

Desenvolvimento de Aplicações para Cliente Servidor

Aula 08 - Trabalhando Componentes

- Uso de Scene Builder e FXML: CheckBox, ComboBox, TextView
- Trabalhando a Classe FXController

Prof. Fernando Gonçalves Abadia

Trabalhando com o Scene Builder

- ▶ Para evitar a criação de formulários por código, foi criado o Scene Builder.
- ▶ Com ele, é possível criar frames em um arquivo semelhante ao XML (chamado FXML), onde possui toda a interface de sua tela e todas as propriedades necessárias.
- ▶ O código Java interpreta o layout montado no arquivo e identifica todos os componentes, através de Annotation (@FXML).



Trabalhando com o Scene Builder

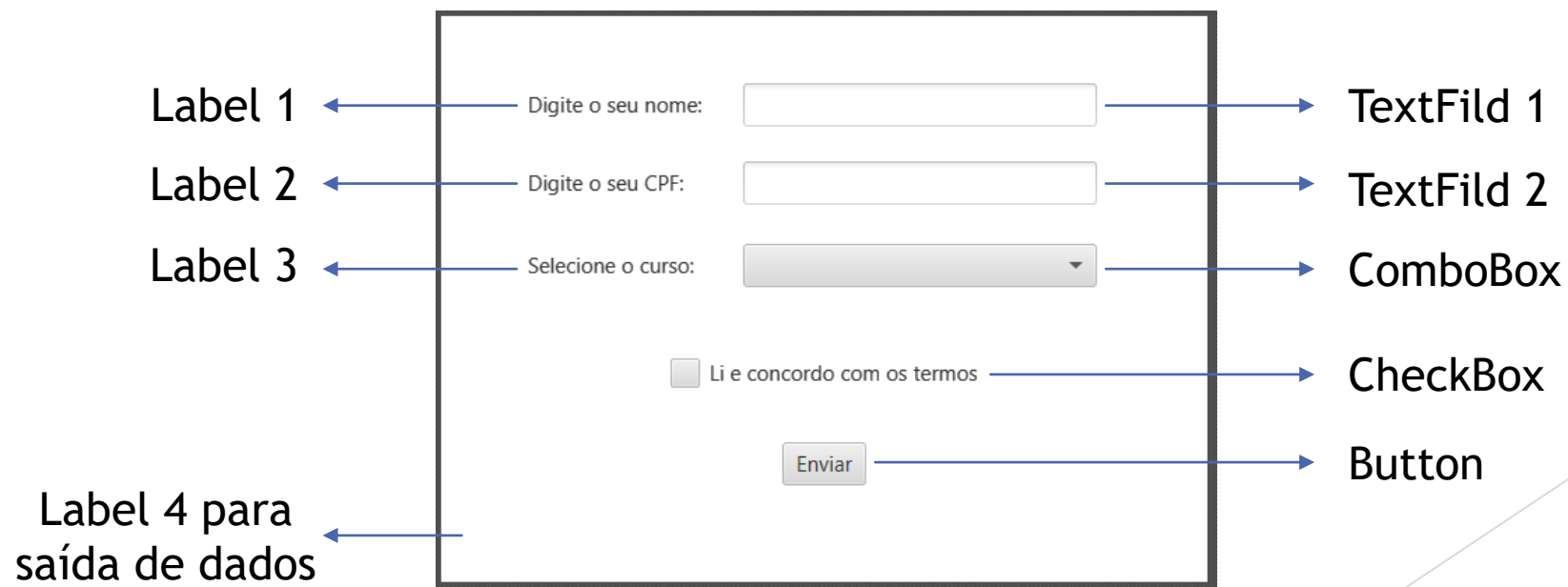
- ▶ Com o Scene Builder, o trabalho do desenvolvedor será apenas criar a lógica de implementação de cada ação e códigos ao abrir a tela, como uma troca de texto em um campo.
- ▶ Para utilizá-los, basta que arrastemos o componente para dentro da tela, posicionando na coordenada X e Y desejada, podendo ser alterada posteriormente.
- ▶ Os componentes estão divididos por tipo: Containers, Controls, Popup Controls, Menu Content, Miscellaneous, Shapes e Charts, além de que cada Node possui um ícone que demonstra bem o resultado do mesmo.

Trabalhando com o Scene Builder

- ▶ Com o Scene Builder, o trabalho do desenvolvedor será apenas criar a lógica de implementação de cada ação e códigos ao abrir a tela, como uma troca de texto em um campo.
- ▶ Para utilizá-los, basta que arrastemos o componente para dentro da tela, posicionando na coordenada X e Y desejada, podendo ser alterada posteriormente.
- ▶ Os componentes estão divididos por tipo: Containers, Controls, Popup Controls, Menu Content, Miscellaneous, Shapes e Charts, além de que cada Node possui um ícone que demonstra bem o resultado do mesmo.

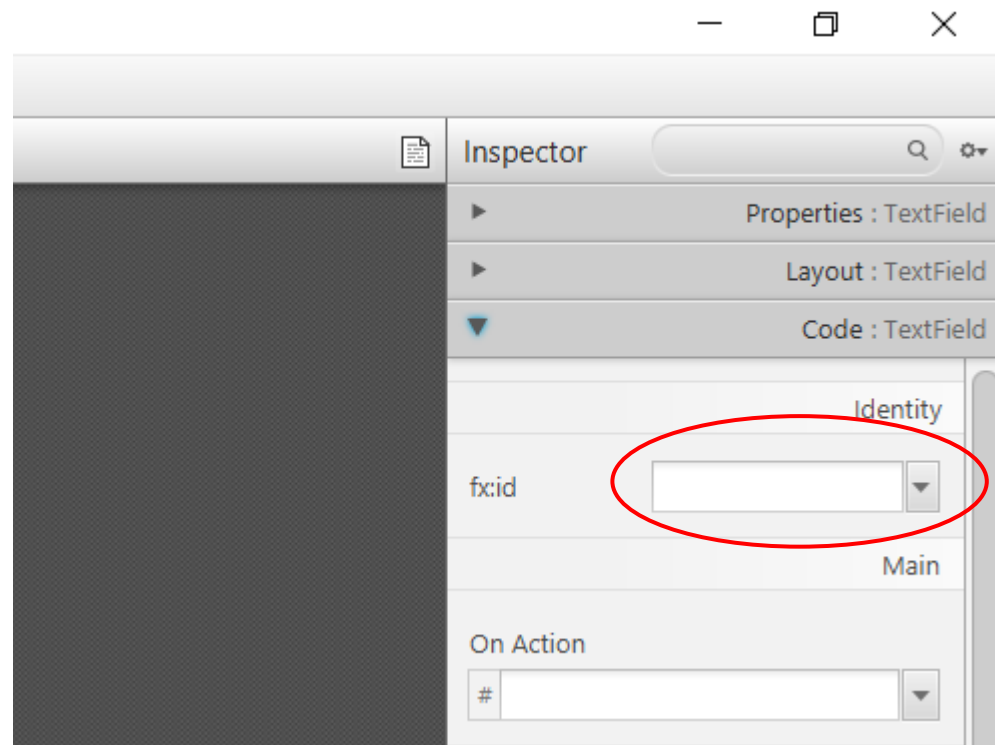
Trabalhando com o Scene Builder

- Para o nosso primeiro projeto JavaFX, trabalharemos um layout simples no Scene Builder. Precisaremos dos componentes: Label, TextField, ComboBox, CheckBox e um Botão.



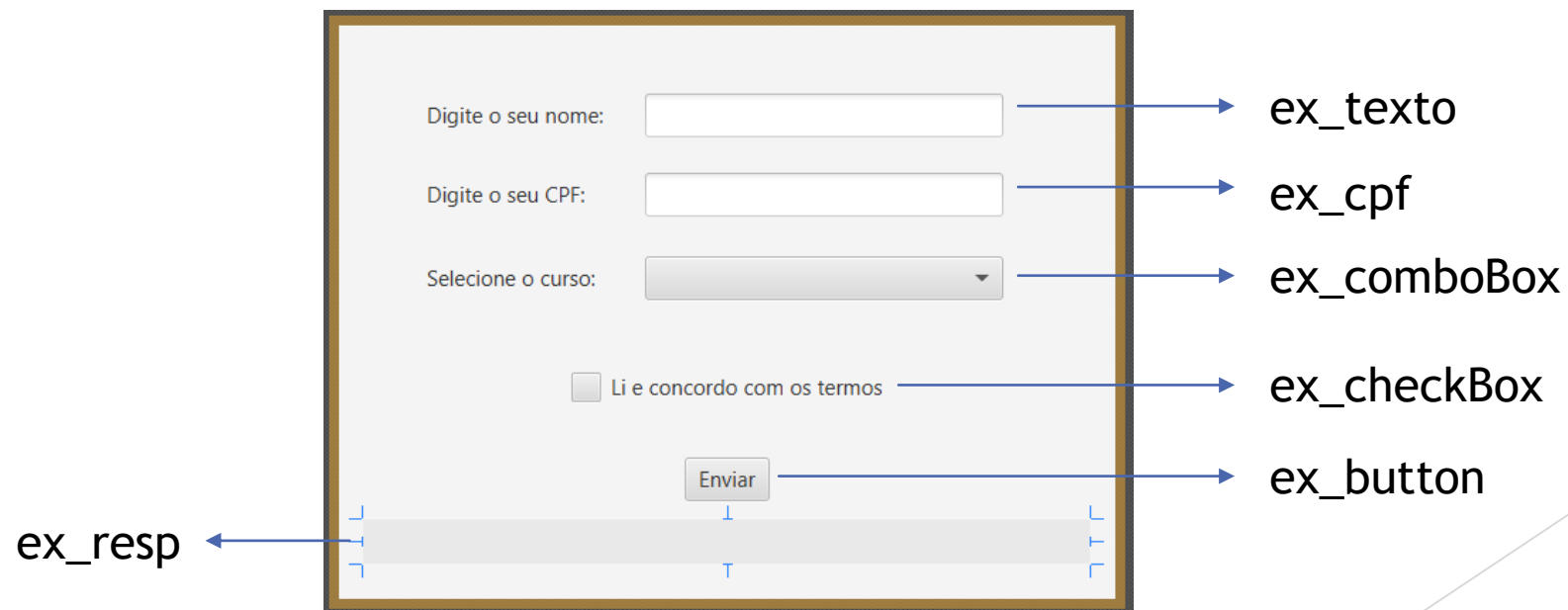
Trabalhando com o Scene Builder

- ▶ Após arrastar os componentes necessários, precisaremos alterar suas ids (fx:id) na parte Code do Scene Builder.



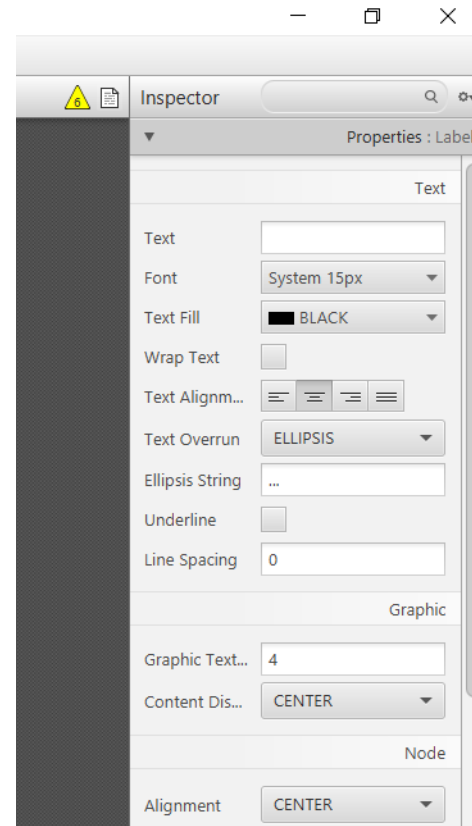
Trabalhando com o Scene Builder

- Desta forma, precisaremos alterar apenas os ids dos componentes principais, conforme mostrado abaixo:



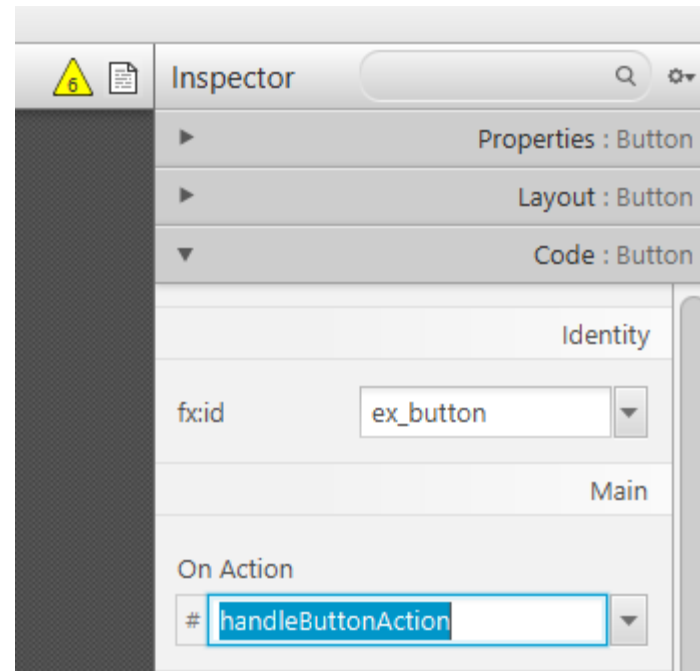
Trabalhando com o Scene Builder

- Podemos trabalhar algumas configurações do label de saída, em propriedades, para que fique centralizado na tela e não alinhado a esquerda.



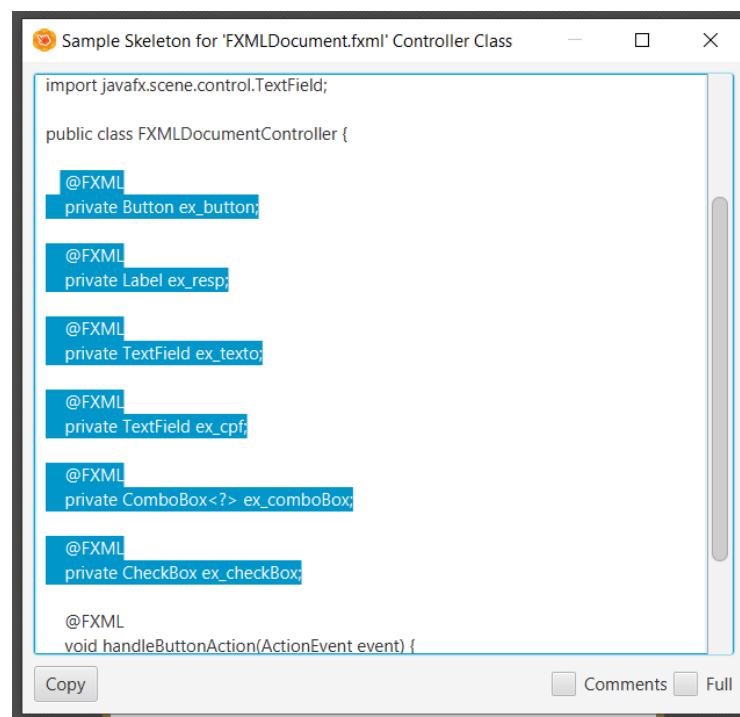
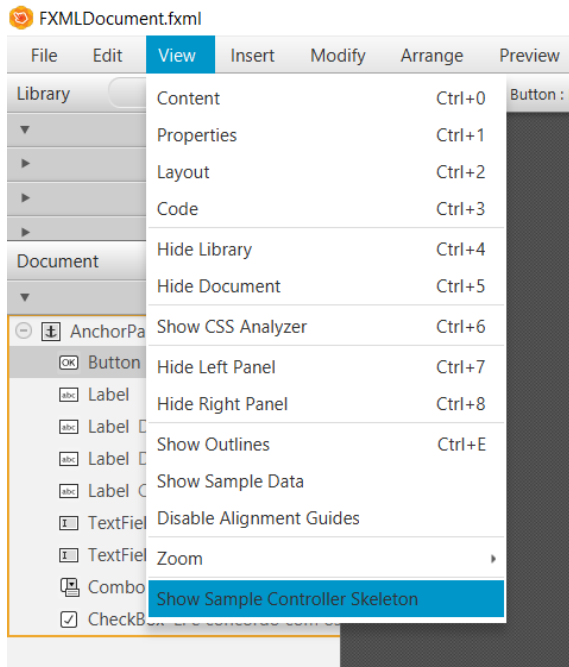
Trabalhando com o Scene Builder

- ▶ É importante ressaltar o evento do botão no On Action. A ação executada após o clique no botão pode ser configurada pelo java, e deve conter o mesmo nome no Scene Builder, que no caso será `handleButtonAction`.



Trabalhando com o Scene Builder

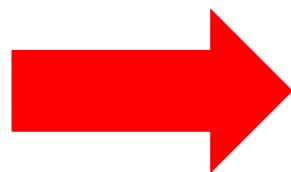
- ▶ Após estas configurações, precisaremos linkar todos os componentes criados no Scene Builder ao Controller do Java, e para isso, precisaremos ir em View, na opção Show Sample Controller Skeleton e copiar as Annotations (@FXML).



Trabalhando com o Scene Builder

- Colando as Annotations no código, precisaremos adicionar também as importações, uma a uma, clicando na lâmpada amarela e colocando a opção correta de import.

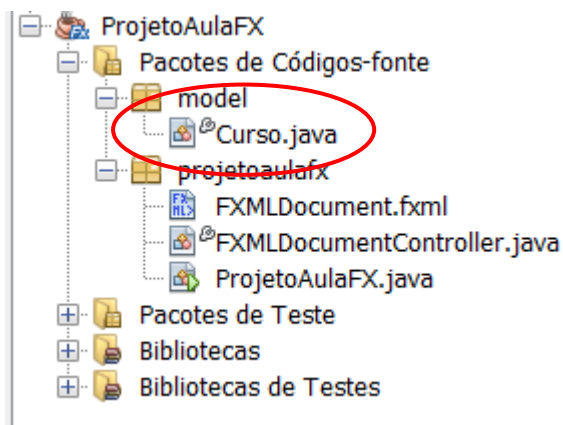
```
14 public class FXMLDocumentController implements Initializable {
15
16     @FXML
17     private Button ex_button;
18
19     @FXML
20     private Label ex_resp;
21
22     @FXML
23     private TextField ex_texto;
24
25     @FXML
26     private TextField ex_cpf;
27
28     @FXML
29     private ComboBox<?> ex_comboBox;
30
31     @FXML
32     private CheckBox ex_checkBox;
```



```
9 import javafx.event.ActionEvent;
10 import javafx.fxml.FXML;
11 import javafx.fxml.Initializable;
12 import javafx.scene.control.Button;
13 import javafx.scene.control.CheckBox;
14 import javafx.scene.control.ComboBox;
15 import javafx.scene.control.Label;
16 import javafx.scene.control.TextField;
```

Trabalhando com o Scene Builder

- Para Trabalhar o ComboBox, como os itens serão os nomes do cursos, precisaremos criar uma classe, no pacote model, para curso, precisaremos de um id e um nome para o curso.



```
1 package model;
2
3 public class Curso {
4
5     private int cod;
6     private String nome;
7 }
```

Trabalhando com o Scene Builder

- Com as opções de criação de código (alt+insert), acrescente dois construtores (vazio e completo), set, get e um toString.

```
8 public Curso(int cod, String nome) {  
9     this.cod = cod;  
10    this.nome = nome;  
11 }  
12  
13 public Curso() {  
14 }
```

```
34 @Override  
35 public String toString() {  
36     return getName();  
37 }  
38  
39 }  
40
```

```
18 public int getCod() {  
19     return cod;  
20 }  
21  
22 public void setCod(int cod) {  
23     this.cod = cod;  
24 }  
25  
26 public String getName() {  
27     return nome;  
28 }  
29  
30 public void setName(String nome) {  
31     this.nome = nome;  
32 }
```

Trabalhando com o Scene Builder

- Adicione a importação do model ao controller do Java e modifique o comboBox para que esteja de acordo com a classe curso que foi criada.

```
8  import model.Curso;
```

```
9
```

```
39
```

```
@FXML
```

```
40
```

```
private ComboBox<Curso> ex_comboBox;
```

Trabalhando com o Scene Builder

- ▶ Para trabalhar com o comboBox, precisaremos adicionar o nome de uma lista de cursos. Para isso, será necessário importar duas bibliotecas e criar uma função capaz de carregar o comboBox.

```
44     private List <Curso> cursos;
45     private ObservableList<Curso> obsCurso;

65     public void carregaCursos() {
66         cursos = new ArrayList<>();
67         cursos.add(new Curso(1, "Engenharia da Computação" ));
68         cursos.add(new Curso(2, "Ciencias da Computação"));
69         cursos.add(new Curso(3, "Análise de Sistemas"));
70
71         obsCurso = FXCollections.observableArrayList(cursos);
72
73         ex_comboBox.setItems(obsCurso);
74     }
```

Trabalhando com o Scene Builder

- ▶ Finalizando, de modo a facilitar o uso do comboBox, precisaremos criar o método pegaCurso, que simplesmente pegará o item do comboBox para uma classe.

```
76  [ ] public Curso pegaCurso() {  
77      Curso c = ex_comboBox.getSelectionModel().getSelectedItem();  
78      return c;  
79  }
```


Trabalhando com o Scene Builder

- ▶ Para que o comBox funcione, precisaremos acrescentar a chamada do método `carregaCurso()` para o método `initialize`.

```
60      @Override
61      public void initialize(URL url, ResourceBundle rb) {
62          carregaCursos();
63      }
```

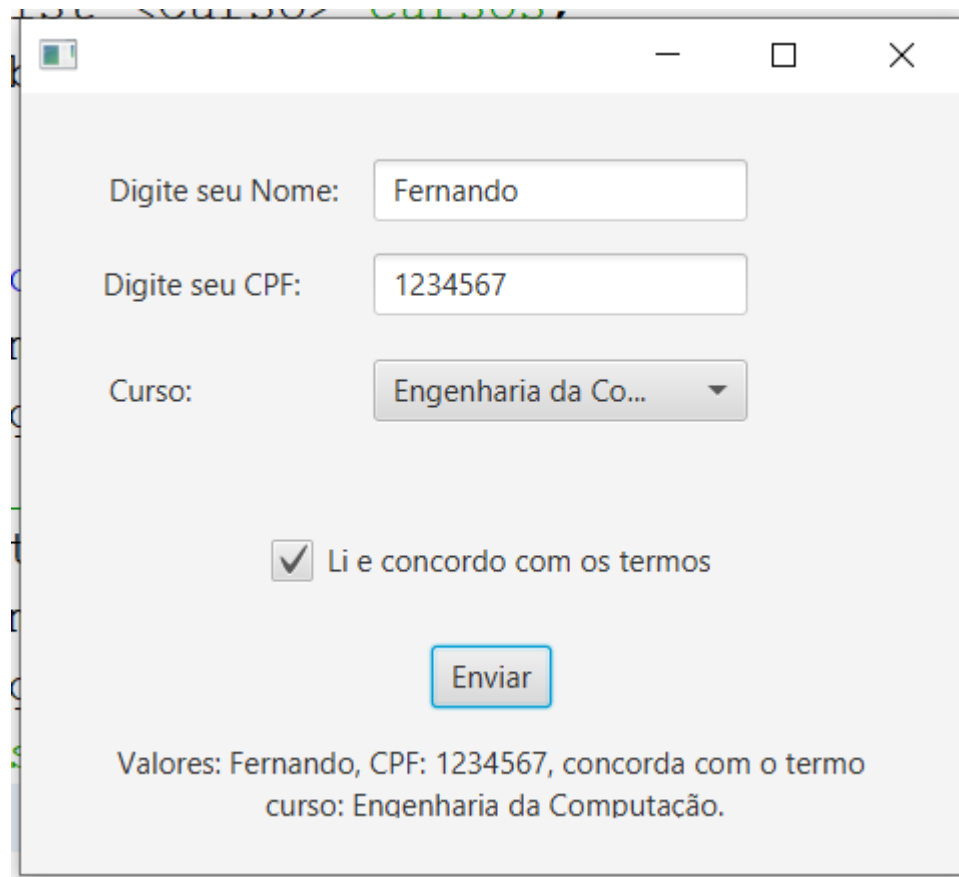
Trabalhando com o Scene Builder

- ▶ Na função do botão, podemos colocar instruções de visualização dos dados que acabamos de criar:

```
47 @FXML
48 private void handleButtonAction(ActionEvent event) {
49     System.out.println("Debug: Botão Pressionado!");
50     String termo;
51     if(ex_checkBox.isSelected()) termo = "concorda com o termo";
52     else termo = "discorda com o termo";
53     System.out.println("Debug Message: Funcionou!");
54     String tcurso = pegaCurso().getNome();
55     ex_resp.setText("Valores: " + ex_texto.getText() + ", CPF: " + ex_cpf.getText()
56         + ", " + termo + ", curso: " + tcurso + ".");
57 }
```

Trabalhando com o Scene Builder

- Agora basta executar e testar o resultado:



A screenshot of a Java Swing window titled "Formulario" (Form). The window contains a form with the following elements:

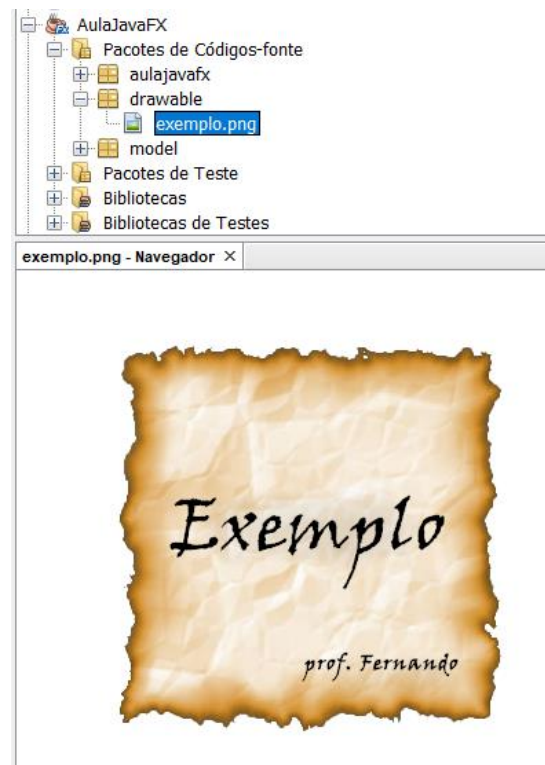
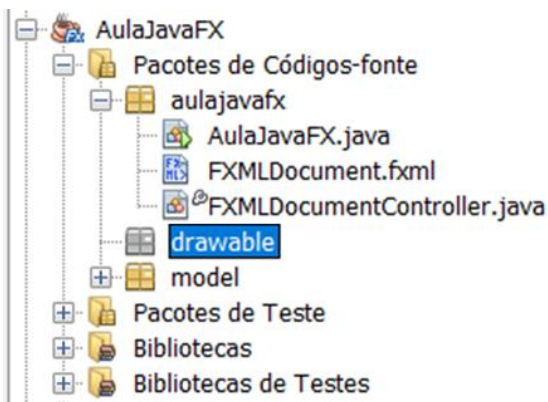
- A text field labeled "Digite seu Nome:" containing the text "Fernando".
- A text field labeled "Digite seu CPF:" containing the text "1234567".
- A dropdown menu labeled "Curso:" with the selected option "Engenharia da Co..." (Engineering of Co...).
- A checkbox labeled "Li e concordo com os termos" (I read and agree with the terms), which is checked.
- A button labeled "Enviar" (Send).
- A summary text at the bottom: "Valores: Fernando, CPF: 1234567, concorda com o termo curso: Engenharia da Computação." (Values: Fernando, CPF: 1234567, agrees with the term course: Engineering of Computing).

Adicionando Imagens - Scene Builder

- ▶ Algo simples e por vezes necessário para o desenvolvimento de um layout interessante seria adicionar imagens ao trabalho, ou projeto Java.
- ▶ Para adicionar imagens, no entanto, pode ser de duas formas: através do FXML (utilizando o Scene Builder) ou através de um CSS, que não será abordado nesta aula.
- ▶ A imagem deve ser trazida para o contexto do programa para que desta forma, seja realizado o link pelo programa Java e a figura seja mostrada.

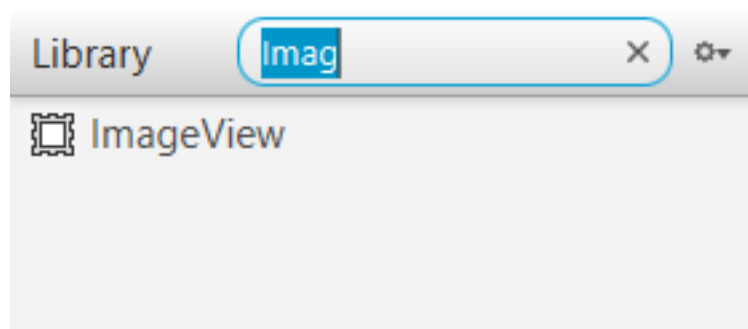
Adicionando Imagens - Scene Builder

- ▶ Desta forma, precisaremos adicioná-la a um pacote dentro do projeto, para que a imagem fique separada dos códigos.
- ▶ Abaixo, está sendo mostrado o pacote criado (drawable) e a imagem colocada neste pacote.



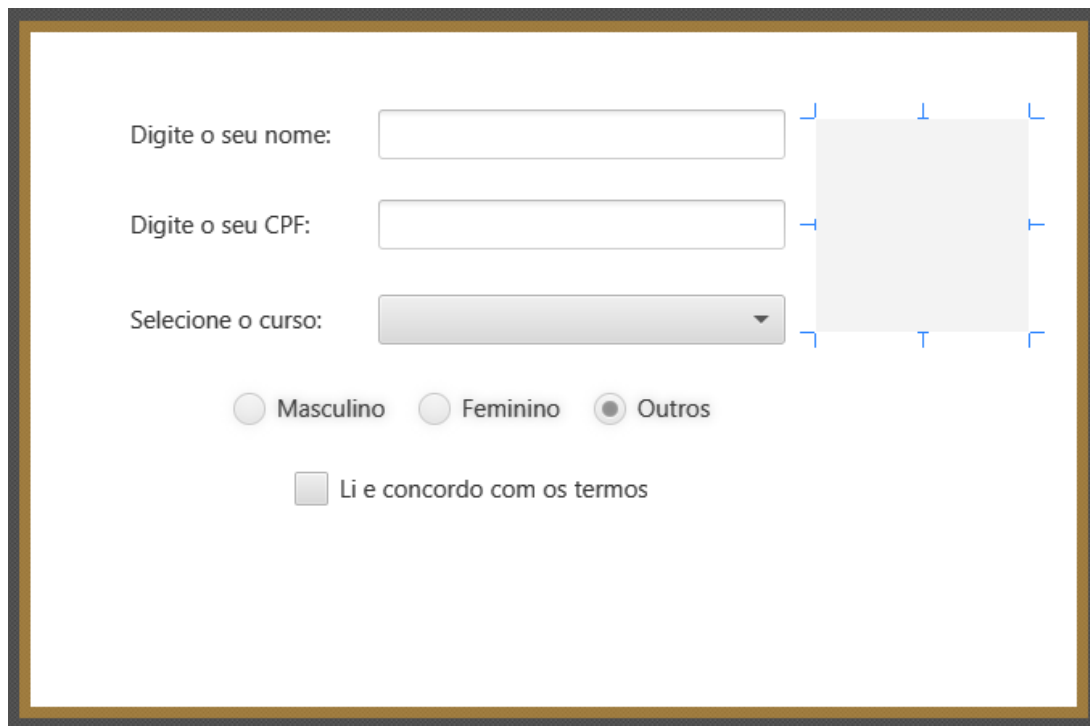
Adicionando Imagens - Scene Builder

- ▶ Precisamos agora inserir um ImageView pelo Scene Builder para que seja possível mostrar a imagem que acabamos de colocar.
- ▶ Acrescente do jeito que preferir... Lembre-se que pode ser adicionado vários tipos de painéis permitindo posicionamentos diferentes de cada componente.
- ▶ O componente pode ser facilmente buscado pela biblioteca (Library) acima dos componentes.



Adicionando Imagens - Scene Builder

- ▶ Adicione a ImageView ao cenário construído de forma que fique visível e que mantenha as proporções da imagem original (nossa imagem é quadrada).



Digite o seu nome:

Digite o seu CPF:

Selecione o curso:

☐ Masculino ☐ Feminino ☒ Outros

☐ Li e concordo com os termos

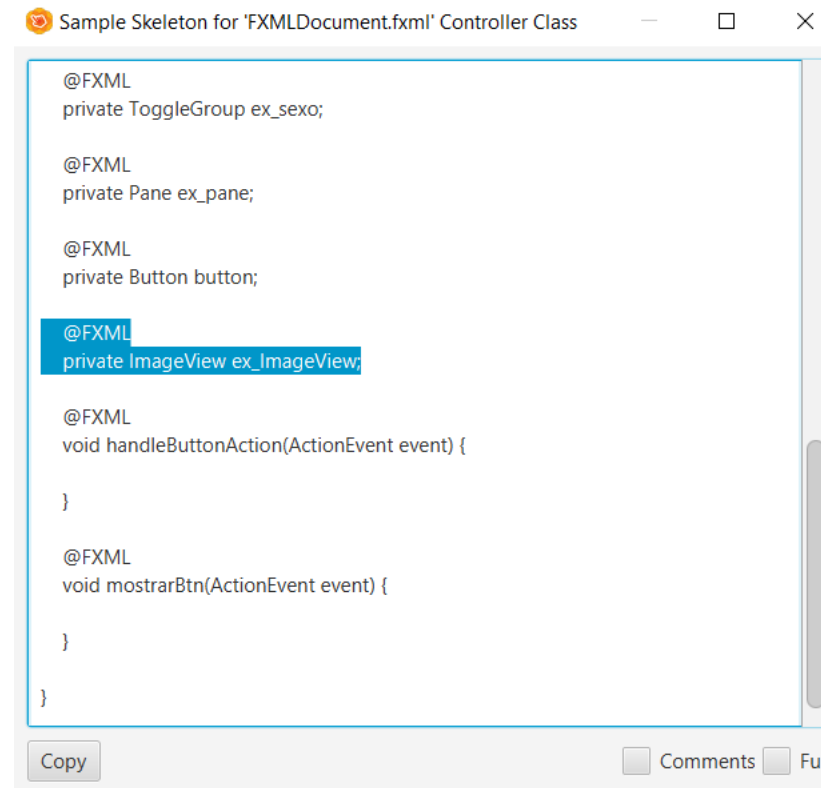
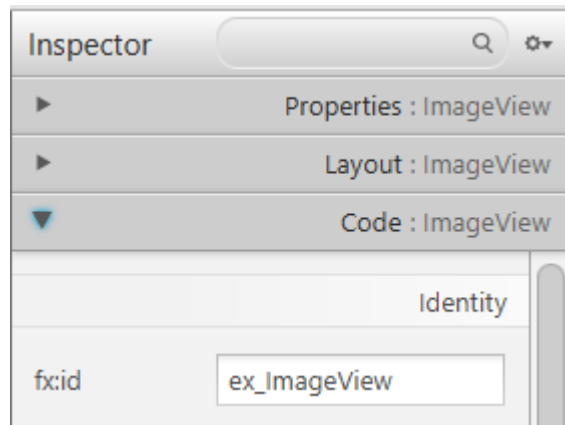
A gray square image placeholder is positioned to the right of the form fields, with blue handles indicating it is selected for layout configuration.

Configuração de Layout

Size	
Fit Width	<input type="text" value="132"/>
Fit Height	<input type="text" value="132"/>

Adicionando Imagens - Scene Builder

- Desta forma, acrescente um id ao ImageView (neste caso, ex_imageView), salve o Scene Builder e busque os itens modificados no “Controller Skeleton”



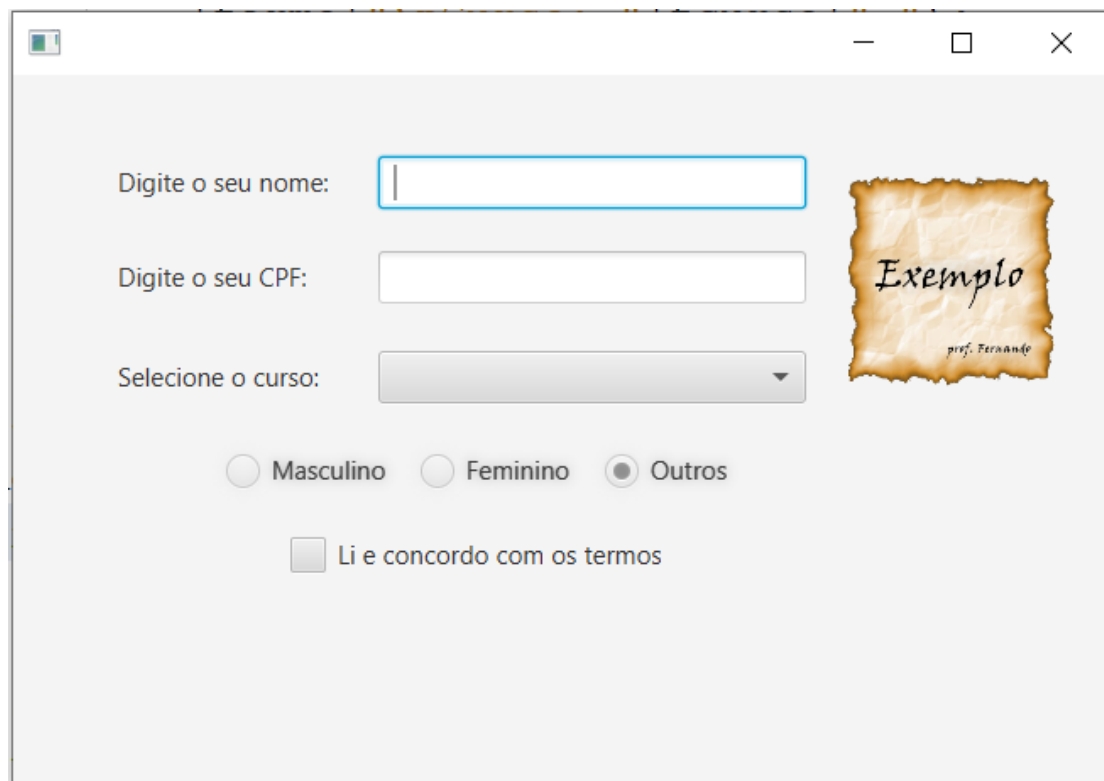
Adicionando Imagens - Scene Builder

- ▶ Voltando ao controller do Java, adicionaremos o código copiado do Skelletion com as devidas importações e na inicialização (initialize), precisaremos indicar o caminho da imagem

```
40  @FXML
41  private ToggleGroup ex_sexo;
42  @FXML
43  private Pane ex_pane;
44  @FXML
45  private ImageView ex_ImageView;
46
65  @Override
66  public void initialize(URL url, ResourceBundle rb) {
67      carregaCursos();
68      mostrarBtn();
69      ex_ImageView.setImage(new Image("drawable/exemplo.png"));
70  }
71
```

Adicionando Imagens - Scene Builder

- Desta forma, podemos observar o resultado ao executar o nosso programa:



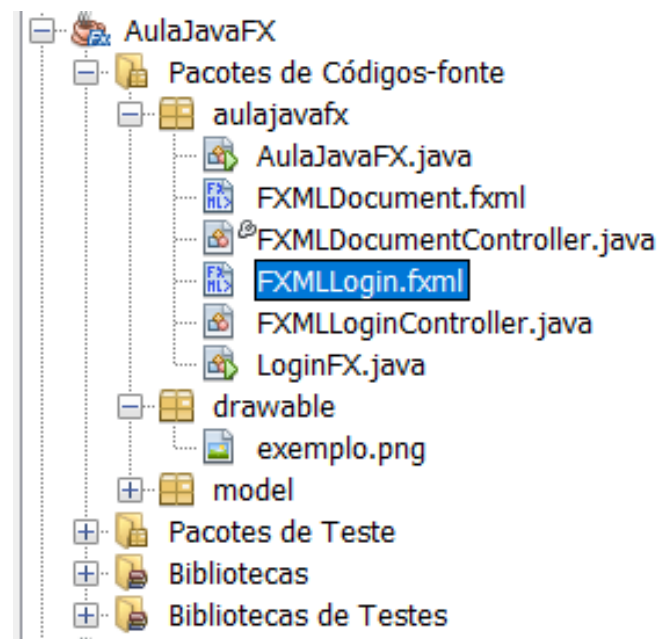
A screenshot of a Java Swing window titled "Exemplo" with a light gray background. The window contains a registration form with the following elements:

- A label "Digite o seu nome:" followed by a text input field with a blue border.
- A label "Digite o seu CPF:" followed by a text input field.
- A label "Selecione o curso:" followed by a dropdown menu.
- Three radio buttons for gender selection: "Masculino", "Feminino", and "Outros". The "Outros" option is selected.
- A checkbox labeled "Li e concordo com os termos".

On the right side of the form, there is a small, square, parchment-like image with the word "Exemplo" written in a stylized font and "prof. Fernando" written in smaller text below it.

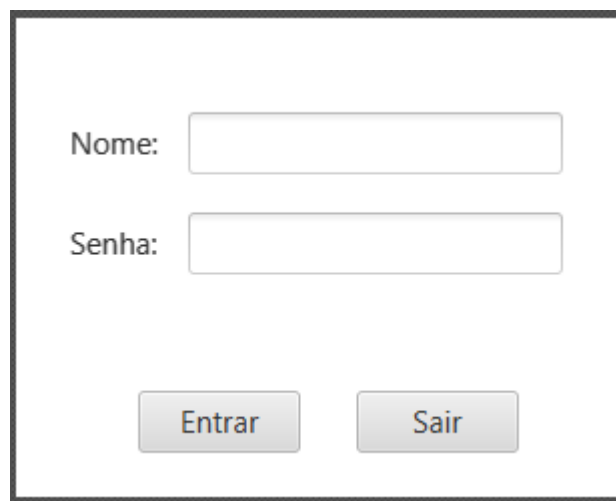
Transição de Telas

- ▶ Para trabalharmos com a transição de telas, inicialmente precisaremos criar alguns arquivos. Sendo eles:
 - ▶ FXML
 - ▶ Java Controller
 - ▶ Java Executável
- ▶ Trabalharemos uma tela de login criando os arquivos:
 - ▶ FXMLLogin.fxml
 - ▶ FXMLLoginController.java
 - ▶ LoginFX.java



Transição de Telas

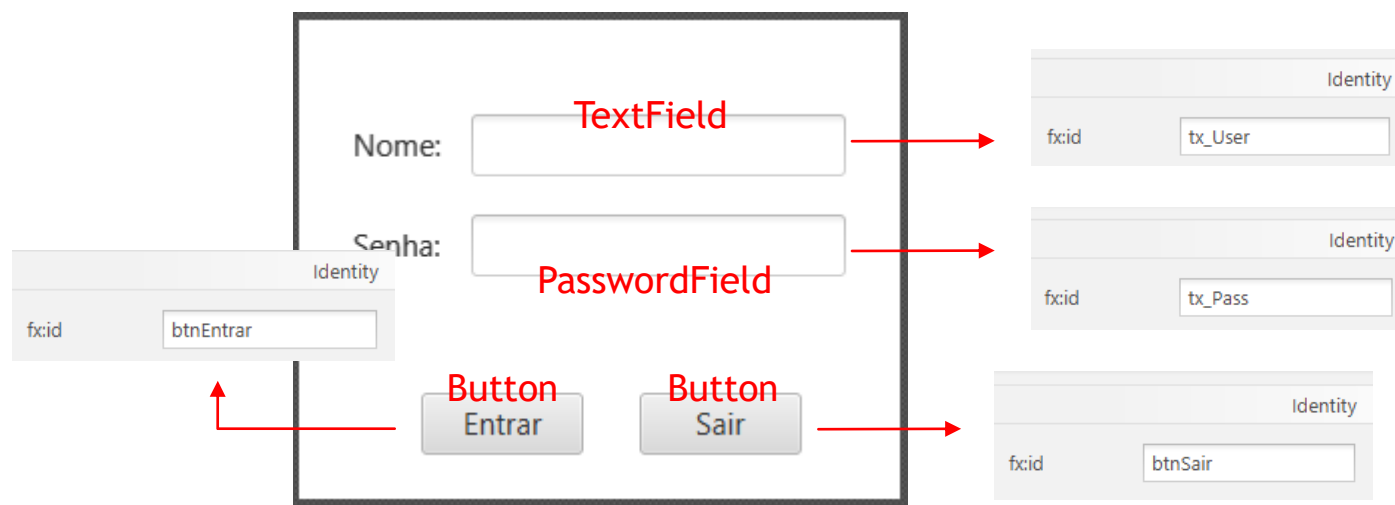
- ▶ Para o arquivo FXMLLogin.fxml, trabalharemos com o Scene Builder, criando labels e inputs necessários para uma tela de login, com seus respectivos ids.



A mockup of a login form, likely created in Scene Builder. It features a white rectangular area with a black border. Inside, there are two text input fields. The first field is preceded by the label 'Nome:' and the second by 'Senha:'. Below the input fields, there are two buttons: 'Entrar' (Enter) and 'Sair' (Exit). The buttons are light gray with rounded corners and black text.

Transição de Telas

- Para o arquivo FXMLLogin.fxml, trabalharemos com o Scene Builder, criando labels e inputs necessários para uma tela de login, com seus respectivos ids.



Transição de Telas

- E através do Skeleton, podemos linkar o fxml criado, com o controller java (FXMLLoginController.java)

```
Sample Skeleton for 'FXMLLogin.fxml' Controller Class

package aulajavaafx;

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;

public class FXMLLoginController {

    @FXML
    private TextField tx_User;

    @FXML
    private PasswordField tx_Pass;

    @FXML
    private Button btnEntrar;

    @FXML
    private Button btnSair;

}
```

Copy Comments Full



```
26      @FXML
27      private TextField tx_User;
28      @FXML
29      private PasswordField tx_Pass;
30      @FXML
31      private Button btnEntrar;
32      @FXML
33      private Button btnSair;
```

Transição de Telas

- Nosso controller java estará assim:

```
11 public class FXMLLoginController implements Initializable {  
12  
13     @FXML private TextField tx_User;  
14     @FXML private PasswordField tx_Pass;  
15     @FXML private Button btnEntrar;  
16     @FXML private Button btnSair;  
17  
18     @Override  
19     public void initialize(URL url, ResourceBundle rb) {  
20  
21     }  
22 }
```

Transição de Telas

- No programa principal (LoginFX.java), teremos a seguinte composição inicial:

```
1  package aulajavafx;
2  import javafx.application.Application;
3  import javafx.stage.Stage;
4
5  public class LoginFX extends Application {
6
7      @Override
8      public void start(Stage stage) throws Exception {
9
10     }
11
12     public static void main(String[] args) {
13         launch(args);
14     }
15
16 }
17
```


Transição de Telas

- Precisaremos agora iniciar e trabalhar com o Stage (classe do JavaFX responsável pelas telas). Criamos um objeto stage e faremos o set e o get para ele.

```
6 public class LoginFX extends Application {  
7  
8     private static Stage stage;  
9  
10    public static Stage getStage() {  
11        return stage;  
12    }  
13  
14    public static void setStage(Stage stage) {  
15        LoginFX.stage = stage;  
16    }  
17
```

Transição de Telas

- Finalizando o LoginFX.java, faremos o controle inicial da tela na função principal start mostrado na composição inicial:

```
21      @Override
22      ④ public void start(Stage stage) throws Exception {
23          Parent root = FXMLLoader.load(getClass()
24              .getResource("FXMLLogin.fxml"));
25          Scene scene = new Scene(root);
26          stage.setScene(scene);
27          stage.setTitle("Login");
28          setStage(stage);
29          stage.show();
30      }
```

Transição de Telas

- ▶ Com isto, poderemos agora voltar ao Controller java (FXMLLoginController.java) e no inicializar poderemos criar o controle do botão sair.
- ▶ A lógica do botão sair é simplesmente fechar o Stage que criamos no programa principal:

```
18      @Override
19      public void initialize(URL url, ResourceBundle rb) {
20          btnSair.setOnMouseClicked((event) -> {
21              System.out.println("Sair");
22              LoginFX.getStage().close();
23          });
24      }
```

Transição de Telas

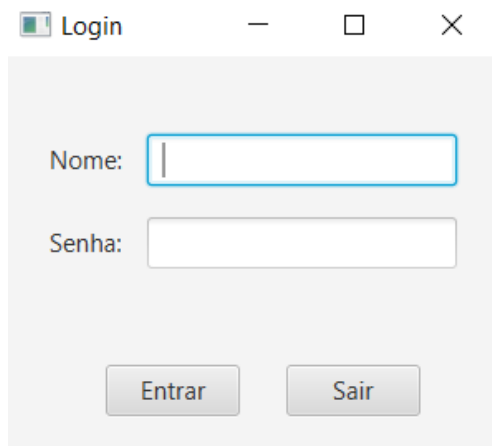
- ▶ Para o botão entrar, também no inicializar, faremos a transição de tela sem se preocupar com os dados inseridos, uma vez que para isso, será necessário acrescentar o banco de dados.
- ▶ Precisaremos iniciar a tela que queremos realizar a transição e fechar a tela antiga (neste caso a de login).

```
btnEntrar.setOnMouseClicked((event) -> {  
    System.out.println("Entrar");  
    AulaJavaFX principal = new AulaJavaFX();  
    LoginFX.getStage().close();  
    try {  
        principal.start(new Stage());  
    } catch (Exception ex) {  
        Logger.getLogger(FXMLLoginController.class.getName())  
            .log(Level.SEVERE, null, ex);  
    }  
});
```

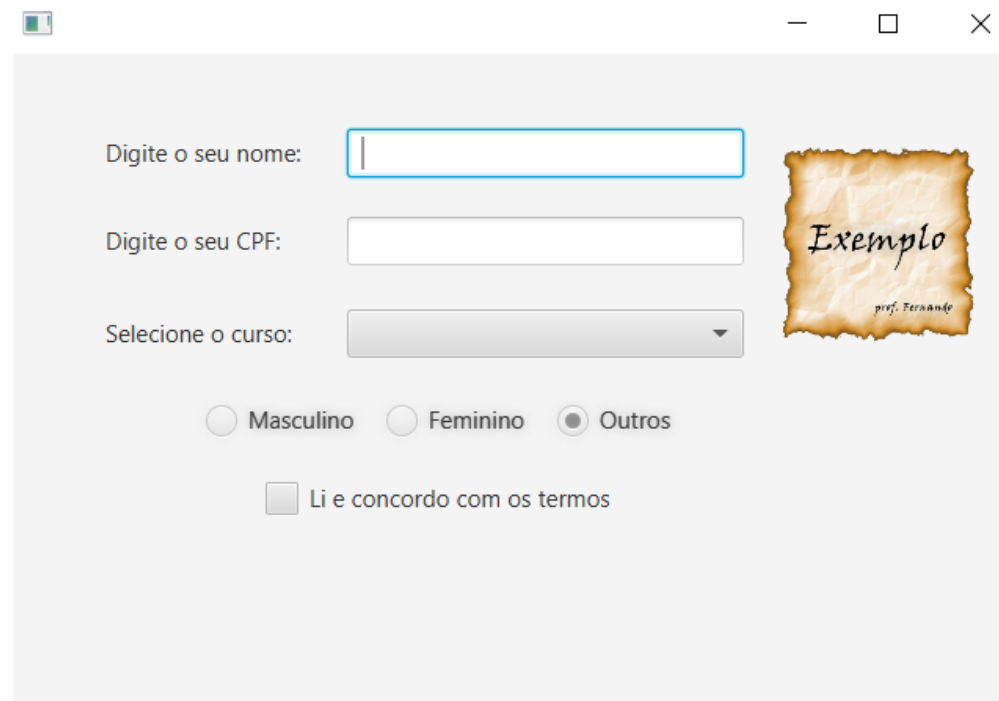
Classe principal da
tela que faremos a
transição

Transição de Telas

- Desta forma, podemos executar e verificar o resultado.



A small window titled "Login" with a standard Windows title bar (minimize, maximize, close buttons). It contains two text input fields: "Nome:" and "Senha:". Below the fields are two buttons: "Entrar" and "Sair".



A larger window titled "Exemplo" with a standard Windows title bar. It contains several form elements: a text input field for "Digite o seu nome:", a text input field for "Digite o seu CPF:", a dropdown menu for "Selecione o curso:", three radio buttons for "Masculino", "Feminino", and "Outros" (with "Outros" selected), and a checkbox for "Li e concordo com os termos". On the right side of the window is a small image of a piece of parchment with the word "Exemplo" written on it.

Exercícios

- ▶ Crie um restaurante virtual com as especificações:
 - ▶ Os dados deverão ser mostrado na tela.
 - ▶ Deverá apresentar um comboBox, checkBox, textView e labels (além do botão).
 - ▶ Ao pressionar o botão os valores digitados deverão ser apresentados em um label.
 - ▶ Customize os valores e layout no Scene Builder

Exercícios

- ▶ Crie dois ambientes de transição de telas de modo que:
 - ▶ No primeiro ambiente, seja mostrado uma tela principal com um botão para a tela secundária e da tela secundária um botão para voltar a tela principal.
 - ▶ No segundo ambiente, vários botões na tela principal, onde cada botão inicie uma tela diferente, sendo que cada tela contenha apenas uma imagem e um botão sair (a tela principal não pode ser fechada).