

Especialização em Desenvolvimento Java

Programação de Interfaces Gráficas

Prof. Vinícius de Paula

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



Objetivos desta aula

Objetivos desta aula

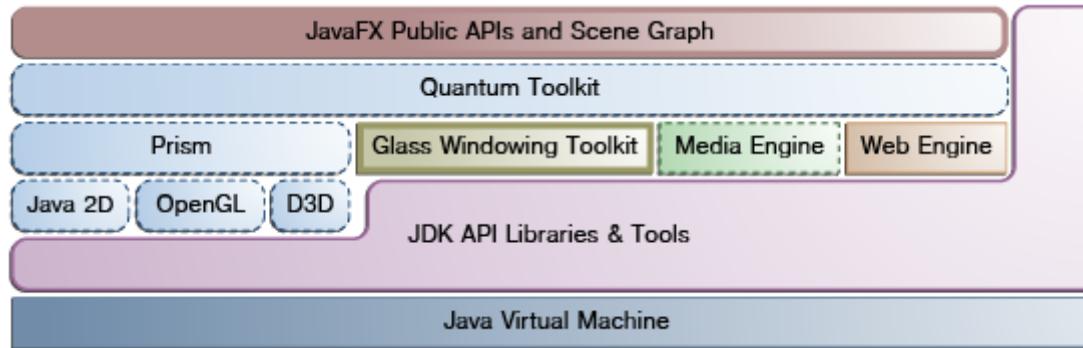
- Apresentar uma visão geral sobre o JavaFX;
- Apresentar os fundamentos conceituais e práticos relacionados ao desenvolvimento de uma aplicação JavaFX utilizando os princípios do MVC.

Noções sobre o JavaFX

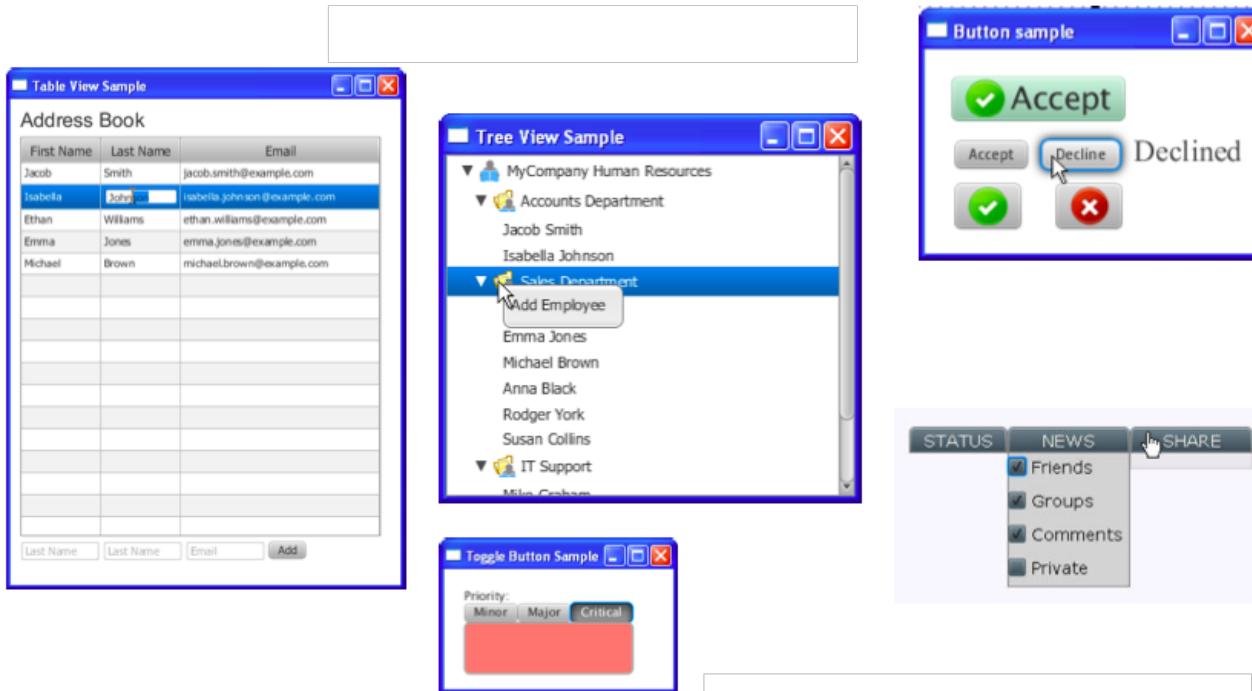
O que é JavaFX?

- Moderna biblioteca Java que dispõe de várias de vários recursos para criação de aplicações ricas.
 - Considerada a próxima geração de ferramentas para desenvolvimento de interfaces gráficas em Java.
 - Possui uma arquitetura otimizada para aproveitar a aceleração gráfica das GPUs modernas.
 - Segue o mantra do Java: “Write once, run anywhere”.

Arquitetura



O que podemos fazer com JavaFX?



Criar interfaces gráficas

O que podemos fazer com JavaFX?

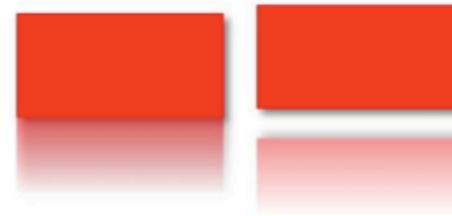
JavaFX
Lighting!

Reflection in JavaFX...
REFLECTION IN JAVAFX...

JavaFX drop shadow effect

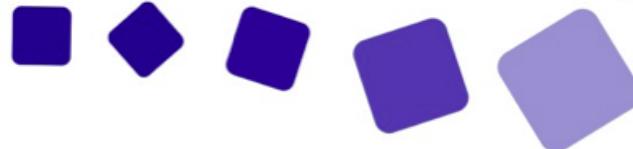


Aplicar efeitos



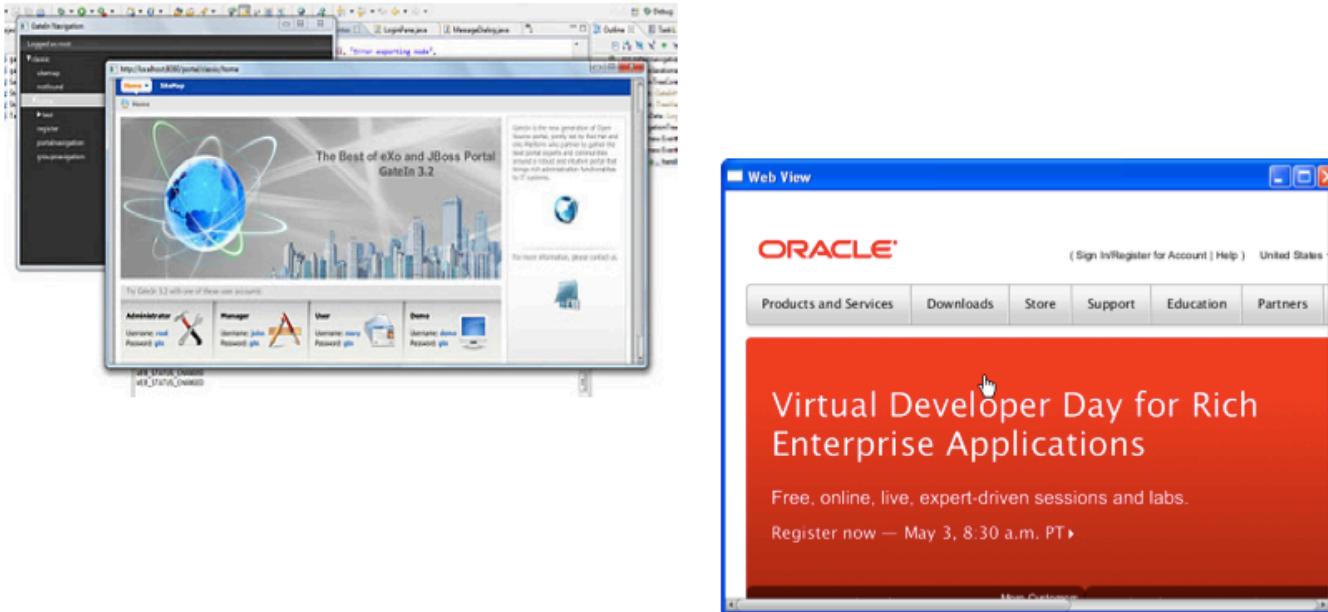
Inner Shadow
Inner Shadow

O que podemos fazer com JavaFX?



Criar animações

O que podemos fazer com JavaFX?



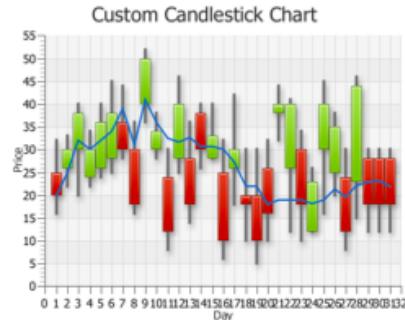
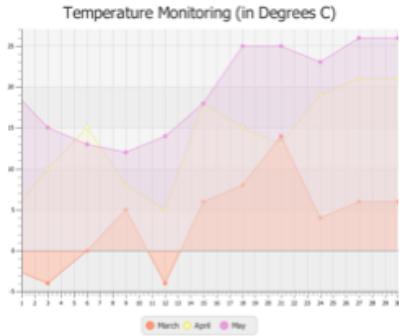
Renderizar conteúdo de páginas Web

O que podemos fazer com JavaFX?



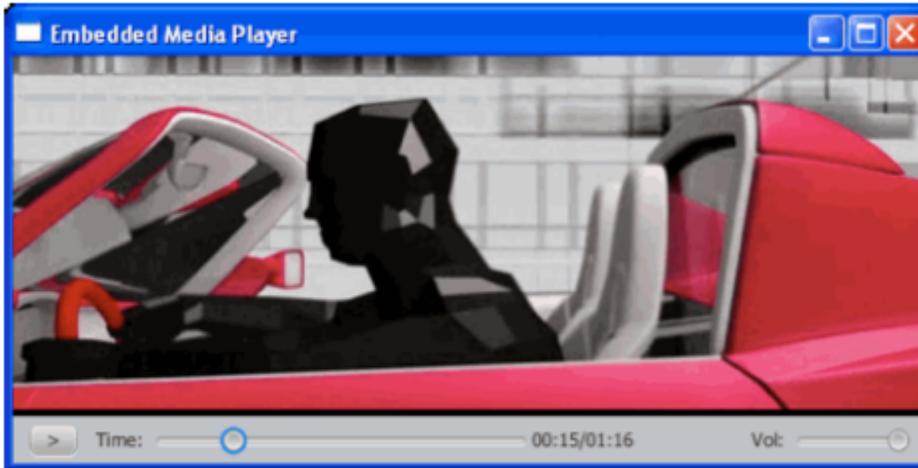
Utilizar CSS para alterar a aparência da aplicação

O que podemos fazer com JavaFX?



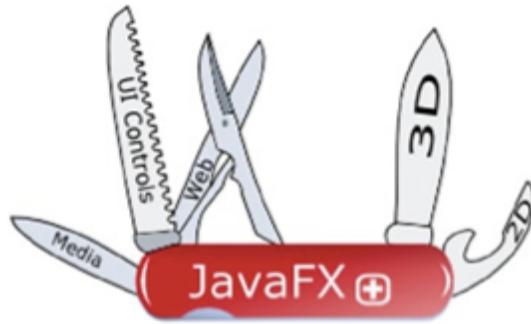
Criar gráficos

O que podemos fazer com JavaFX?



Executar conteúdo multimídia

JavaFX



“JavaFX is the Swiss Army Knife of GUI toolkits”

Conceitos Fundamentais

Classe Application

- Toda aplicação baseada em JavaFX deve extender a classe *javafx.application.Application*.
- A classe *Application* fornece os métodos para manipular o ciclo de vida de uma aplicação JavaFX.
 - `launch()`, `start()`, `stop()`

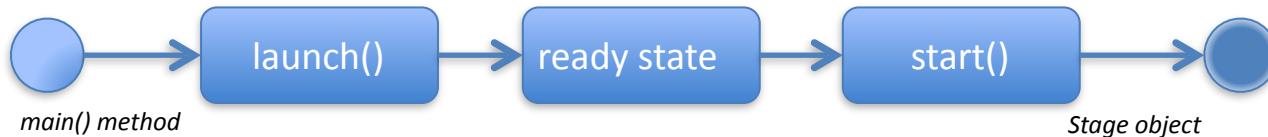


Diagrama de estados simplificado

Stage (Palco)

- Representa a janela principal da aplicação (top level container).
- Conceitualmente equivalente ao JFrame do *Swing*.
- A *Stage* principal é instanciada no processo de inicialização da aplicação e passada como argumento para o método *start()*.
- Uma aplicação JavaFX pode ter inúmeras *Stages*.

Scene ou Scene Graph (Cena)

- Ponto de partida para a construção de uma aplicação JavaFX.
- Representa a árvore hierárquica dos nós (Nodes) que compoem os elementos visuais da aplicação.

Scene ou Scene Graph (Cena)

- A API *javafx.scene* possibilita a criação de diversos tipos de conteúdo:
 - Nós: formas 2-D e 3-D, imagens, conteúdo de media, componentes de interface gráfica com o usuário, gráficos, conteúdo Web.
 - Estado: transformação (posicionamento e orientação dos nós), efeitos visuais e outros estados de conteúdo.
 - Efeitos: alteração da aparência de uma cena ou nó (brilho, sombra e outros ajustes na coloração).

Node (Nó)

- Qualquer elemento em um Scene é chamado de *Node*.
 - Cada Node possui um ID, um style e um class.
 - Cada Node em um Scene possui um pai e zero ou mais filhos (com exceção do *root node* de um Scene que não possui nenhum pai).
- Nodes também podem ter:
 - Efeitos
 - Manipuladores de eventos

Scene Builder

- Ferramenta visual para a construção de interfaces gráficas de aplicações JavaFX.
 - Clara separação de papéis (interface design/application logic).
 - Pode ser instalada separadamente, trabalha de forma integrada com as principais IDEs Java: NetBeans, Eclipse ou IntelliJ.
 - <http://www.oracle.com/technetwork/java/javafxscenebuilder-1x-archive-2199384.html>

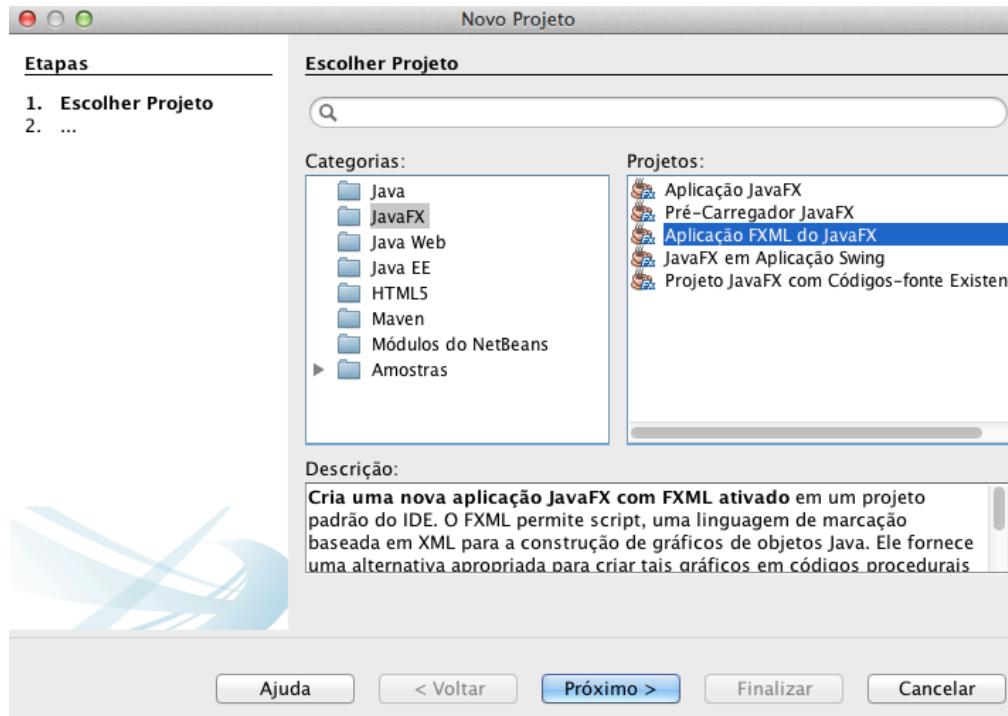
FXML

- Arquivo XML utilizado para definir os elementos da interface gráfica de uma aplicação JavaFX.
 - Similar ao:
 - MXML (Adobe)
 - XAML (Microsoft)
- Sua utilização possibilita a separação da camada de apresentação da lógica da aplicação.

Primeira Aplicação

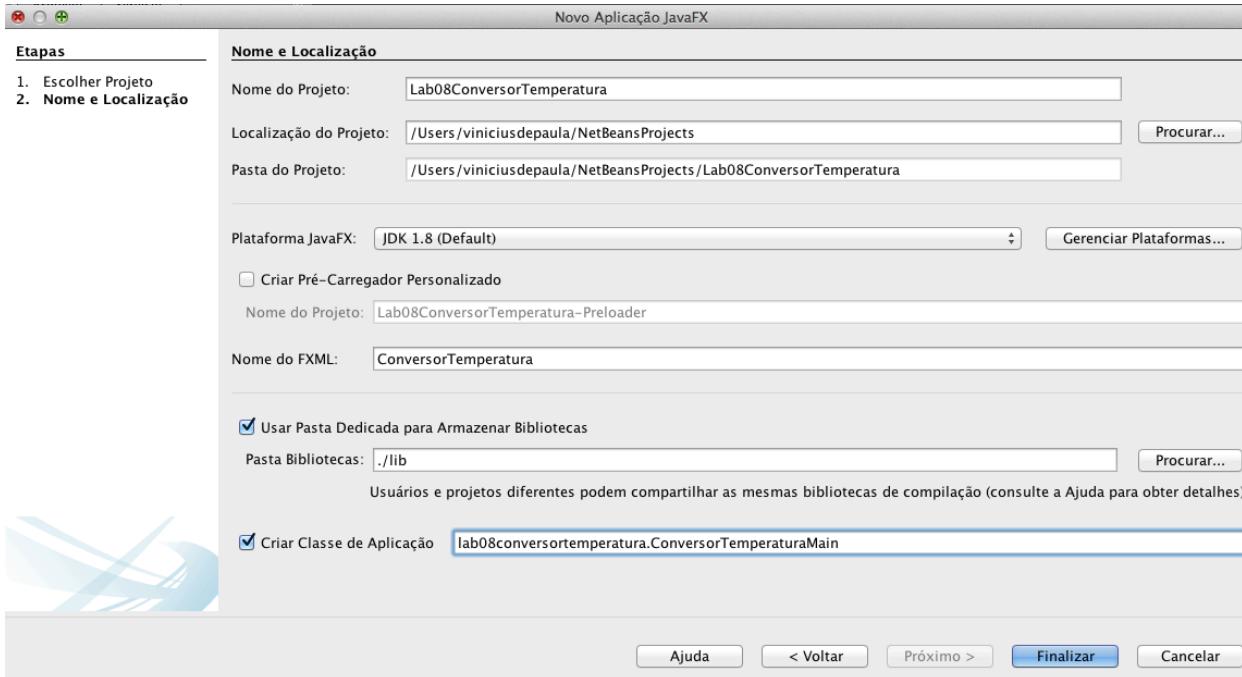
JavaFX

Primeira Aplicação JavaFX



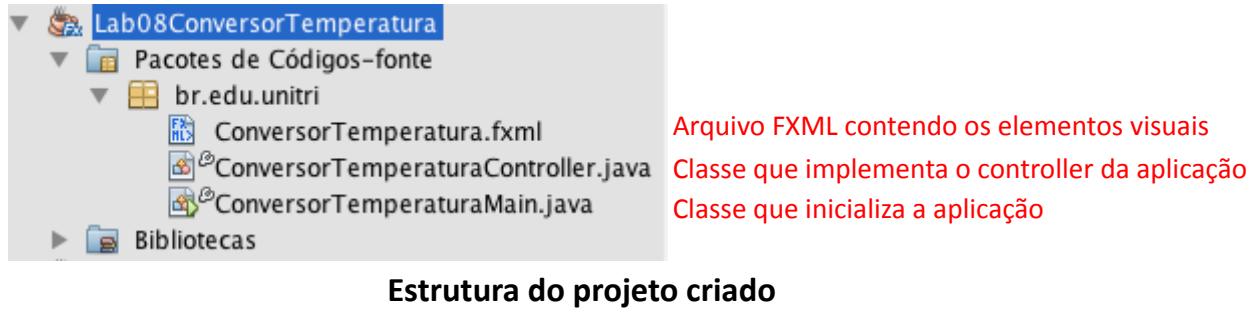
Seleção do tipo de projeto

Primeira Aplicação JavaFX



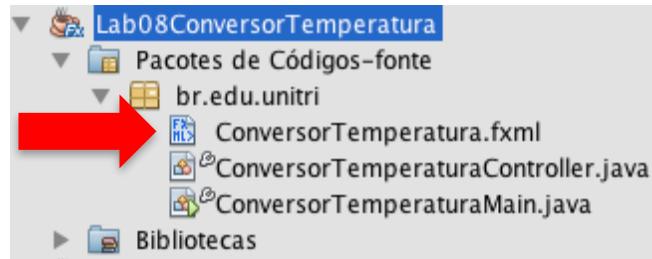
Parametrização do projeto

Primeira Aplicação JavaFX

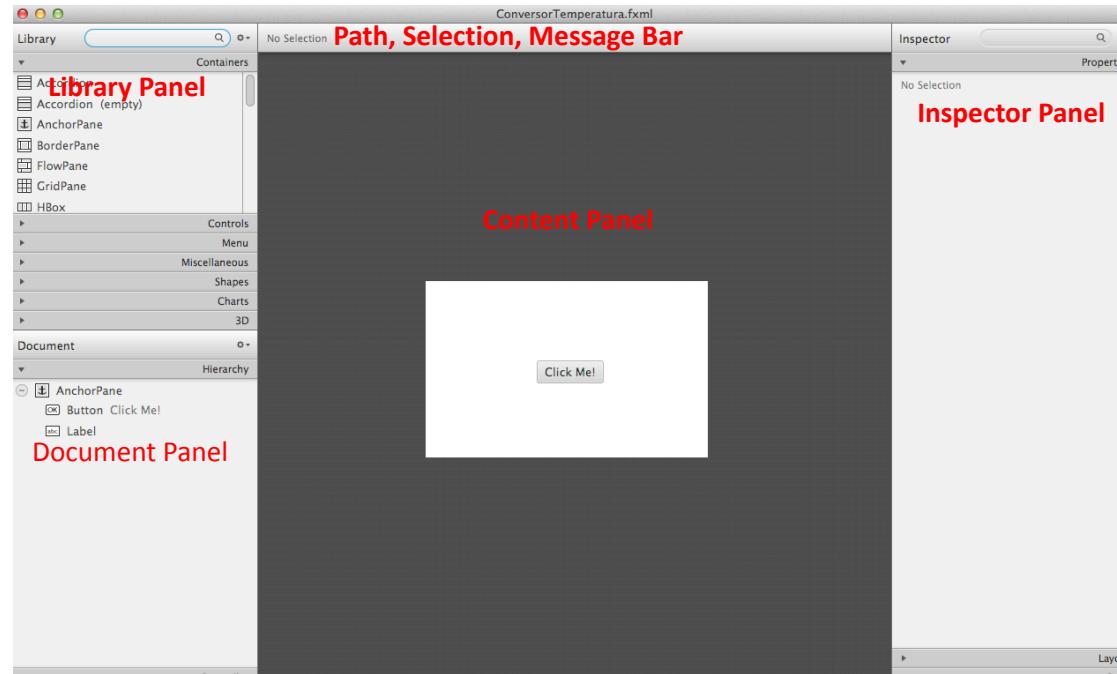


Primeira Aplicação JavaFX

- Para visualizar e editar os elementos visuais da aplicação no JavaFX *Scene Builder*, clique duas vezes sobre o arquivo FXML.



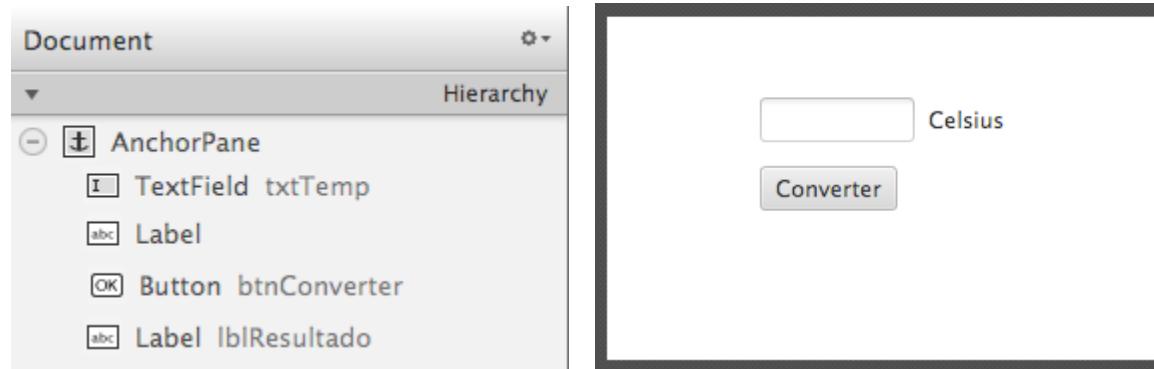
Primeira Aplicação JavaFX



Janela principal do JavaFX Scene Builder

Primeira Aplicação JavaFX

- A hierarquia dos componentes visuais da aplicação podem ser visualizada no *Document Panel* e sua aparência no *Content Panel*.



Primeira Aplicação JavaFX

- Antes de implementar o método que tratará o evento de clique no botão, temos que fazer o sincronismo entre o arquivo *FXML* e a classe Java que representa o *controller*.



Este sincronismo é necessário para que o NetBeans gere as declarações das variáveis definidas na interface gráfica.

Primeira Aplicação JavaFX

```
public class ConversorTemperaturaController implements Initializable {  
    @FXML  
    private Label label;  
  
    @FXML  
    private void handleButtonAction(ActionEvent event) {  
        System.out.println("You clicked me!");  
        label.setText("Hello World!");  
    }  
  
    @Override  
    public void initialize(URL url, ResourceBundle rb) {  
        // TODO  
    }  
}
```

Remova o trecho acima da classe ConversorTemperaturaController

Primeira Aplicação JavaFX

- Para fazer o sincronismo entre o FXML e o controller:
 - 1) Clique com o botão direito sobre o arquivo FXML e em seguida “editar”.
 - 2) Na barra de menus selecione “Código Fonte” e em seguida “Controlador Make”.



Primeira Aplicação JavaFX

```
public class ConversorTemperaturaController implements Initializable {  
    @FXML  
    private TextField txtTemp;  
    @FXML  
    private Button btnConverter;  
    @FXML  
    private Label lblResultado;  
  
    @Override  
    public void initialize(URL url, ResourceBundle rb) {  
        // TODO  
    }  
}
```

Declaração das variáveis gerada pela IDE

Primeira Aplicação JavaFX

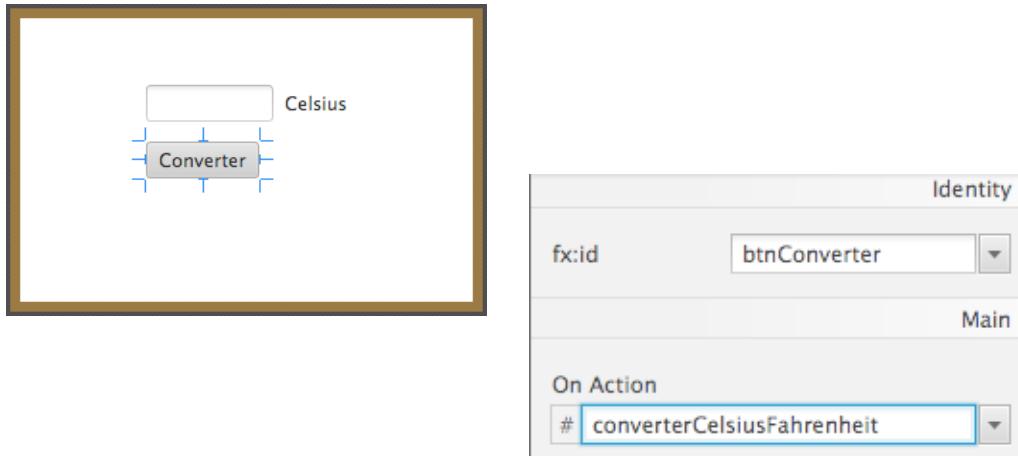
- A classe que representa o *controller* da aplicação será responsável por implementar o método que responderá ao clique do botão.

```
@FXML  
private void converterCelsiusFahrenheit(ActionEvent event) {  
  
    double tempFahr = (Double.parseDouble(txtTemp.getText()) * 1.8 + 32);  
    lblResultado.setText(tempFahr + " Fahrenheit");  
  
}
```

Implemente o método `converterCelsiusFahrenheit`
logo após a declaração das variáveis na classe `ConversorTemperaturaController`

Primeira Aplicação JavaFX

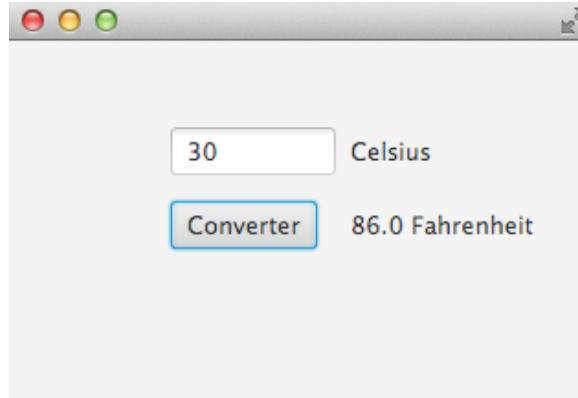
- Após a implementação do método que tratará o evento, temos que referenciá-lo na propriedade *On Action* do botão.



**Selecione o botão “Converter” e no “Inspector Panel > Code” defina em
On Action o método que tratará o evento**

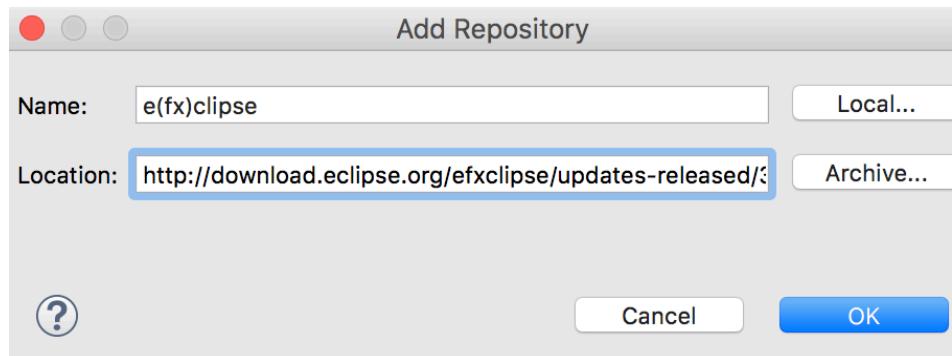
Primeira Aplicação JavaFX

- No Netbeans, execute o projeto clicando em “Executar Projeto”.

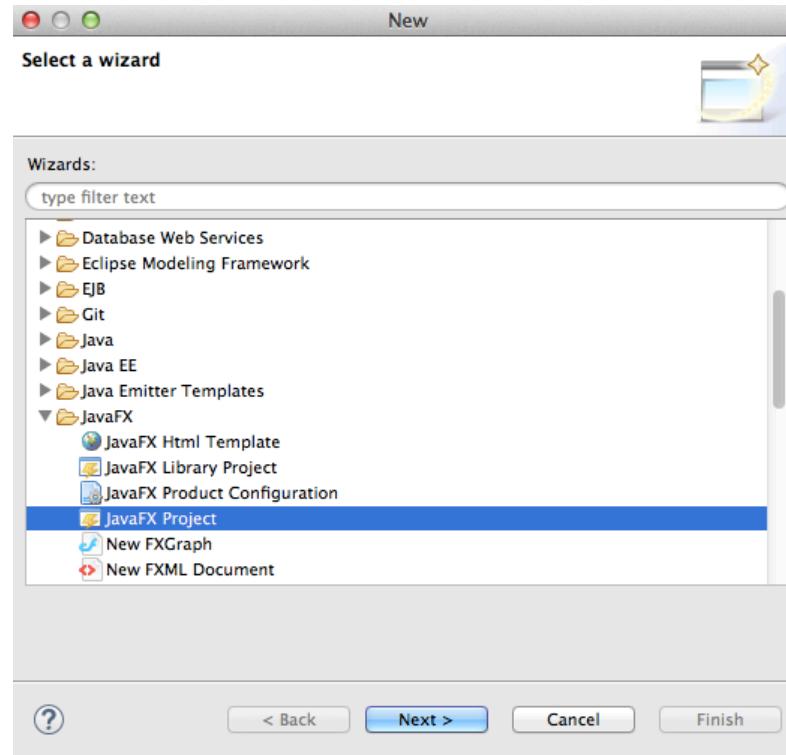


JavaFX no Eclipse IDE

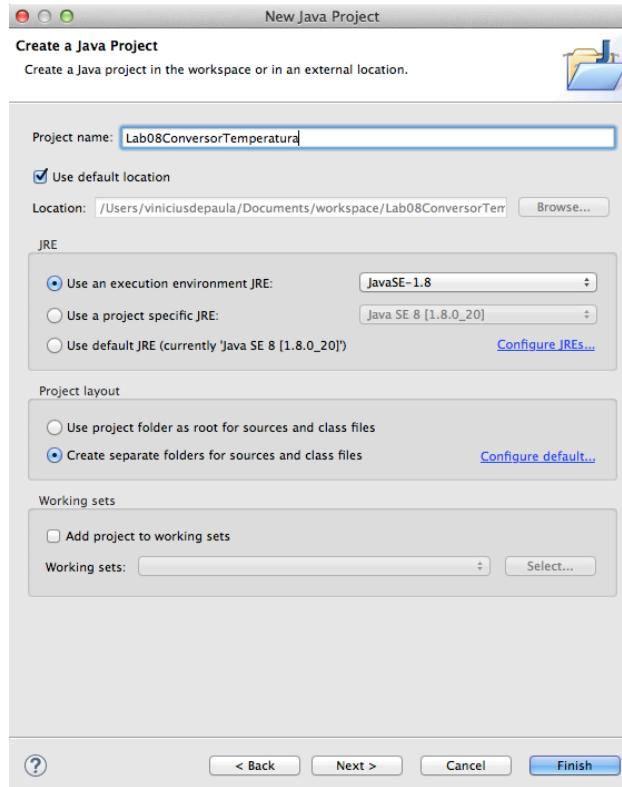
- É possível desenvolver JavaFX utilizando o Eclipse, para tal, se faz necessário a instalação do plugin e(fx)clipse.
 - Esta instalação pode ser feita adicionando o seguinte repositório:
 - **Name:** e(fx)clipse
 - **Location:** <http://download.eclipse.org/efxclipse/updates-released/3.0.0/site>



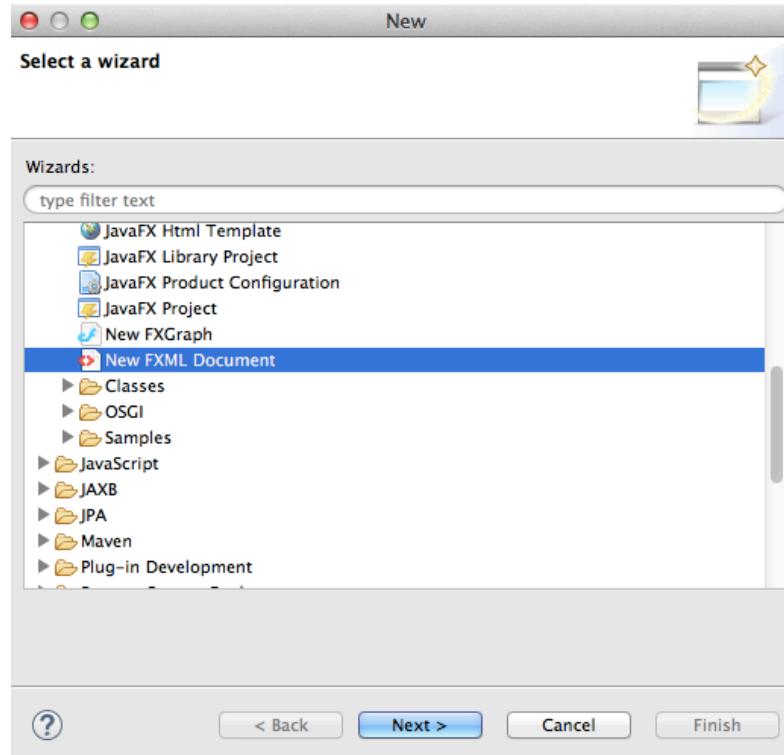
JavaFX no Eclipse IDE



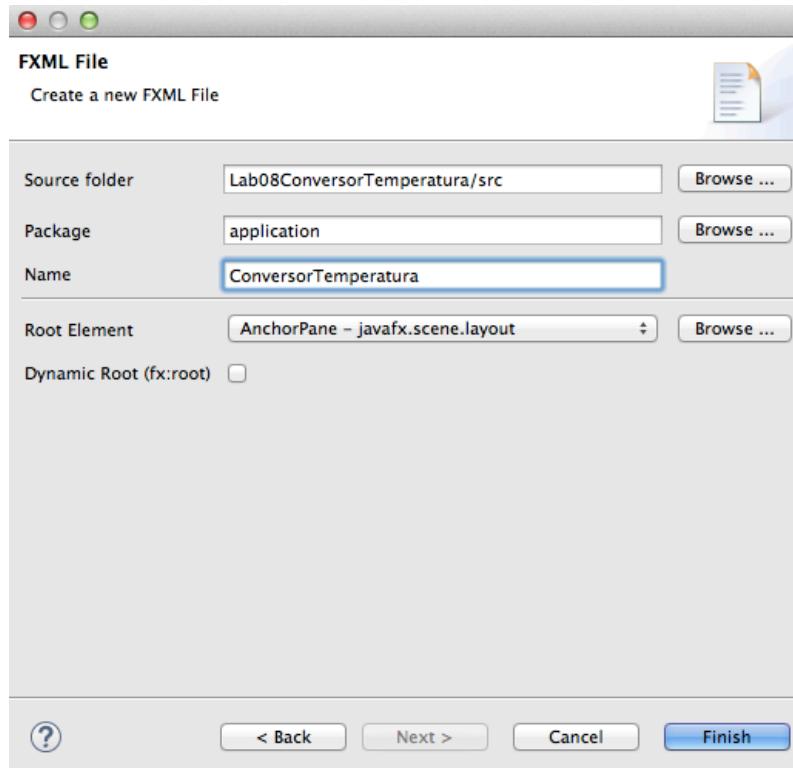
JavaFX no Eclipse IDE



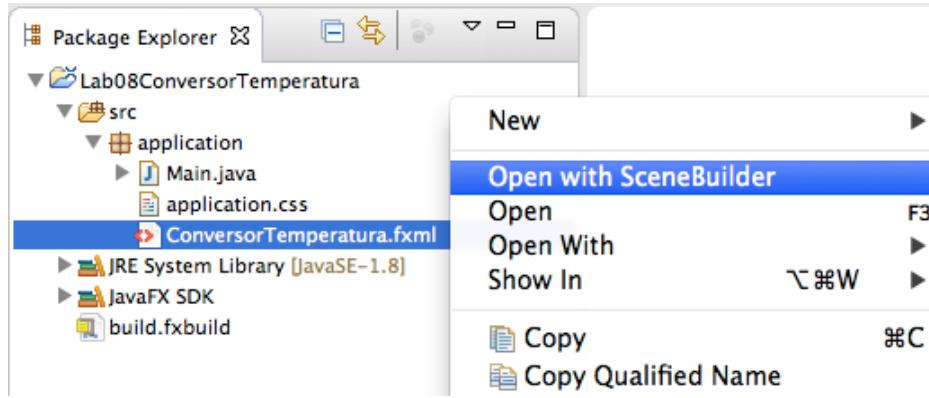
JavaFX no Eclipse IDE



JavaFX no Eclipse IDE



JavaFX no Eclipse IDE



Aplicação JavaFX com MVC

Aplicação JavaFX com MVC

- Visando o aprofundamento nos conceitos fundamentais que envolvem o JavaFX implementaremos a seguir uma aplicação baseada no padrão MVC.

Aparência da Aplicação

Aparência da Aplicação

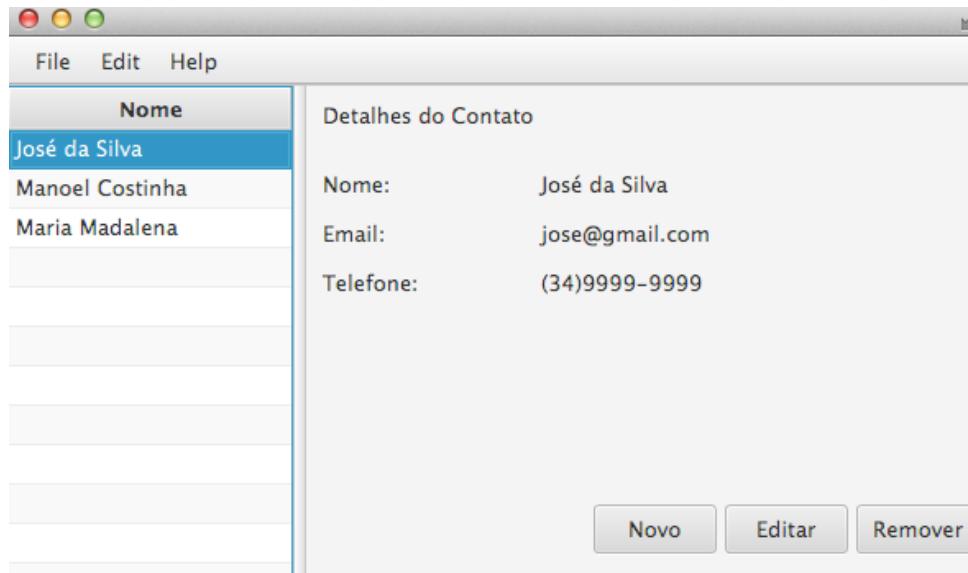
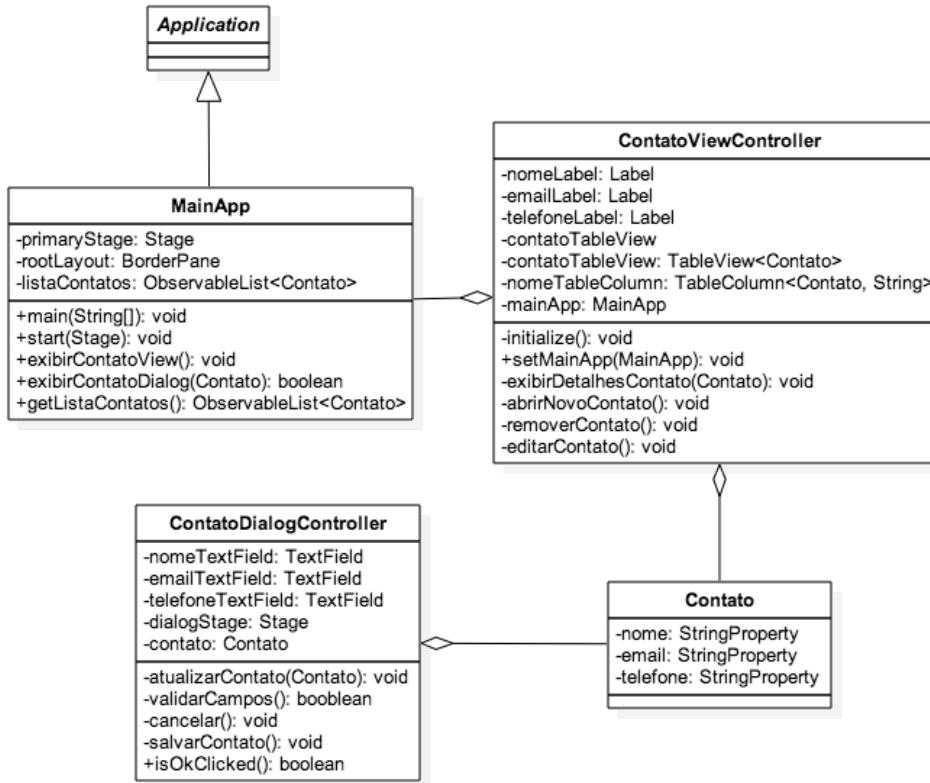


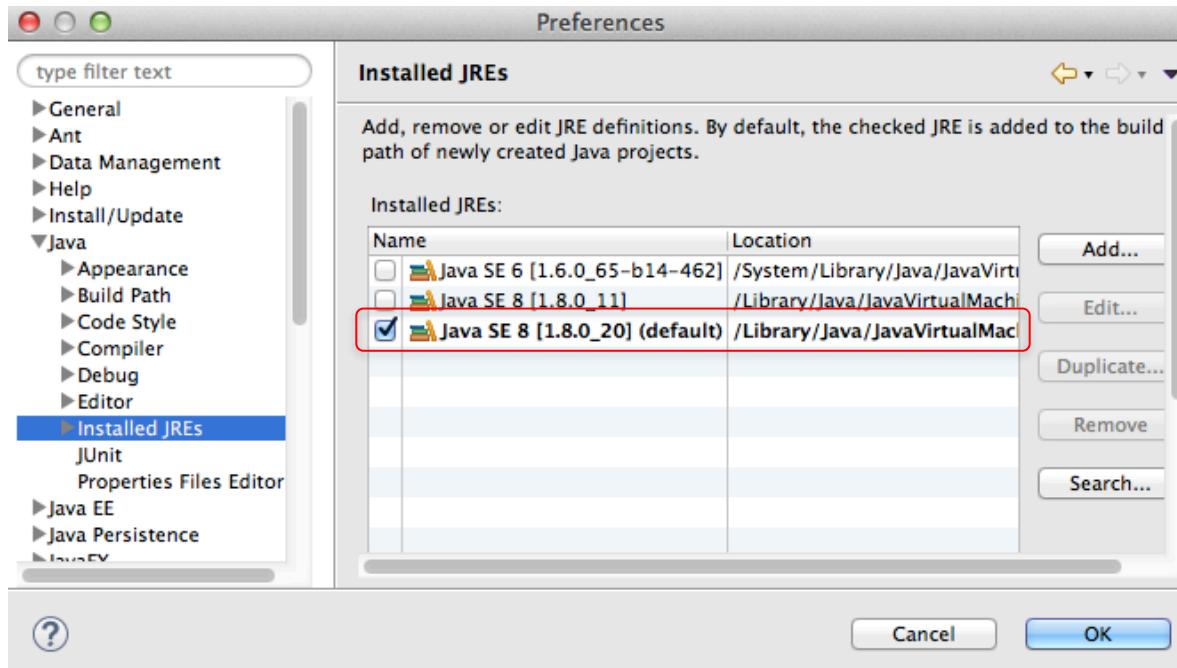
Diagrama de Classes

Diagrama de Classes



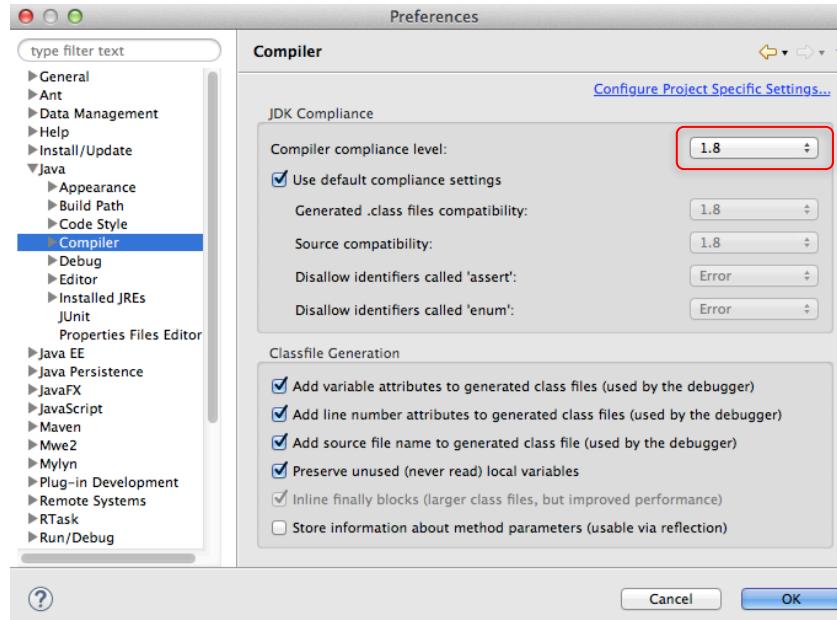
Checagem do Ambiente

Configurações do Eclipse



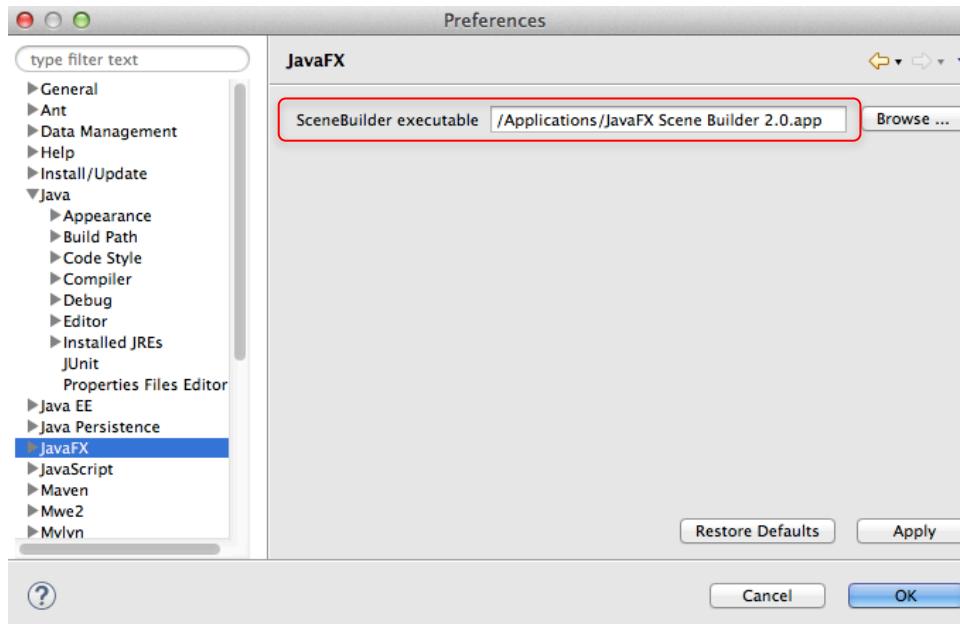
Acesse as preferências do Eclipse em seguida Java > Installed JREs

Configurações do Eclipse



Em seguida navegue até Compiler e defina o Compiler compliance level para 1.8

Configurações do Eclipse

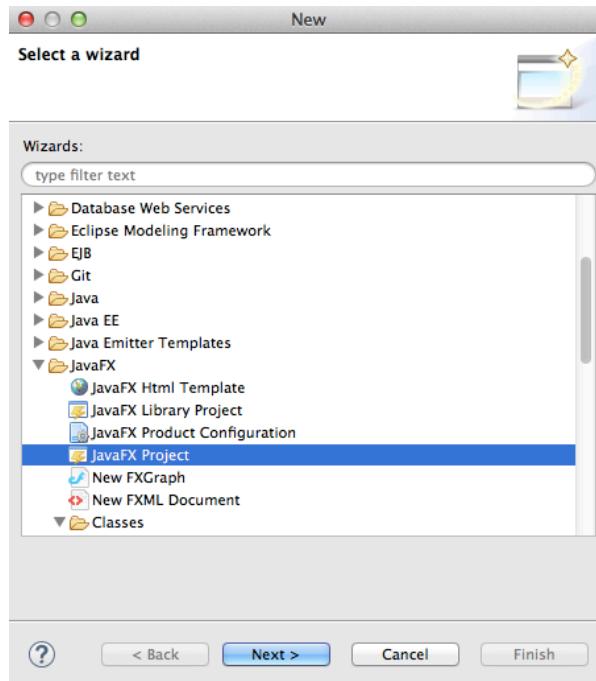


Na aba JavaFX defina o caminho do executável do Scene Builder

Primeira Parte

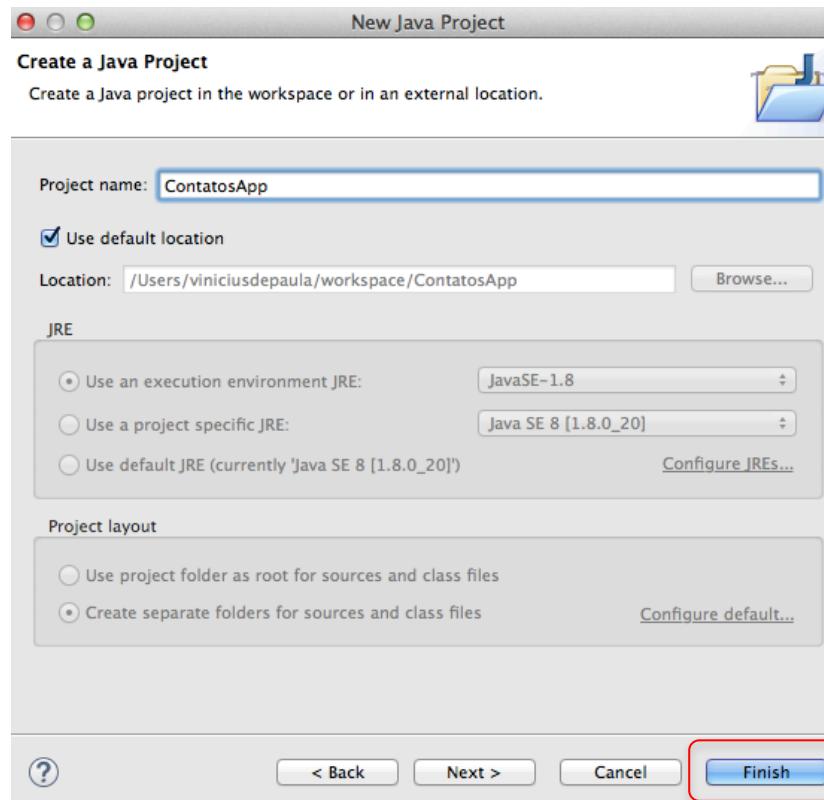
Layout da Aplicação

Criação do Projeto

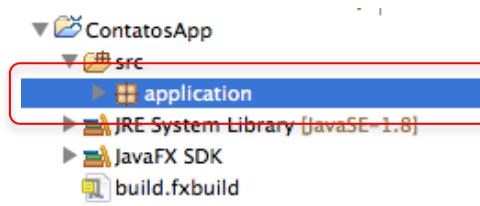


Crie um novo projeto em: **File > New > Other > JavaFX > JavaFX Project**

Criação do Projeto



Criação do Projeto



Remova o pacote application criado automaticamente pelo Eclipse

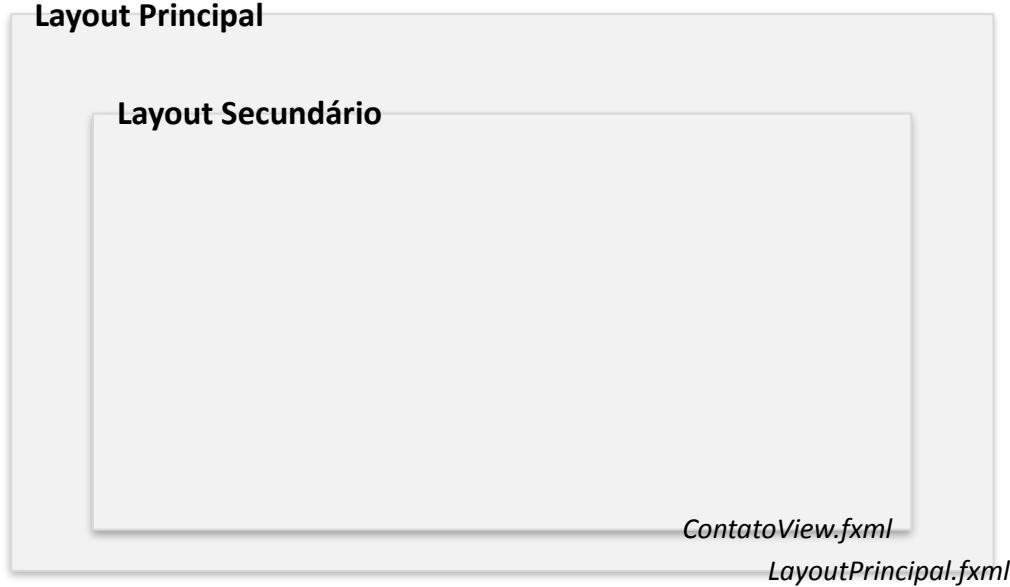
Criação do Projeto

Crie a seguinte estrutura de pacotes

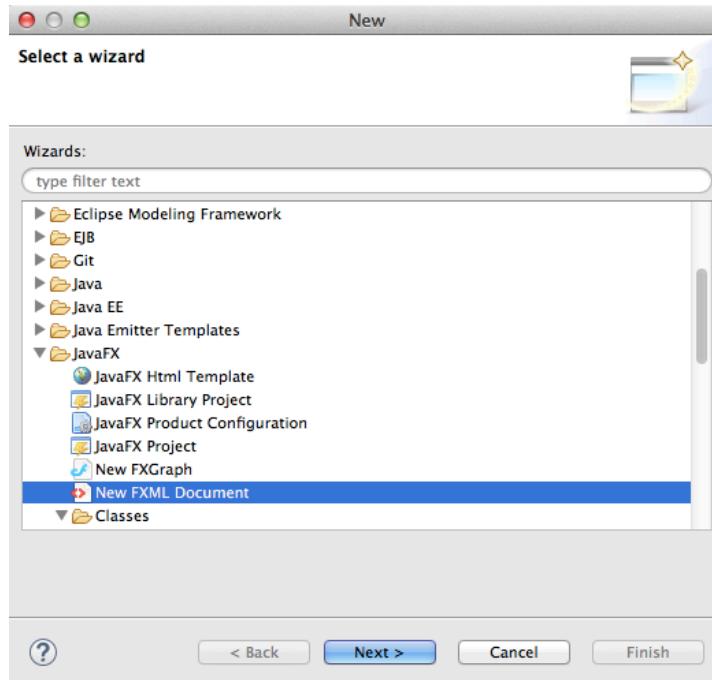


O pacote *view* também conterá alguns *controllers* que estarão ligados diretamente às visões, denominados *view-controllers*.

Estrutura do Layout da Aplicação

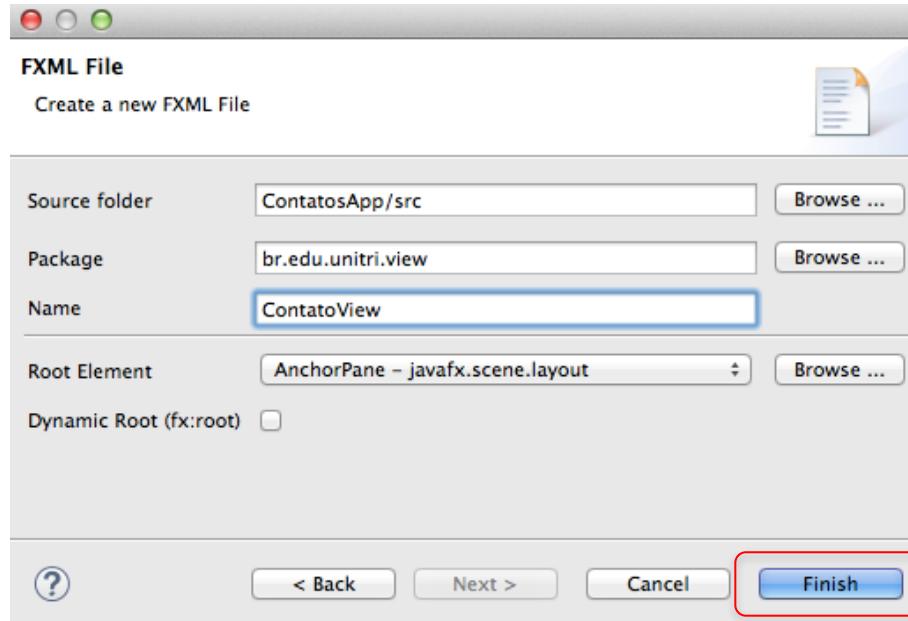


Criação do Layout Secundário



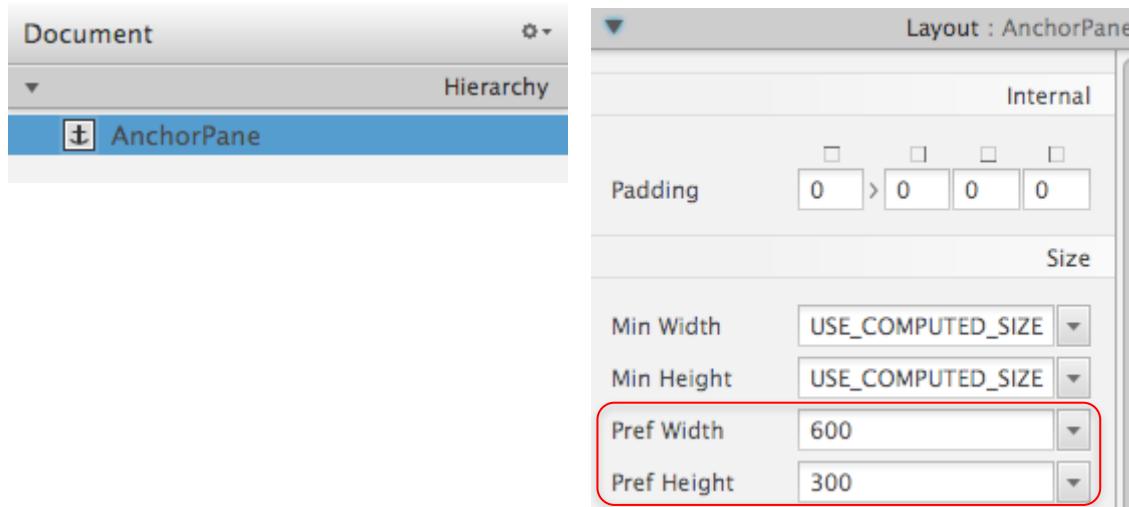
**Crie um novo arquivo FXML dentro do pacote view
New > Other > JavaFX > New FXML Document**

Criação do Layout Secundário



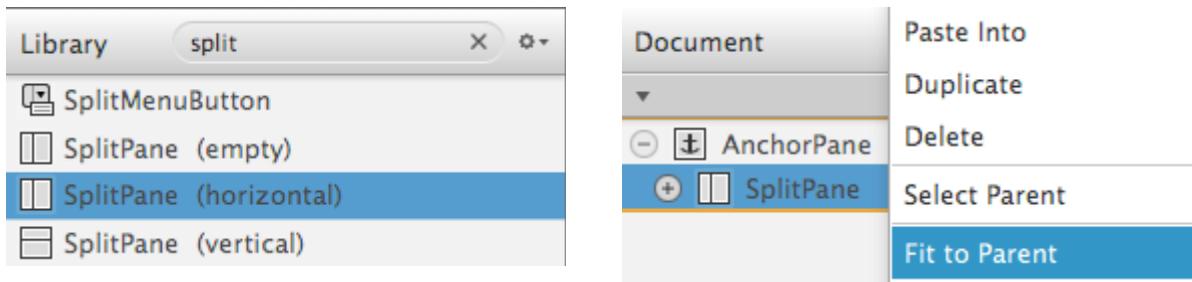
Defina o nome do arquivo como *ContatoView*

Criação do Layout Secundário



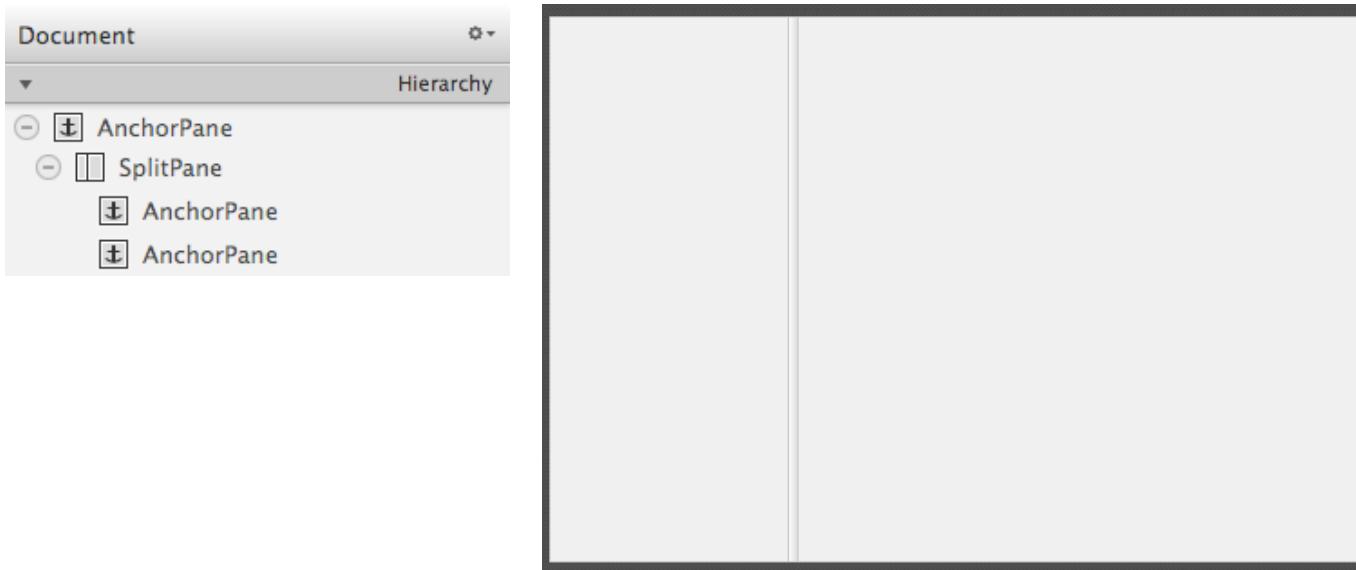
Abra o arquivo ContatoView.fxml no Scene Builder e
defina as dimensões do AnchorPane

Criação do Layout Secundário

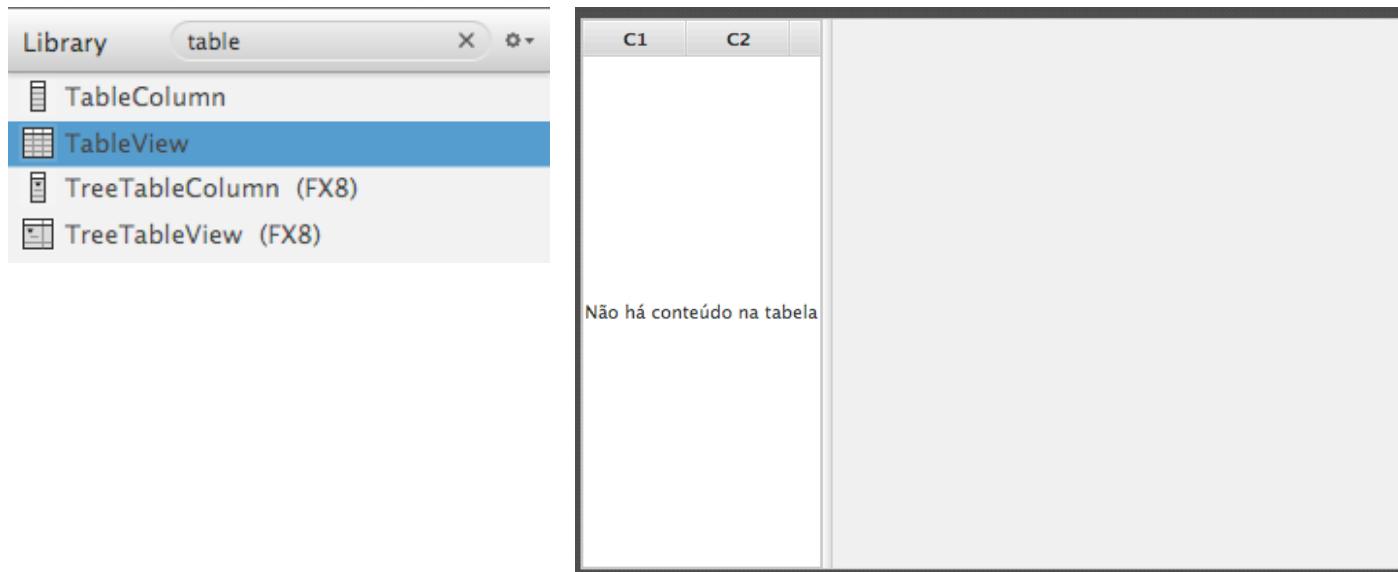


Adicione ao *Anchor Pane* um *SplitPane (Horizontal)* e em seguida defina a propriedade *Fit to Parent*

Criação do Layout Secundário

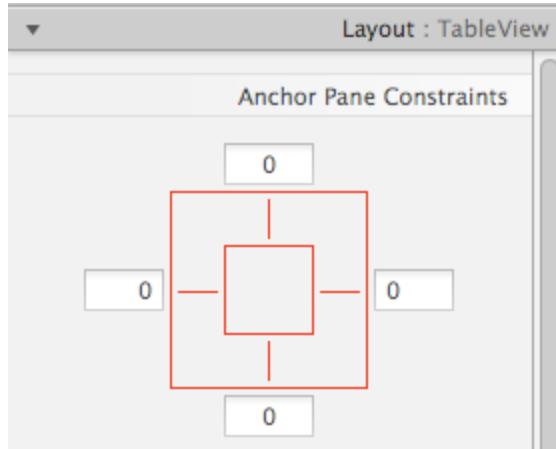


Criação do Layout Secundário



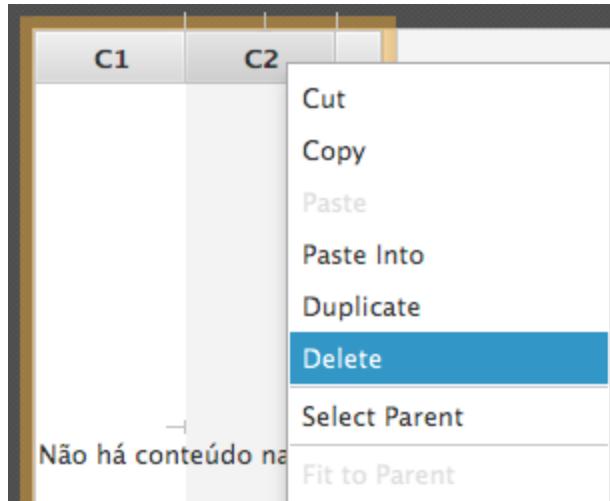
Adicione um *TableView* na coluna esquerda do *SplitPane*

Criação do Layout Secundário



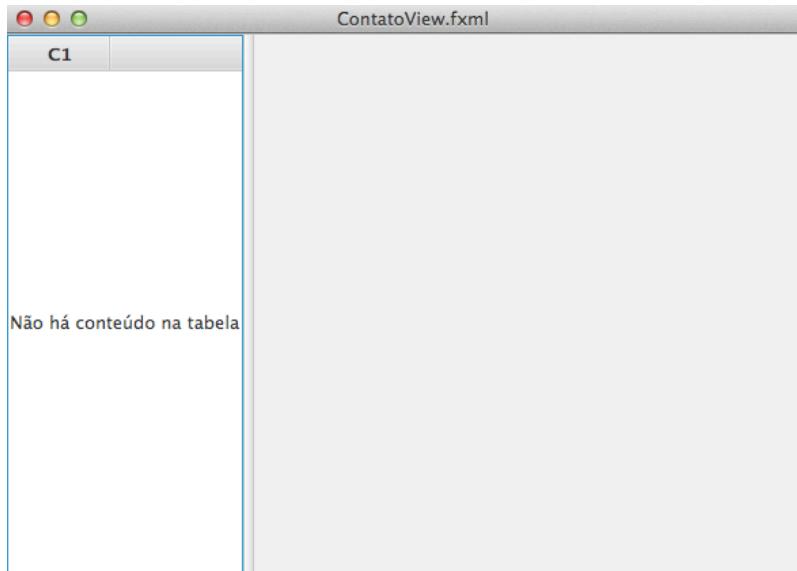
Defina as seguintes propriedades de layout do *TableView*

Criação do Layout Secundário



Remova a segunda coluna (C2) do *TableView*

Criação do Layout Secundário



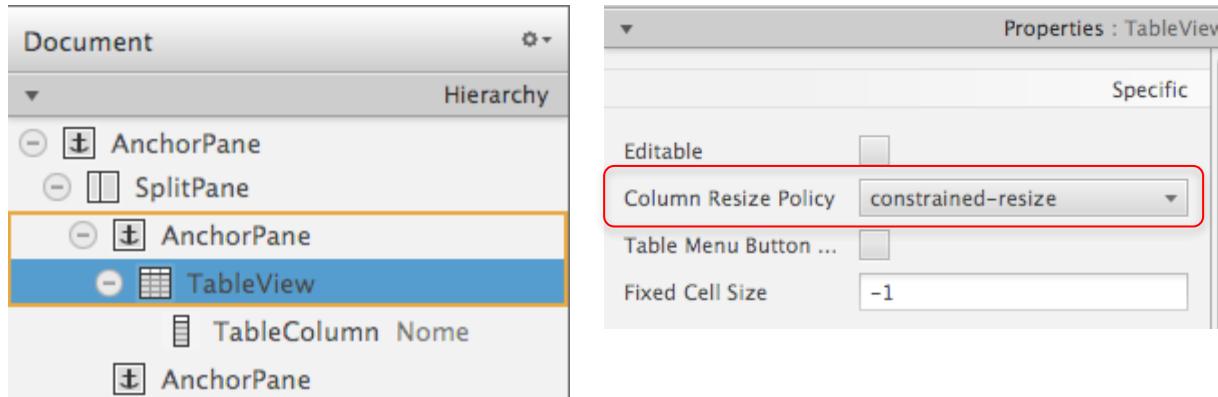
Preview do layout no Scene Builder: Menu Preview > Show Preview in Window

Criação do Layout Secundário

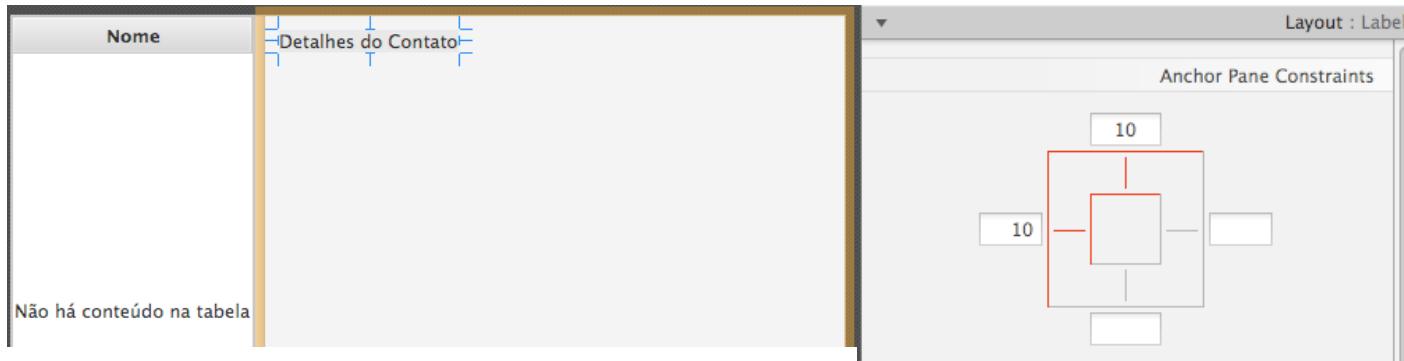
Nome
Não há conteúdo na tabela

Altere o nome do título da coluna do TableView

Criação do Layout Secundário

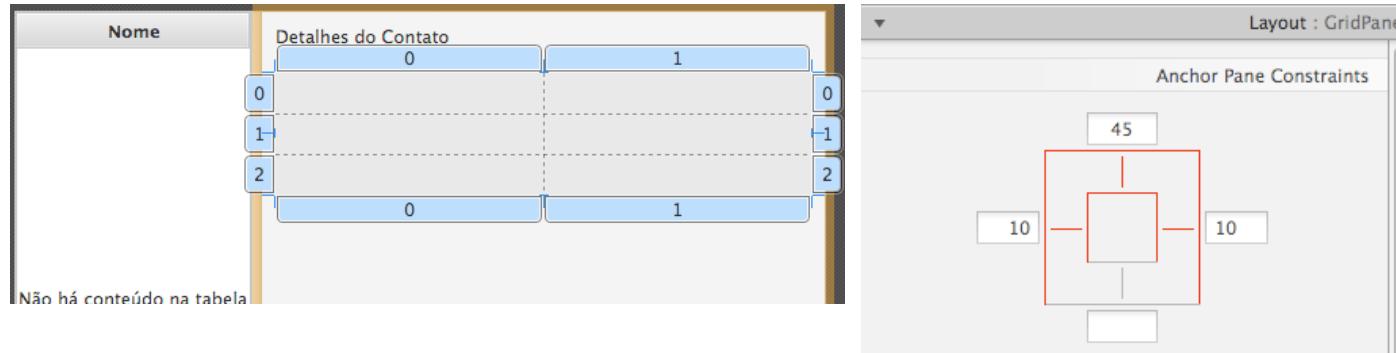


Criação do Layout Secundário



Adicione um label e defina as seguintes propriedades

Criação do Layout Secundário



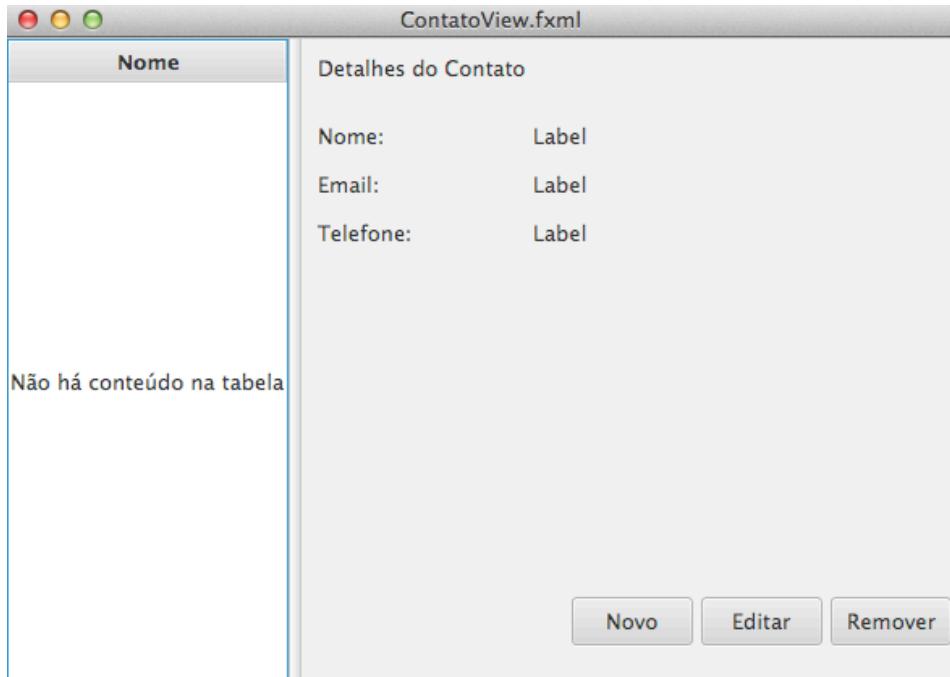
Adicione um GridPane e defina as seguintes propriedades

Criação do Layout Secundário

Nome	Detalhes do Contato
	Nome: Label Email: Label Telefone: Label
Não há conteúdo na tabela	
<button>Novo</button> <button>Editar</button> <button>Remover</button>	

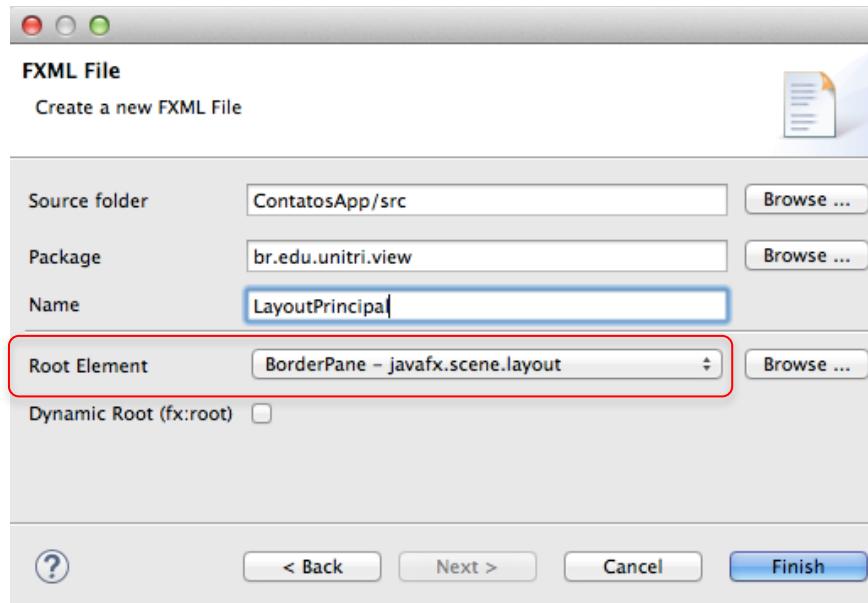
Adicione os seguintes componentes (labels e botões) ao layout

Criação do Layout Secundário



Preview do layout criado

Criação do Layout Principal

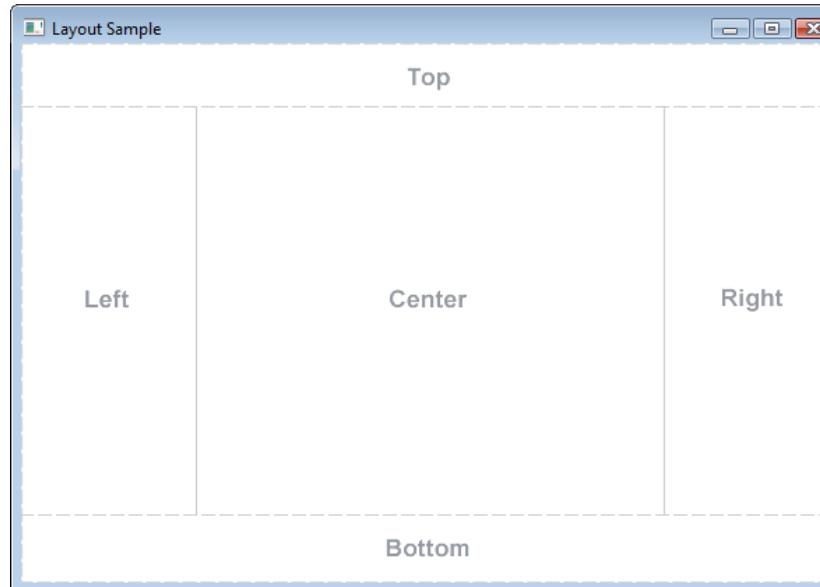


Defina o elemento raiz
(tipo de layout) como *BorderPane*

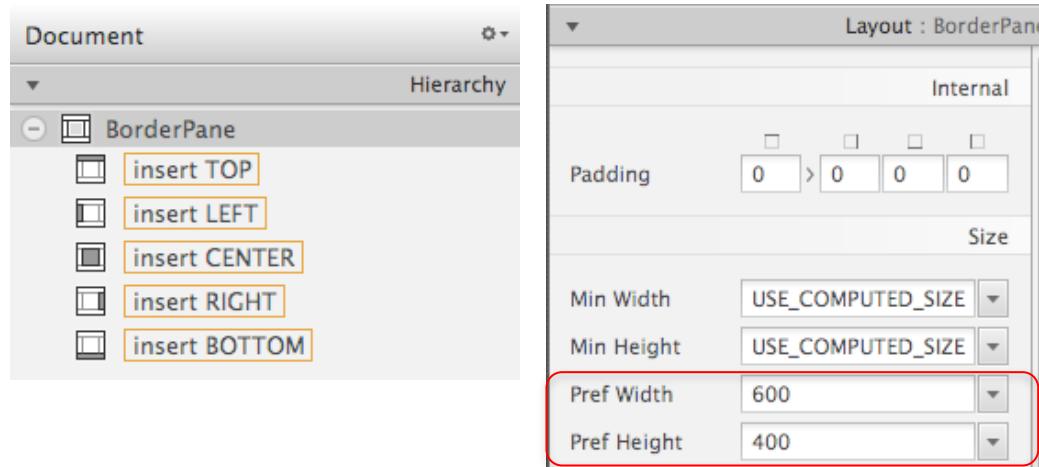
Crie um novo arquivo FXML dentro do pacote view
New > Other > JavaFX > New FXML Document

BorderPane

- O tipo de layout *BorderPane* fornece cinco regiões para inclusão de *nodes*.

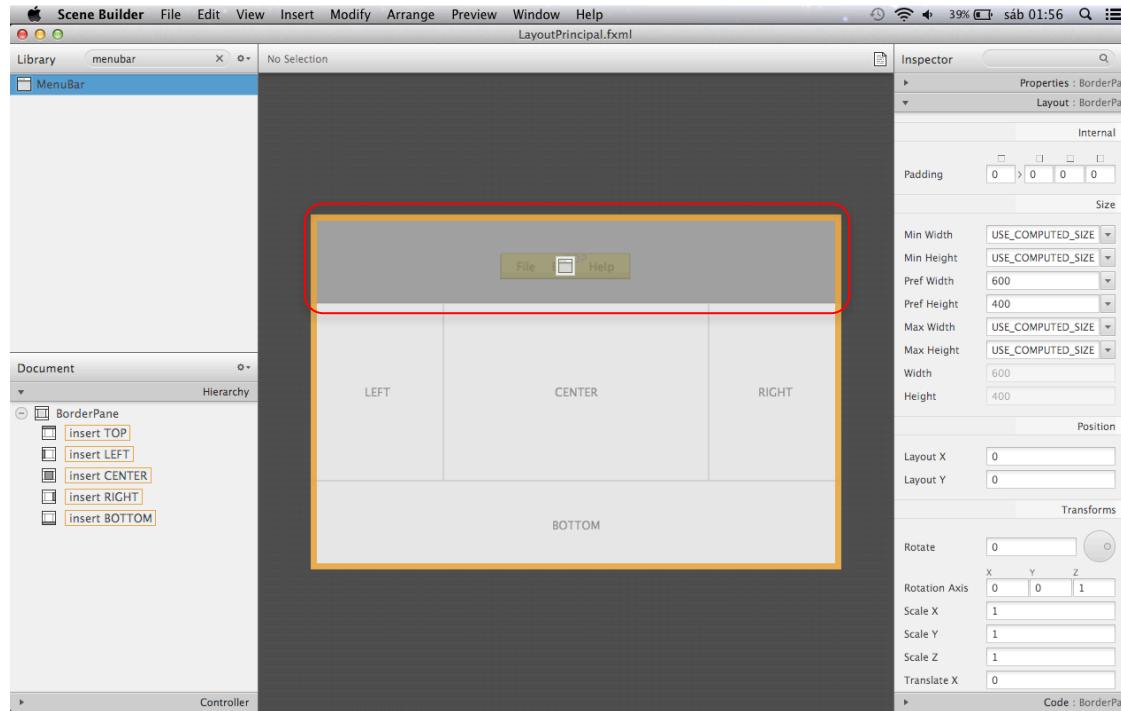


Criação do Layout Principal



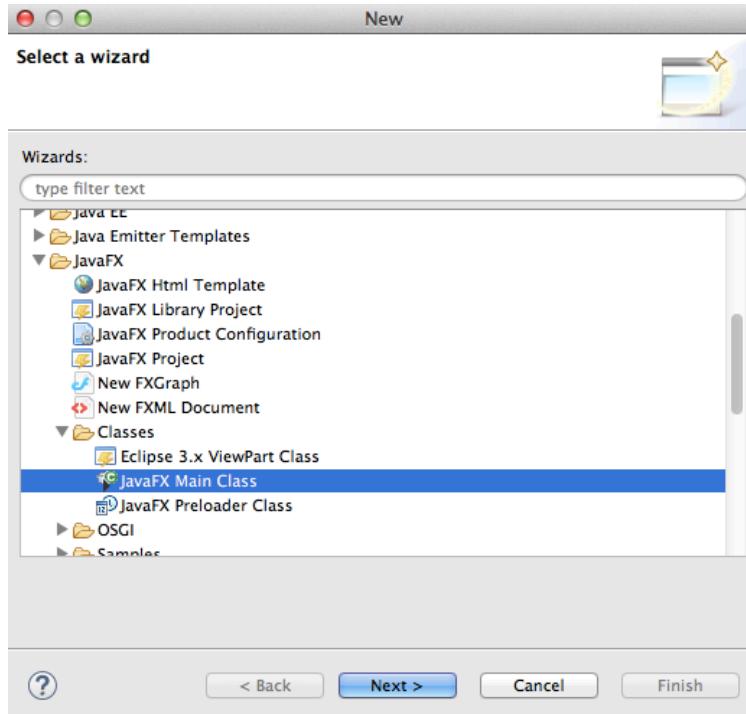
**Abra o arquivo LayoutPrincipal.fxml no Scene Builder e
defina a dimensões do BorderPane**

Criação do Layout Principal



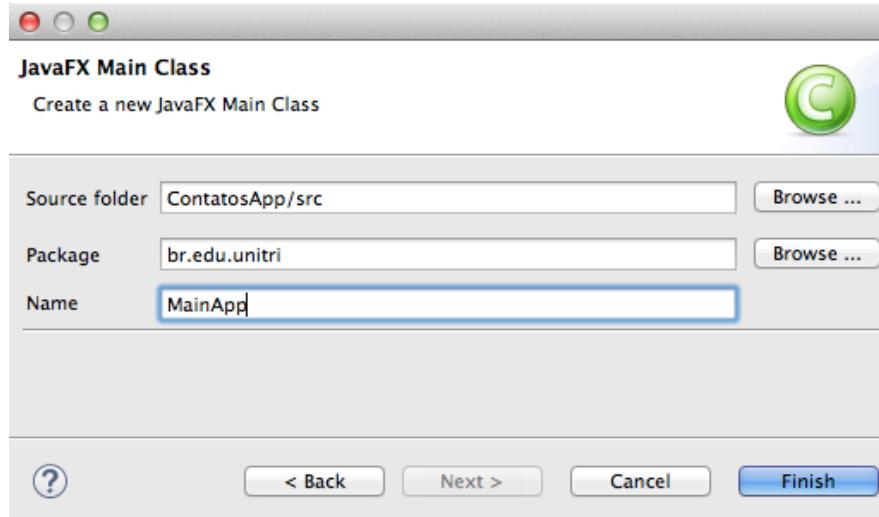
Adicione umMenuBar no topo do BorderPane

Criação do Classe Principal



**Crie uma nova classe Java clicando com o botão direito sobre src:
New > Other > JavaFX > Classes > JavaFX Main Class**

Criação do Classe Principal



Defina o nome e o pacote da classe

Criação do Classe Principal

```
package br.edu.unitri;

import javafx.application.Application;
import javafx.stage.Stage;

public class MainApp extends Application {

    @Override
    public void start(Stage primaryStage) {

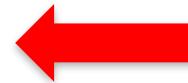
    }

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

Estrutura da classe gerada pelo Eclipse

Criação do Classe Principal

```
public class MainApp extends Application {  
  
    private Stage primaryStage;  
    private BorderPane rootLayout;  
  
    @Override  
    public void start(Stage primaryStage) {  
  
        this.primaryStage = primaryStage;  
        iniciarLayoutPrincipal();  
        exibirContatoView();  
    }  
}
```



Adicione a seguintes implementações na classe MainApp

Criação do Classe Principal

```
public void iniciarLayoutPrincipal() {  
    try {  
        FXMLLoader loader = new FXMLLoader();  
        loader.setLocation(MainApp.class.getResource("view/LayoutPrincipal.fxml"));  
        rootLayout = (BorderPane) loader.load();  
        Scene scene = new Scene(rootLayout);  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

Implemente o método *iniciarLayoutPrincipal()* logo abaixo do método *start(primaryStage)*



Este método será responsável por carregar o layout principal da aplicação e exibir a cena que contém este layout.

Criação do Classe Principal

```
public void exibirContatoView() {  
    try {  
        FXMLLoader loader = new FXMLLoader();  
        loader.setLocation(MainApp.class.getResource("view/ContatoView.fxml"));  
        AnchorPane contatoView = (AnchorPane) loader.load();  
        rootLayout.setCenter(contatoView);  
        ContatoViewController controller = loader.getController();  
        controller.setMainApp(this);  
  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

Na sequência, implemente o método `exibirContatoView()`



Este método será responsável por carregar o layout do ContatoView e apresentá-lo no centro do layout principal.

Criação do Classe Principal

```
public Stage getPrimaryStage() {  
    return primaryStage;  
}
```

Por fim, implemente o método `getPrimaryStage()`



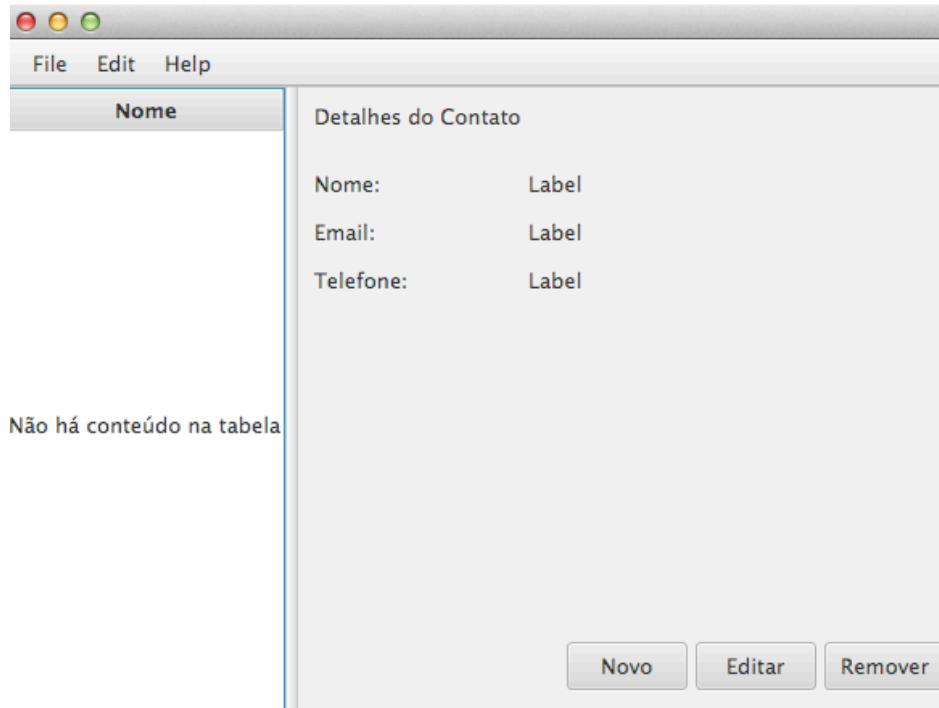
Este método será responsável por retornar o palco principal da aplicação gerenciado pela classe MainApp.

Criação do Classe Principal

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

O método **main()** deverá encerrar a classe **MainApp**

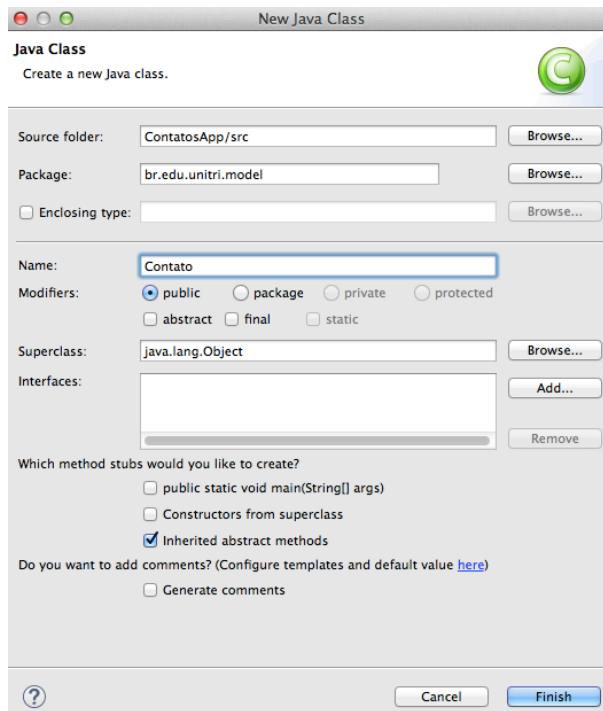
Execução do Projeto



Segunda Parte

Definição do Modelo

Criação da Classe do Modelo



Crie uma nova classe chamada Contato dentro do pacote model

Criação da Classe do Modelo

```
package br.edu.unitri.model;  
|  
import javafx.beans.property.StringProperty;  
  
public class Contato {  
  
    private final StringProperty nome;  
    private final StringProperty email;  
    private final StringProperty telefone;  
  
}
```

Implemente os seguintes atributos no corpo da classe Contato



Nas implementações JavaFX é comum utilizarmos o tipo [Properties](#) para os atributos da classe do modelo. Um *Property* implementa a notificação automática de qualquer alteração de estado do atributo.

Criação da Classe do Modelo

```
public Contato() {  
    this(null, null, null);  
}  
  
public Contato(String nome, String email, String telefone) {  
  
    this.nome = new SimpleStringProperty(nome);  
    this.email = new SimpleStringProperty(email);  
    this.telefone = new SimpleStringProperty(telefone);  
}
```

Adicione os seguintes Contrutores na classe Contato.

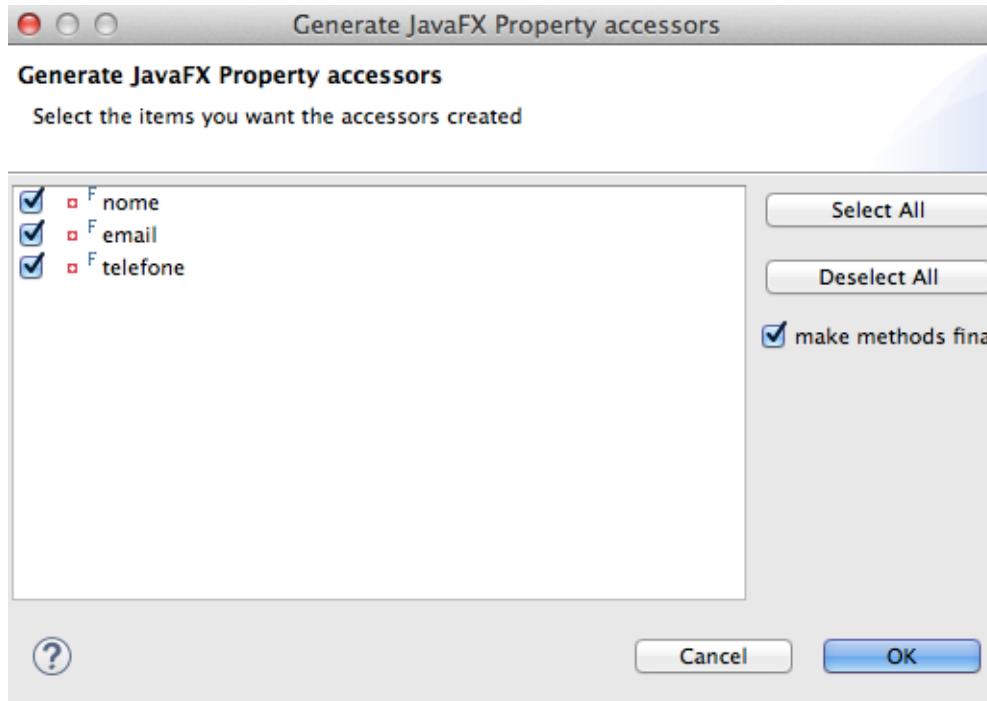
Criação da Classe do Modelo

```
public class Contato {  
  
    private final StringProperty nome;  
    private final StringProperty email;  
    private final StringProperty telefone;  
  
    public Contato(String nome, String email, String telefone) {  
  
        this.nome = new SimpleStringProperty(nome);  
        this.email = new SimpleStringProperty(email);  
        this.telefone = new SimpleStringProperty(telefone);  
    }  
}
```



Adicione os métodos Getters e Setters na classe Contato
Clique com o botão direito no trecho destacado
Source > Generate JavaFX Getters and Setters

Criação da Classe do Modelo



Seleciona todos os itens

Criação da Classe do Modelo

```
br.edu.unitri.model
▼ G Contato
  □ F nome : StringProperty
  □ F email : StringProperty
  □ F telefone : StringProperty
  ● C Contato(String, String, String)
  ● F nomeProperty() : StringProperty
  ● F getNome() : String
  ● F setNome(String) : void
  ● F emailProperty() : StringProperty
  ● F getEmail() : String
  ● F setEmail(String) : void
  ● F telefoneProperty() : StringProperty
  ● F getTelefone() : String
  ● F setTelefone(String) : void
```

Estrutura da classe Contato (*Outline*)

Lista de Contatos

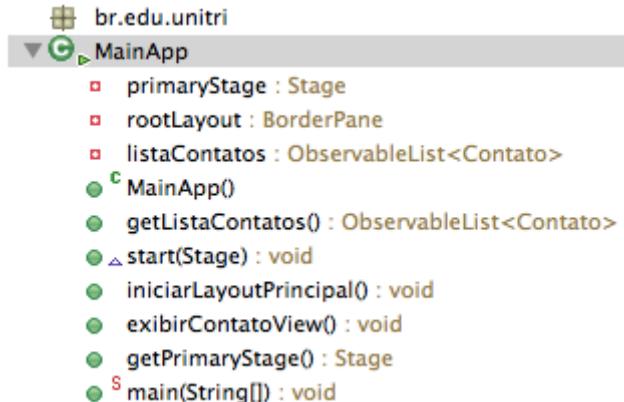
- Uma *Collection* do tipo *ObservableList* será utilizada para armazenar a lista de contatos da aplicação. O JavaFX introduziu no pacote *javafx.collections* alguns novos tipos de [Collections](#).
- Um *ObservableList* é um tipo de lista que implementa *listeners* para manipular as alterações de estado dos objetos quando elas ocorrerem.

Lista de Contatos

```
public class MainApp extends Application {  
  
    private Stage primaryStage;  
    private BorderPane rootLayout;  
    private ObservableList<Contato> listaContatos = FXCollections.observableArrayList();  
  
    public MainApp() {  
  
        listaContatos.add(new Contato("José da Silva", "jose@gmail.com", "(34)9999-9999"));  
        listaContatos.add(new Contato("Manoel Costinha", "manoel@gmail.com", "(34)9977-7777"));  
        listaContatos.add(new Contato("Maria Madalena", "maria@gmail.com", "(34)9966-6666"));  
    }  
  
    public ObservableList<Contato> getListaContatos() {  
        return listaContatos;  
    }  
}
```

Adicione a seguinte implementação na classe *MainApp* logo acima do método *start()*

Lista de Contatos

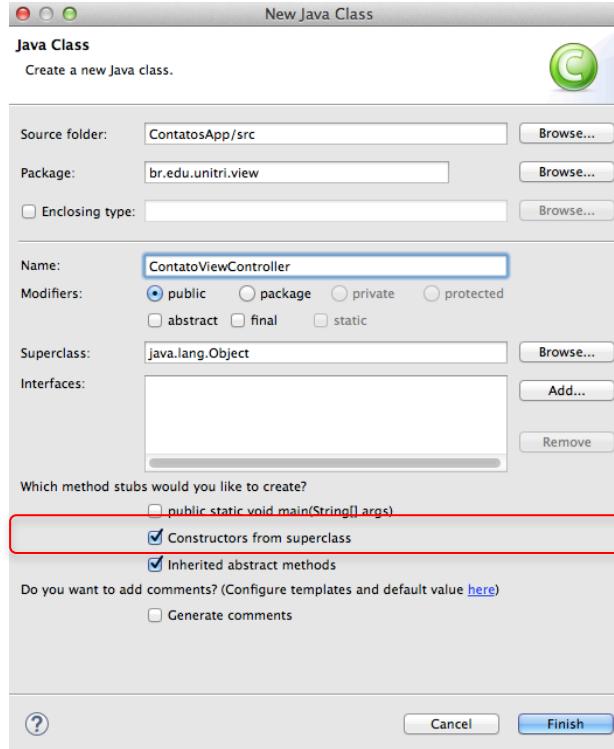


Estrutura da classe MainApp (*Outline*)

Controlador do ContatoView

- Para adicionarmos alguns valores no *TableView* utilizado no arquivo *ContatoView.fxml* criaremos uma classe que fará o papel de *view-controller* (*ContatoViewController.java*) e será responsável por gerenciar as interações realizadas com a visão.

Controlador do ContatoView



Crie uma nova classe chamada *ContatoViewController* dentro do pacote *view*

Controlador do ContatoView

```
public class ContatoViewController {  
  
    @FXML  
    private TableView<Contato> contatoTableView;  
    @FXML  
    private TableColumn<Contato, String> nomeTableColumn;  
    @FXML  
    private Label nomeLabel;  
    @FXML  
    private Label emailLabel;  
    @FXML  
    private Label telefoneLabel;  
  
    private MainApp mainApp;  
}
```

Adicione os seguintes atributos na classe *ContatoViewController*



Sempre utilize nos imports o pacote javafx, e **não** o pacote awt ou swing.

Controlador do ContatoView

```
@FXML  
private void initialize() {  
  
    nomeTableColumn.setCellValueFactory(cellData -> cellData.getValue().nomeProperty());  
}  
  
public void setMainApp(MainApp mainApp) {  
  
    this.mainApp = mainApp;  
    contatoTable.setItems(mainApp.getListaContatos());  
}
```

Implemente os seguintes métodos logo abaixo do construtor da classe *ContatoViewController*



O método *initialize()* é chamado automaticamente após o arquivo *FXML* ser carregado.

Controlador do ContatoView

```
br.edu.unitri.view
└── ContatoViewController
    ├── contatoTable : TableView<Contato>
    ├── nomeTableColumn : TableColumn<Contato, String>
    ├── nomeLabel : Label
    ├── emailLabel : Label
    ├── telefoneLabel : Label
    └── mainApp : MainApp
        ⏴ ContatoViewController()
        └── initialize() : void
        ⏴ setMainApp(MainApp) : void
```

Estrutura da classe ContatoViewController

Conectando a Classe Principal com o ContatoViewController

```
public void exibirContatoView() {
    try {
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(MainApp.class.getResource("view/ContatoView.fxml"));
        AnchorPane personOverview = (AnchorPane) loader.load();
        rootLayout.setCenter(personOverview);
        ContatoViewController controller = loader.getController();
        controller.setMainApp(this);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

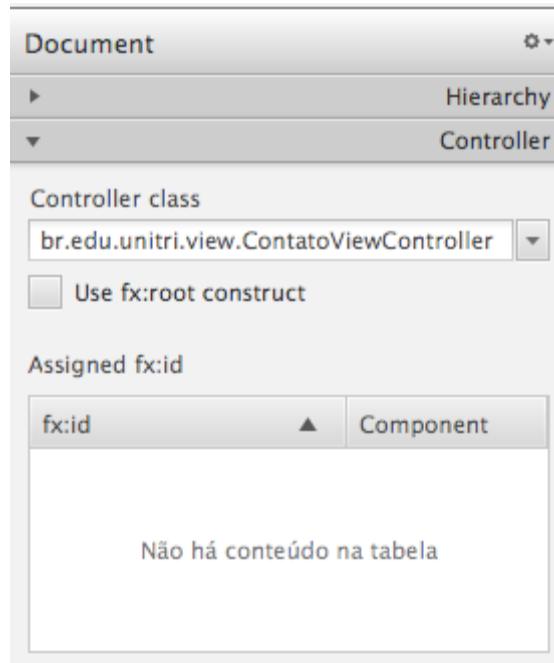
Implemente o seguinte trecho no método `exibirContatoView()` da classe `MainApp`



Com o trecho acima permitimos que o controlador tenha acesso às propriedades da classe MainApp. O método `getController()` define qual controlador está associado ao objeto raiz (`ContatoView.fxml`).

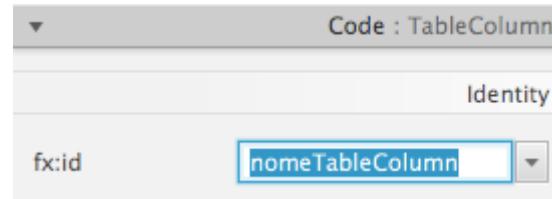
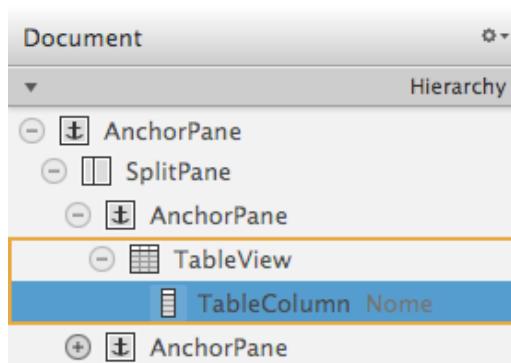
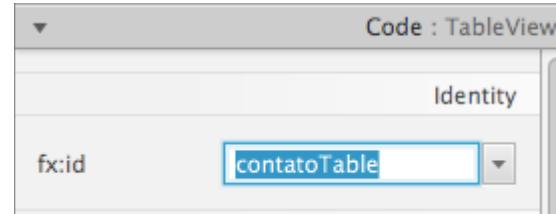
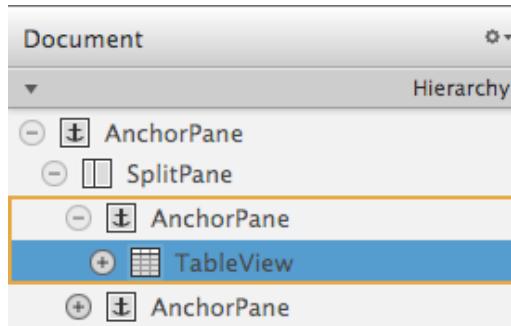
Mapeamento View x Controller

Abra o arquivo *ContatoView.fxml* com o Scene Builder e no painel Controller (à esquerda) defina a classe que implementará o controlador da visão



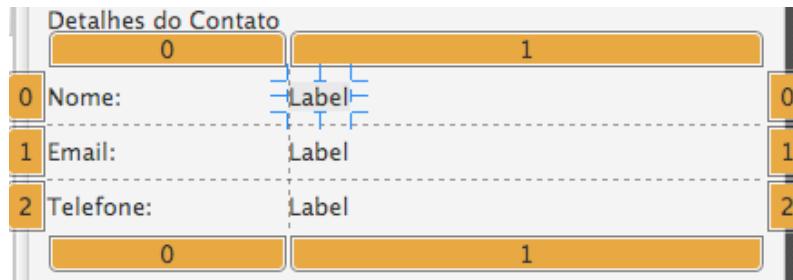
Mapeamento View x Controller

Defina o fx:id para os elementos TableView e TableColumn



Mapeamento View x Controller

Defina o fx:id para os labels adicionados

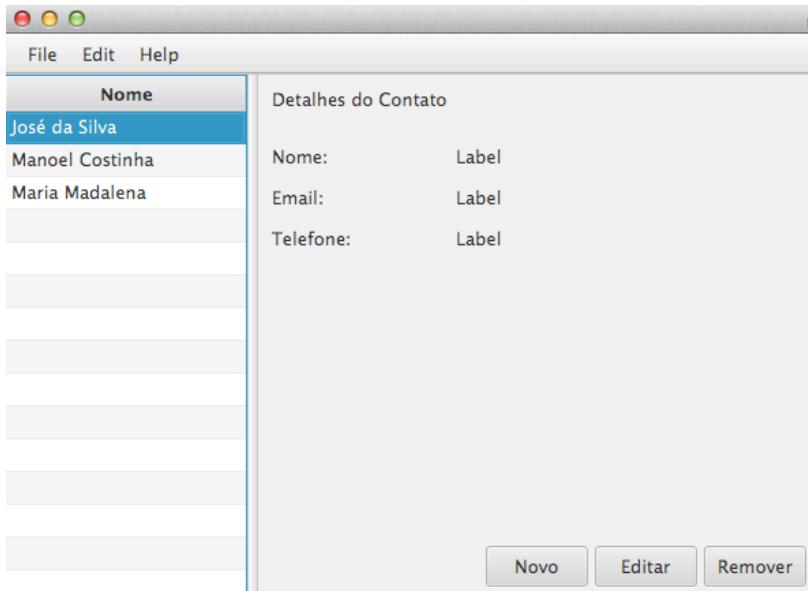


Code : Label	
Identity	
fx:id	nomeLabel

Code : Label	
Identity	
fx:id	emailLabel

Code : Label	
Identity	
fx:id	telefoneLabel

Executando a Aplicação



Antes de executar a aplicação, certifique-se de ter salvo os arquivos FXML no Scene Builder e dê um Refresh no projeto *ContatosApp*.

Terceira Parte

Tratativa de Eventos

Tratativa de Eventos

- Uma das funcionalidades principais da aplicação *ContatosApp* é apresentar os detalhes do usuário selecionado no *TableView* em um *GridPane*.
- Iremos implementar o método *exibirDetalhesContato()* na classe *ContatoViewController* para tratar o evento do clique em uma linha do *TableView*.

Tratativa de Eventos

```
private void exibirDetalhesContato(Contato contato) {  
    if (contato != null) {  
        nomeLabel.setText(contato.getNome());  
        emailLabel.setText(contato.getEmail());  
        telefoneLabel.setText(contato.getTelefone());  
    } else {  
        nomeLabel.setText("");  
        emailLabel.setText("");  
        telefoneLabel.setText("");  
    }  
}
```

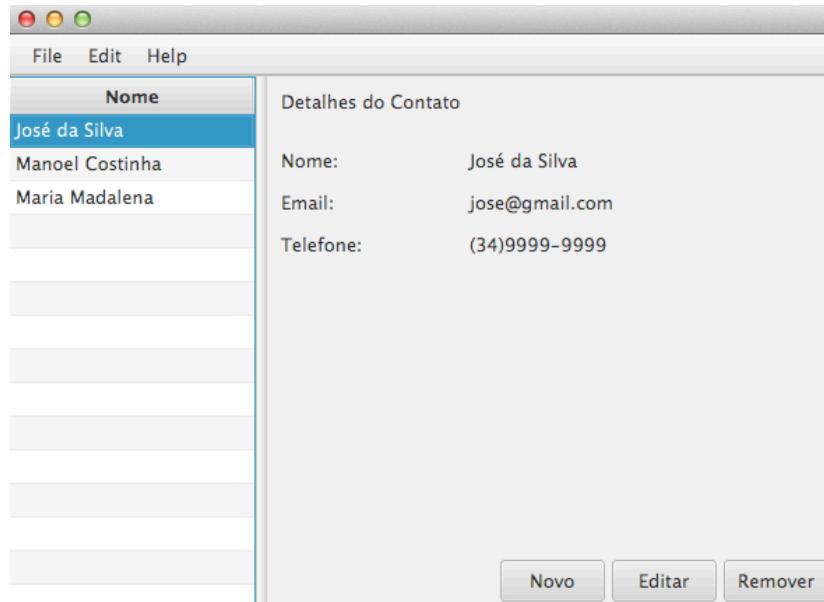
Implemente o método *exibirDetalhesContato()* na classe ContatoViewController
(logo abaixo do método *setMainApp*)

Tratativa de Eventos

```
@FXML  
private void initialize() {  
  
    nomeTableColumn.setCellValueFactory(cellData -> cellData.getValue().nomeProperty());  
  
    exibirDetalhesContato(null);  
  
    contatoTable.getSelectionModel().selectedItemProperty().addListener(  
        (observable, oldValue, newValue) -> exibirDetalhesContato(newValue));  
}  
  
Adicione o trecho destacado ao método initialize() da classe ContatoViewController
```

Tratativa de Eventos

- Executando a aplicação, os detalhes do contato selecionado no *TableView* serão apresentados nos labels do *GridPane*.



Tratativa de Eventos

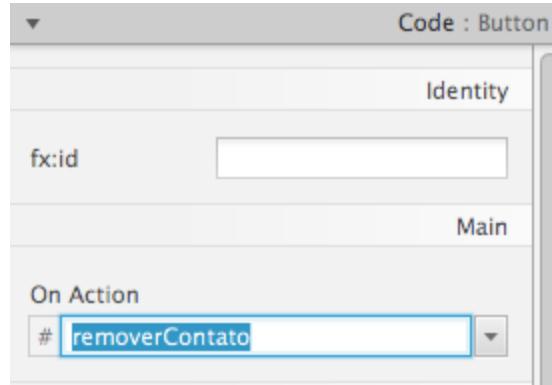
- Outra funcionalidade do *ContatosApp* será possibilitar ao usuário a remoção de um determinado contato.
- Para tal, implementaremos o método *removerContato()* na classe *ContatoViewController* que será chamado quando o usuário clicar no botão “Remover”.

Tratativa de Eventos

```
@FXML  
private void removerContato() {  
    int selectedIndex = contatoTable.getSelectionModel().getSelectedIndex();  
    contatoTable.getItems().remove(selectedIndex);  
}
```

Implemente o método *removerContato()* na classe **ContatoViewController**
(logo abaixo do método *exibirDetalhesContato()*)

Tratativa de Eventos



No arquivo *ContatoView.fxml* adicione no On Action do botão Remover
a referência ao método `removerContato()`

Tratativa de Eventos

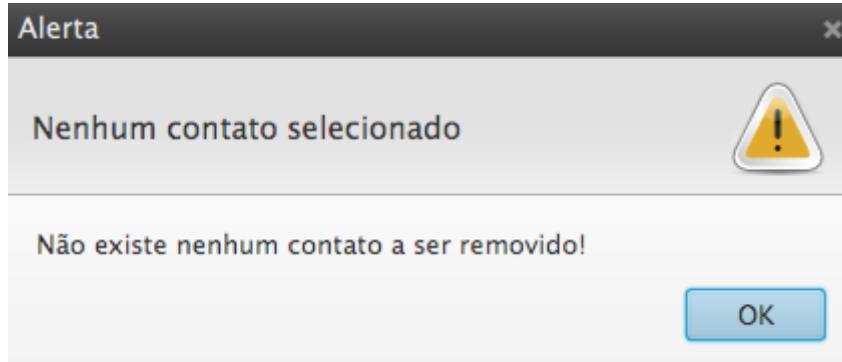
- Existe uma exceção do tipo *ArrayListOutOfBoundsException* não tratada no método *removerContato()*, pois não podemos remover um contato que possua index -1.

```
Caused by: java.lang.ArrayIndexOutOfBoundsException: -1
  at java.util.ArrayList.elementData(ArrayList.java:418)
  at java.util.ArrayList.remove(ArrayList.java:495)
  at com.sun.javafx.collections.ObservableListWrapper.doRemove(ObservableListWrapper.java:116)
  at javafx.collections.ModifiableObservableListBase.remove(ModifiableObservableListBase.java:179)
  at br.edu.unitri.view.ContatoViewController.removerContato(ContatoViewController.java:64)
  ... 57 more
```

Tratativa de Eventos

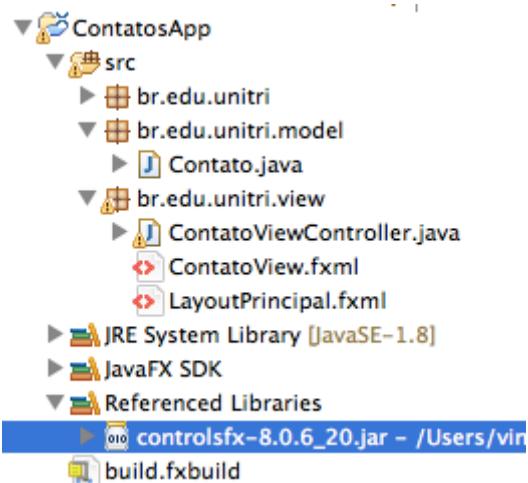
- Faremos a tratativa para os casos onde o index for igual a -1 e apresentaremos uma mensagem alertando ao usuário que tal evento ocorreu.
- Utilizaremos a biblioteca controlsfx-8.0.6_20.jar para apresentar uma mensagem amigável ao usuário.
 - A biblioteca [controlsfx](#) é um projeto *open source* que tem como objetivo disponibilizar ricos componentes visuais implementados para o JavaFX .

Tratativa de Eventos



Exemplo de dialog exibido com a biblioteca *controlsfx*

Adicionando a biblioteca



Clique com o botão direito sobre o projeto **ContatosApp**:
Build Path > Add External Archives... > localize o arquivo **controlsfx-8.0.6_20.jar**

Tratativa de Eventos

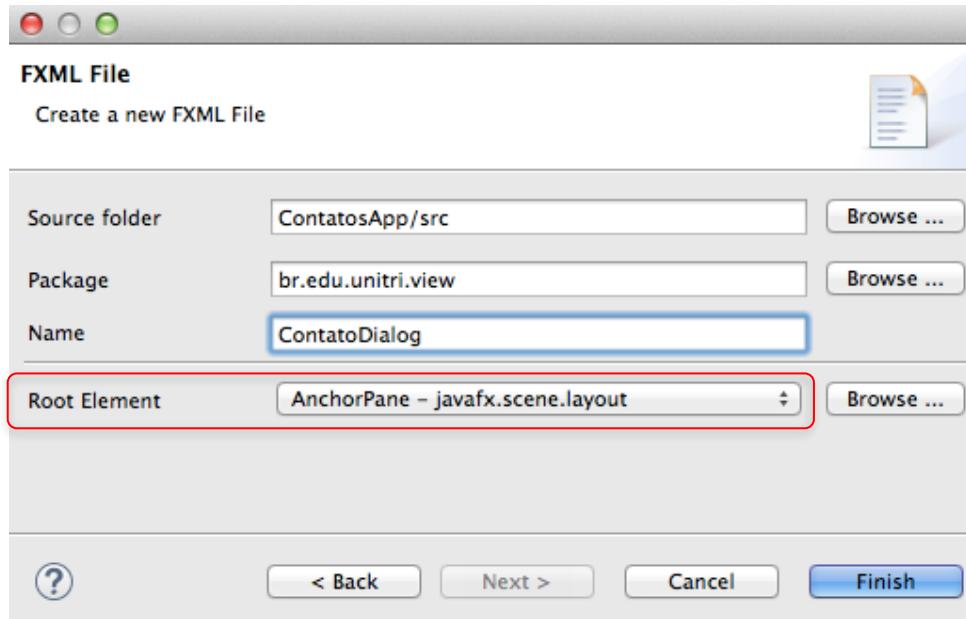
```
@FXML  
private void removerContato() {  
  
    int selectedIndex = contatoTable.getSelectionModel().getSelectedIndex();  
  
    if (selectedIndex >= 0) {  
        contatoTable.getItems().remove(selectedIndex);  
    } else {  
        Dialogs.create()  
            .title("Alerta")  
            .masthead("Nenhum contato selecionado")  
            .message("Não existe nenhum contato a ser removido!")  
            .showWarning();  
    }  
}
```

Atualize a implementação do método *removerContato()* na classe *ContatoViewController*

Tratativa de Eventos

- Além de remover um determinado contato, a aplicação *ContatosApp* possibilitará a inclusão de um novo contato e a edição de um contato existente no *TableView*.
- Vamos adicionar um novo arquivo FXML que será responsável pelo layout do *dialog* a ser utilizado para a inclusão e edição de um contato.

Tratativa de Eventos



Defina o elemento raiz (tipo de layout) como *AnchorPane*

Crie um novo arquivo FXML dentro do pacote *view*
New > Other > JavaFX > New FXML Document

Tratativa de Eventos

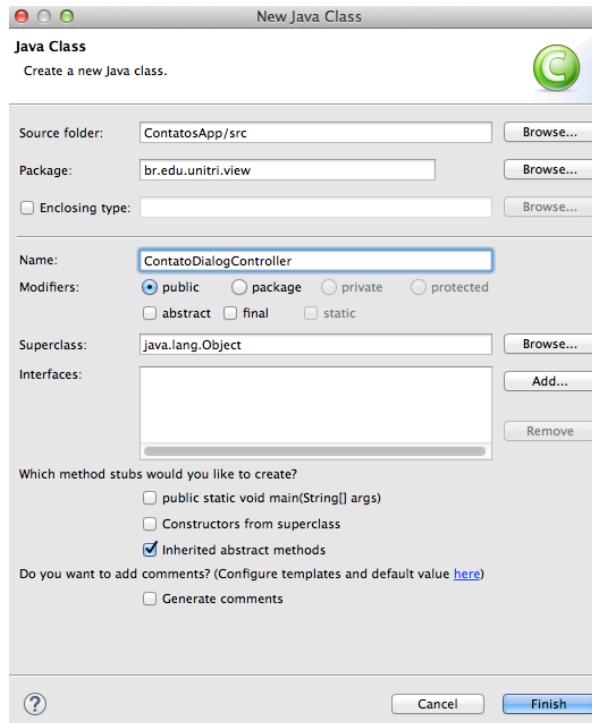


Adicione o seguintes elementos ao arquivo ContatoDialog.fxml

Tratativa de Eventos

- Criaremos uma classe view-controller para tratar o comportamento do layout definido no arquivo *ContatoDialog.fxml*.

Tratativa de Eventos



Crie uma nova classe chamada *ContatoDialogController* dentro do pacote *view*

Tratativa de Eventos

```
@FXML  
private TextField nomeTextField;  
@FXML  
private TextField emailTextField;  
@FXML  
private TextField telefoneTextField;  
  
private Stage dialogStage;  
private Contato contato;  
private boolean okClicked = false;
```

Adicione os atributos acima no corpo da classe **ContatoDialogController**

Tratativa de Eventos

```
@FXML  
private void initialize() {  
}  
  
public void setDialogStage(Stage dialogStage) {  
    this.dialogStage = dialogStage;  
}  
  
public boolean isOkClicked() {  
    return okClicked;  
}
```

Adicione os métodos acima na classe ContatoDialogController
(logo abaixo aos atributos)

Tratativa de Eventos

```
public void atualizarContato(Contato contato) {  
  
    this.contato = contato;  
    nomeTextField.setText(contato.getNome());  
    emailTextField.setText(contato.getEmail());  
    telefoneTextField.setText(contato.getTelefone());  
}
```

Adicione o método `atualizarContato()` na classe `ContatoDialogController`
(logo abaixo do método `setDialogStage()`)

Tratativa de Eventos

```
private boolean validarCampos() {  
  
    String errorMessage = "";  
    if (nomeTextField.getText() == null || nomeTextField.getText().length() == 0) {  
        errorMessage += "Nome inválido!\n";  
    }  
    if (emailTextField.getText() == null || emailTextField.getText().length() == 0) {  
        errorMessage += "Email inválido!\n";  
    }  
    if (telefoneTextField.getText() == null || telefoneTextField.getText().length() == 0) {  
        errorMessage += "Telefone inválido!\n";  
    }  
    if (errorMessage.length() == 0) {  
        return true;  
    } else {  
        Dialogs.create()  
            .title("Alerta")  
            .masthead("Existem campos inválidos no formulário")  
            .message(errorMessage)  
            .showError();  
        return false;  
    }  
}
```

Tratativa de Eventos

```
@FXML  
private void salvarContato() {  
  
    if (validarCampos()) {  
  
        contato.setNome(nomeTextField.getText());  
        contato.setEmail(emailTextField.getText());  
        contato.setTelefone(telefoneTextField.getText());  
        okClicked = true;  
        dialogStage.close();  
    }  
}
```

Adicione o método `salvarContato()` na classe `ContatoDialogController`
(logo abaixo do método `validarCampos()`)

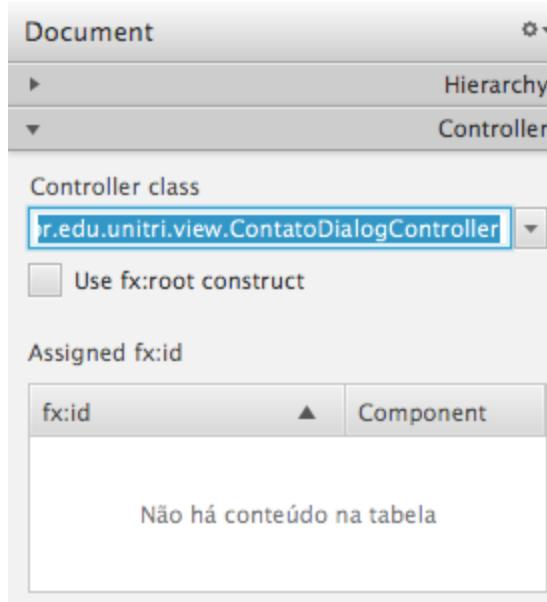
Tratativa de Eventos

```
@FXML  
private void cancelar() {  
  
    dialogStage.close();  
}
```

Adicione o método cancelar() na classe ContatoDialogController
(logo abaixo do método salvarContato())

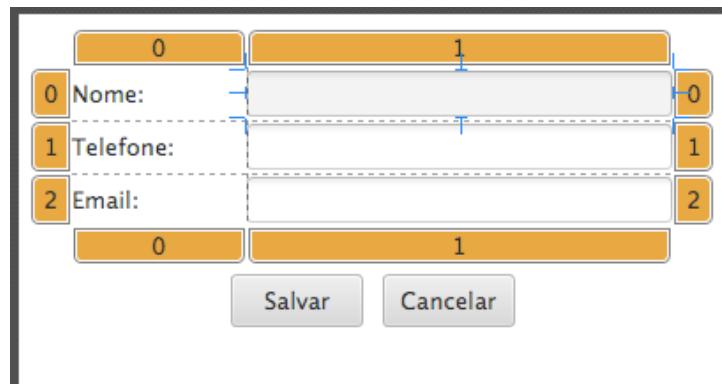
Mapeamento View x Controller

Abra o arquivo *ContatoDialog.fxml* com o Scene Builder e no painel Controller (à esquerda) defina a classe que implementará o controlador da visão



Mapeamento View x Controller

Defina o fx:id para os textfields adicionados



Code : TextField	
Identity	
fx:id	nomeTextField

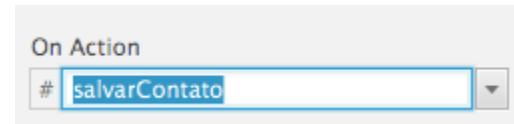
Code : TextField	
Identity	
fx:id	telefoneTextField

Code : TextField	
Identity	
fx:id	emailTextField

Mapeamento View x Controller

Defina o método que tratará ao evento On Action

A screenshot of a contact form view. It contains three input fields labeled 'Nome', 'Telefone', and 'Email'. Below the fields are two buttons: 'Salvar' (Save) on the left and 'Cancelar' (Cancel) on the right. The entire form is enclosed in a brown border.



A screenshot of the same contact form view as the first one, but with the buttons swapped. The 'Cancelar' button is now on the left and the 'Salvar' button is on the right. The entire form is enclosed in a brown border.



Tratativa de Eventos

- Teremos que implementar o método *exibirContatoDialog()* na classe *MainApp* para apresentar o *dialog* definido no arquivo *ContatoDialog.fxml*.

```
public boolean exibirContatoDialog(Contato contato) {  
    try {  
        FXMLLoader loader = new FXMLLoader();  
        loader.setLocation(MainApp.class.getResource("view/ContatoDialog.fxml"));  
        AnchorPane page = (AnchorPane) loader.load();  
  
        Stage dialogStage = new Stage();  
        dialogStage.initModality(Modality.WINDOW_MODAL);  
        dialogStage.initOwner(primaryStage);  
        Scene scene = new Scene(page);  
        dialogStage.setScene(scene);  
  
        ContatoDialogController controller = loader.getController();  
        controller.setDialogStage(dialogStage);  
        controller.atualizarContato(contato);  
        dialogStage.showAndWait();  
        return controller.isOkClicked();  
    } catch (IOException e) {  
        e.printStackTrace();  
        return false;  
    }  
}
```

Tratativa de Eventos

```
@FXML  
private void abrirNovoContato() {  
    Contato tempContato = new Contato();  
    boolean okClicked = mainApp.exibirContatoDialog(tempContato);  
    if (okClicked) {  
        mainApp.getListaContatos().add(tempContato);  
    }  
}
```

Adicione o método `abrirNovoContato()` na classe `ContatoViewController`
(logo abaixo do método `removerContato()`)

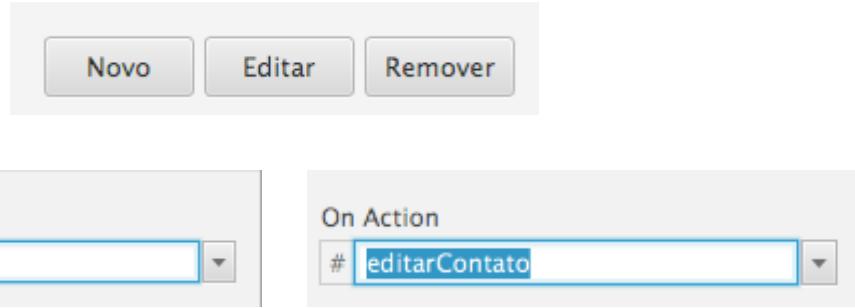
Tratativa de Eventos

```
@FXML  
private void editarContato() {  
    Contato contatoSelecionado = contatoTable.getSelectionModel().getSelectedItem();  
    if (contatoSelecionado != null) {  
        boolean okClicked = mainApp.exibirContatoDialog(contatoSelecionado);  
        if (okClicked) {  
            exibirDetalhesContato(contatoSelecionado);  
        }  
  
    } else {  
        Dialogs.create()  
            .title("Alerta")  
            .masthead("Nenhum contato selecionado")  
            .message("Não existe nenhum contato a ser removido!")  
            .showWarning();  
    }  
}
```

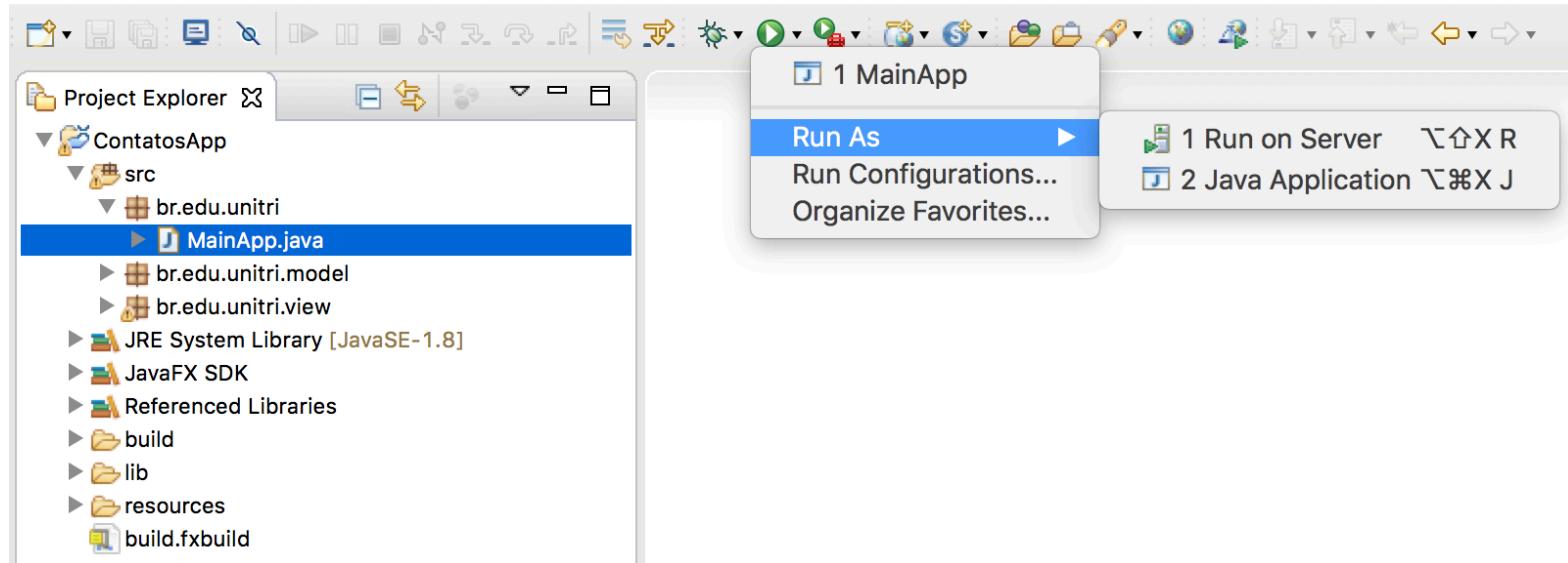
**Adicione o método editarContato() na classe ContatoViewController
(logo abaixo do método abrirNovoContato())**

Mapeamento View x Controller

Abra o arquivo *ContatoView.fxml* com o Scene Builder e defina o evento On Action para os botões Novo e Editar



Executando a Aplicação



Selecione a classe MainApp.java > Run As > Java Application



Referências

- [1] JavaFX 8; Introduction by Example 2a ed; Carl Dea, Mark Heckler, Gerrit Grunwald, José Pereda, Sean Phillips; Apress
- [2] Pro JavaFX 8: A Definitive Guide to Building Desktop, Mobile, and Embedded; James Weaver, Weiqi Gao, Stephen Chin, Dean Iverson, Johan Vos; Apress