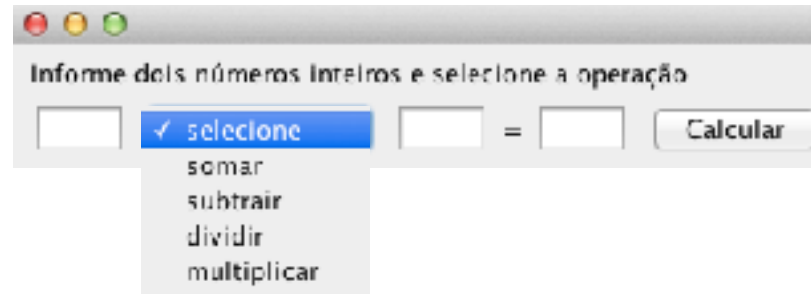
# Desafio

- Implemente uma aplicação Swing utilizando o padrão MVC para realizar a soma de dois números inteiros.



# Desafio

- Faça uma melhoria na aplicação criada anteriormente, onde agora será possível selecionar através de um *combobox* a operação a ser realizada.



Não se esqueça de apresentar uma mensagem caso a operação selecionada seja inválida e seguir a implementação no padrão MVC.

# Referências

- [1] The Definitive Guide to Java Swing; John Zukowski; 3ª ed; Apress
- [2] Programação com Java: Uma Introdução Abrangente; Herbert Schildt; Dale Skrien; McGraw-Hill
- [3] Tutorial Java: <http://docs.oracle.com/javase/tutorial>