

JavaFx

O que é o JavaFx

JavaFX é uma plataforma de software multimídia desenvolvida pela Sun Microsystems baseada em Java para a criação e disponibilização de Aplicação Rica para Internet que pode ser executada em vários dispositivos diferentes.

A versão atual (JavaFX 2.1.0) permite a criação para desktop, browser e dispositivos móveis. TVs, video-games, Blu-rays players e há planos de adicionar novas plataformas no futuro. O suporte nos desktops e browsers é através da JRE e nos dispositivos móveis através do JavaME.

Para construir aplicações os desenvolvedores usam uma linguagem estática tipada e declarada chamada JavaFX Script. No desktop existe implementação para Windows(x86/x64), Mac OS X e Linux (X86/X64). Nos dispositivos móveis, JavaFx é capaz de suportar vários sistemas operativos moveis como Android, Windows Mobile, e outros sistemas proprietários.

Instalação do JavaFx

É necessária a versão 11 ou superior do JDK instalada para que seja possível o uso do JavaFx. Você pode encontrar a última versão do JDK disponível para download em:

<https://jdk.java.net/>

<https://www.oracle.com/java/technologies/javase-downloads.html>

Após instalar o JDK requerido, faça o download do JavaFx em:

<https://gluonhq.com/products/javafx/>

Extraia o arquivo baixado no diretório de sua preferência e pronto. Agora você está pronto para começar a usar o JavaFx em seus projetos.

Criando o primeiro projeto JavaFx com IntelliJ

Abra a IDE IntelliJ (você pode baixa-la [aqui](#))



Figura1: Tela inicial do IntelliJ

Crie um novo projeto Java. Assegure-se de estar criando um projeto Java e não JavaFx uma vez que este não está configurado.

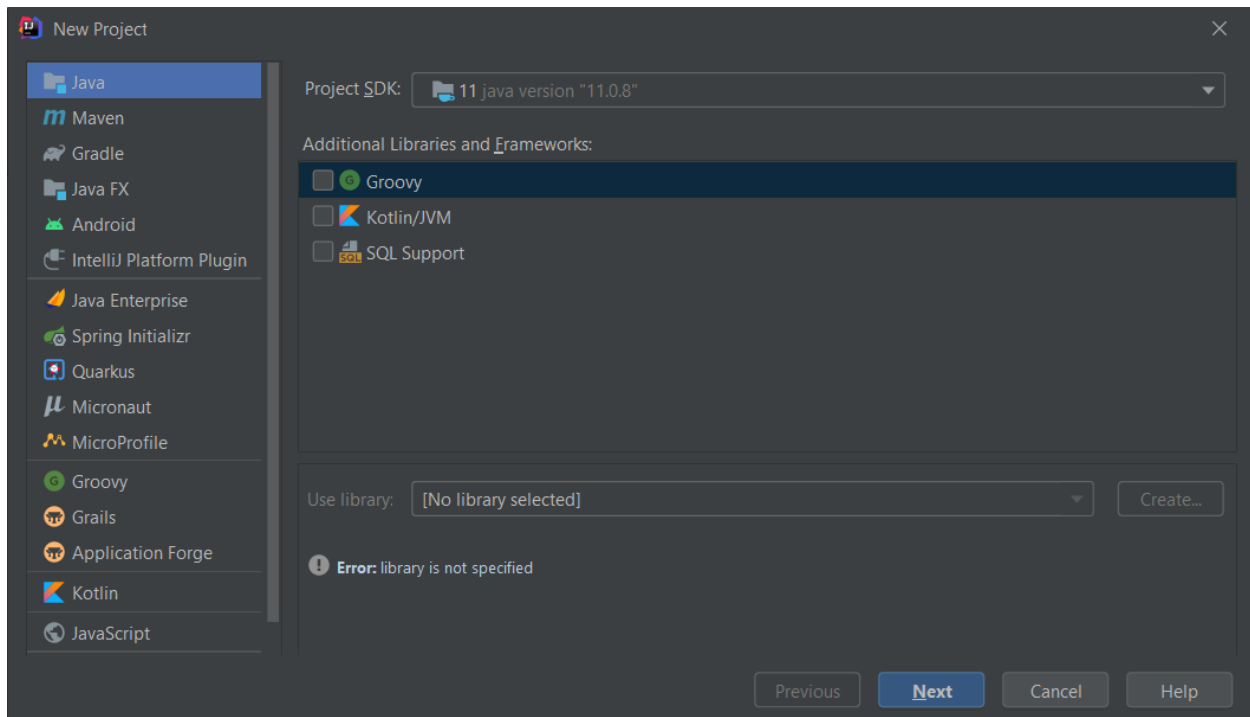


Figura 2: Criando um novo projeto Java.

Clique em next

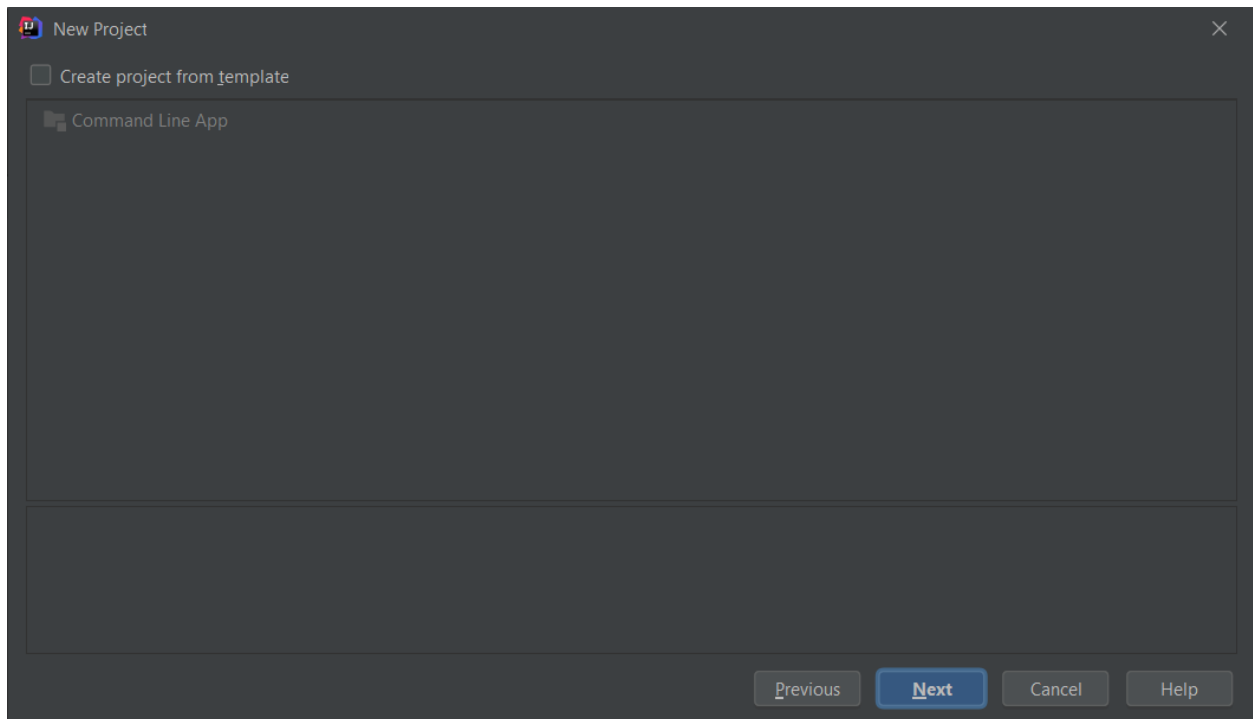


Figura 3: Criando um novo projeto Java passo 2.

Clique em next, defina o nome de seu projeto e o diretório onde ser criado. Clique em finish.

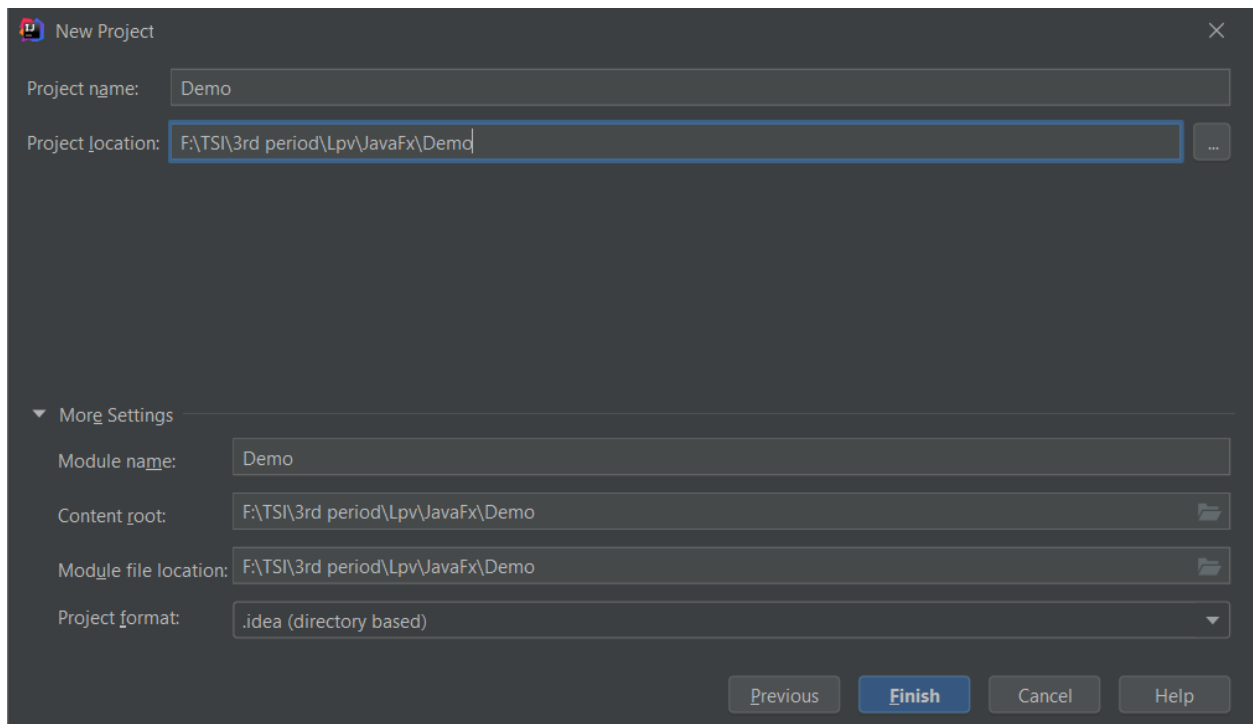


Figura 4: Criando um novo projeto Java – definindo nome e local do projeto.

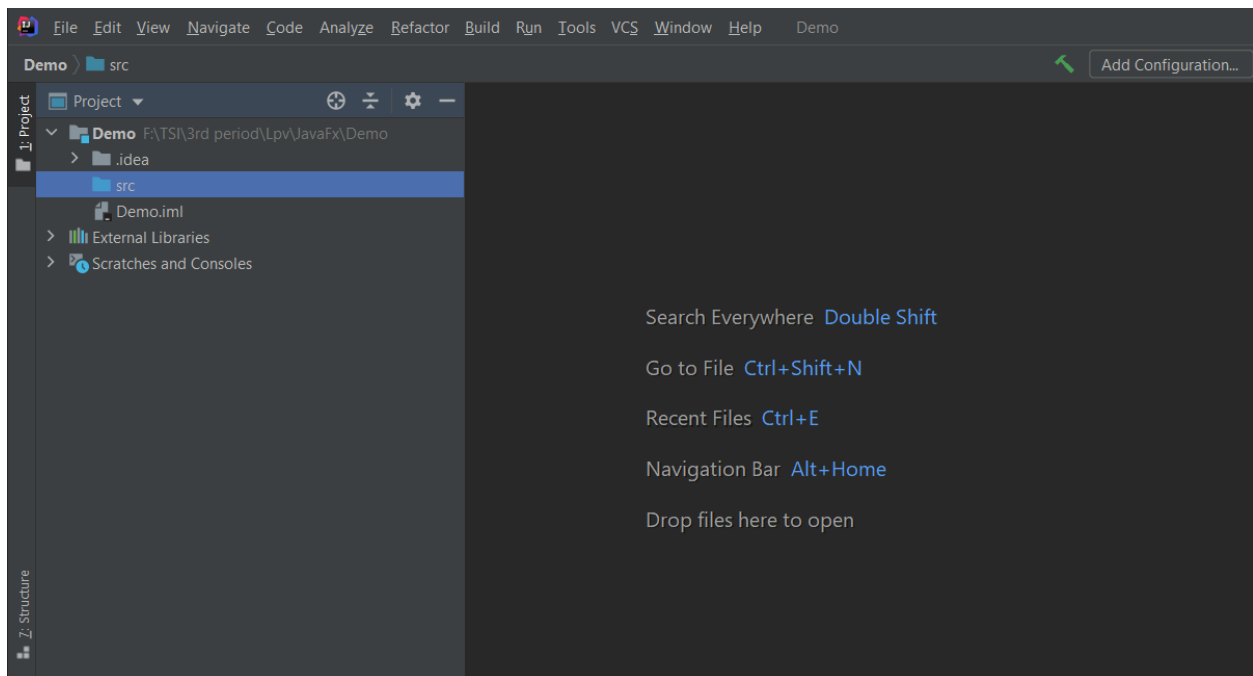


Figura 5: Tela inicial do projeto.

Com o projeto Java criado, é hora de incluir a biblioteca JavaFx. Para isso, vá em File -> Project Structure -> Libraries -> New Project Library -> Java

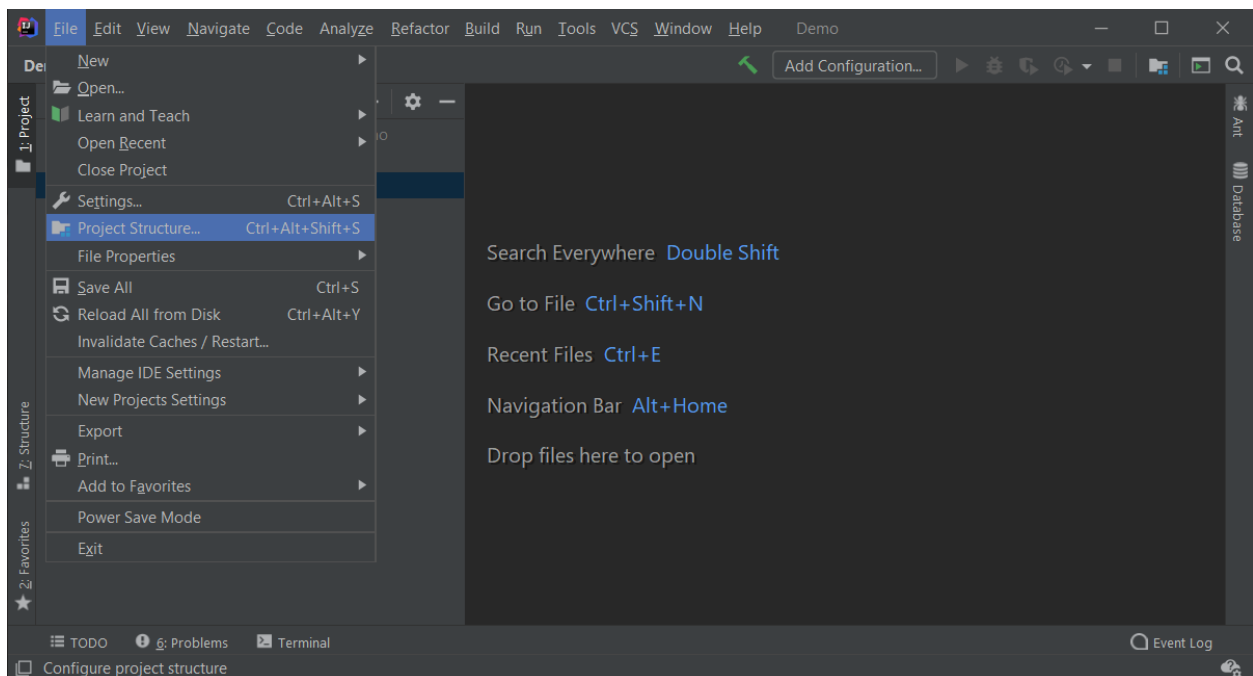


Figura 6: Abrindo estrutura do projeto.

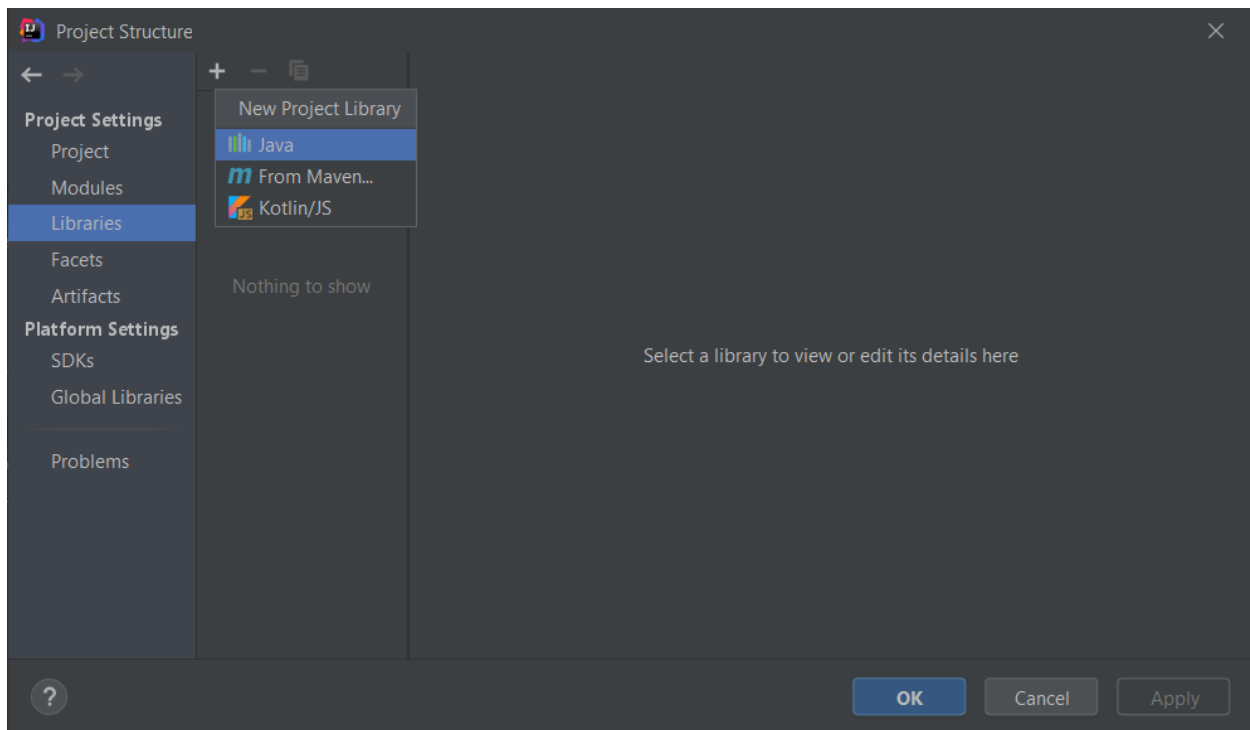


Figura 6: Adicionando biblioteca ao do projeto.

Navegue até o diretório onde o JavaFx foi extraído anteriormente, selecione todo o conteúdo que se encontra no subdiretório *lib* e clique em ok. Confirme se solicitado.

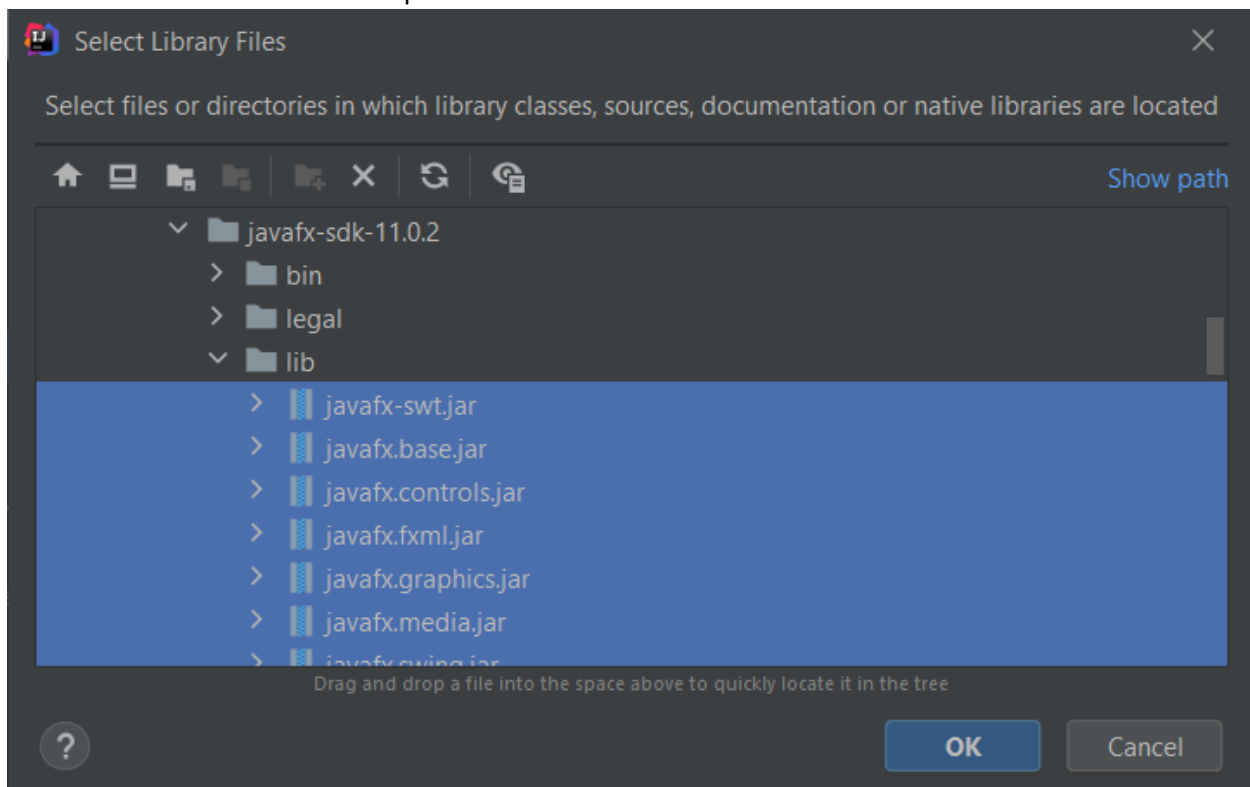


Figura 7: Selecionando as bibliotecas JavaFx para inclusão no projeto.

Aplique e clique em ok.

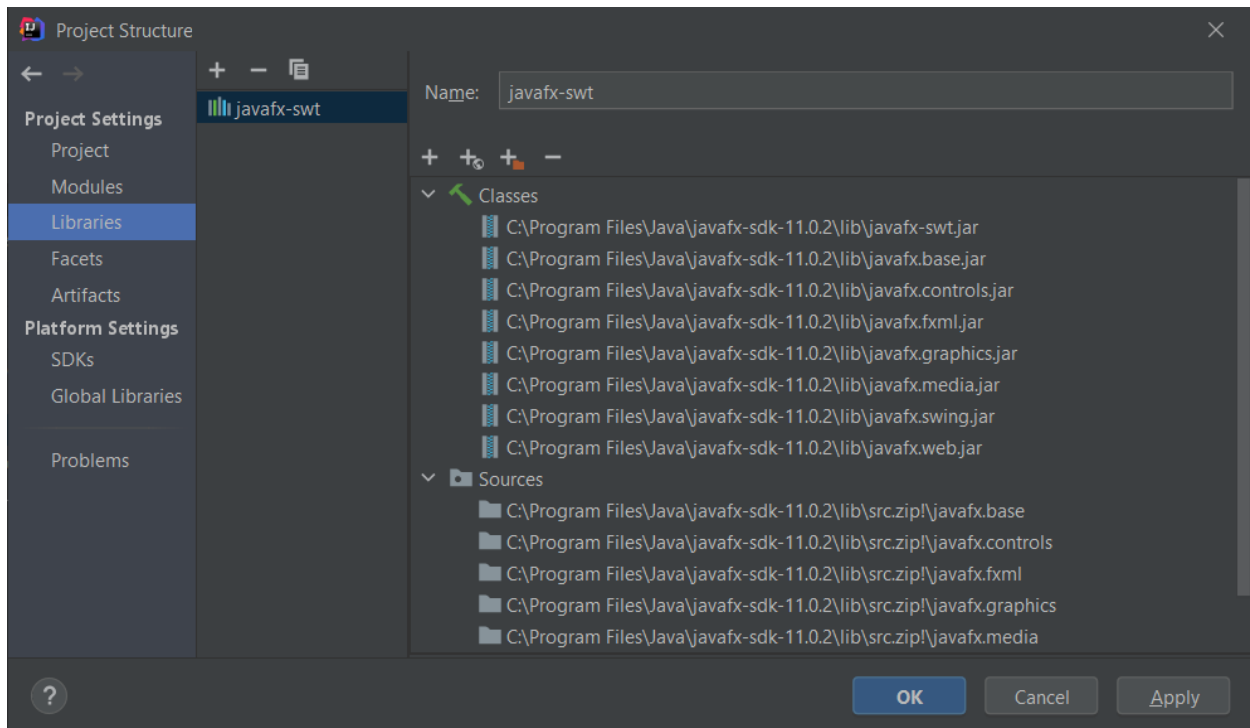


Figura 8: aplicando alterações.

De volta à tela inicial do projeto, podemos ver as bibliotecas do JavaFx agora incluídas.

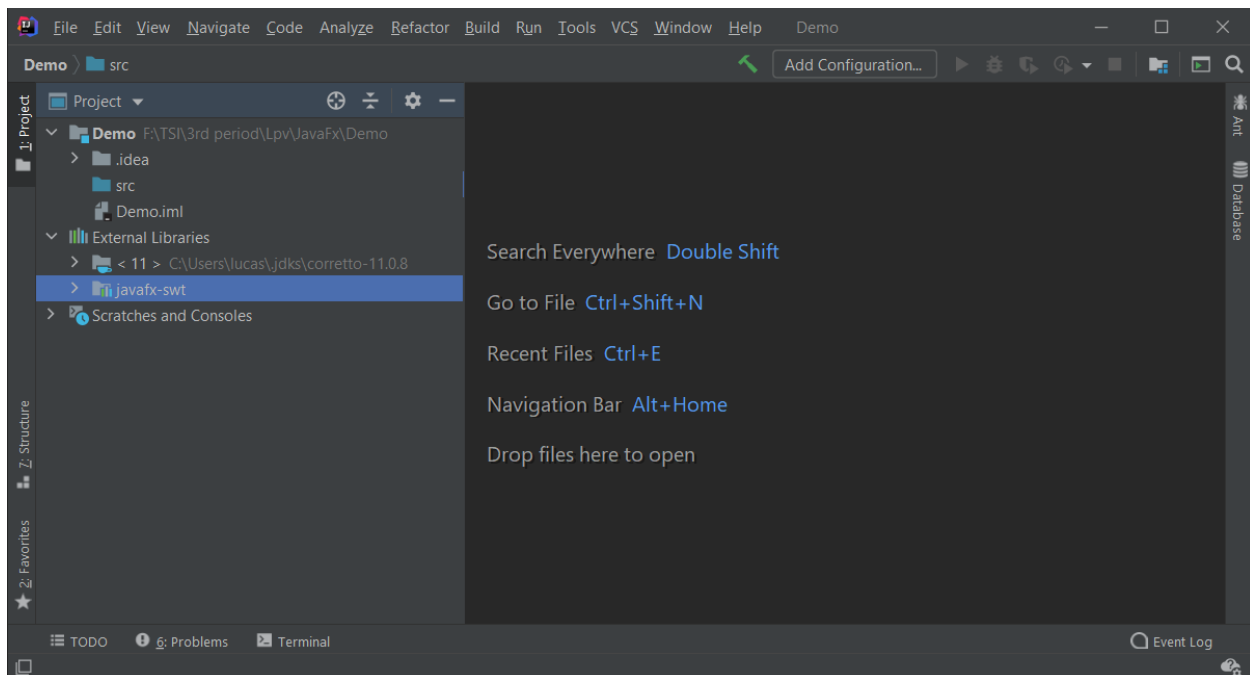


Figura 9: Bibliotecas JavaFx incluídas.

Criando a primeira View via código Java e configurando os módulos

Agora, o ambiente já se encontra configurado e pronto para iniciarmos a programação! Vamos criar e rodar nossa primeira tela. Defina o pacote raiz de seu projeto e nele crie uma sua classe de inicialização;

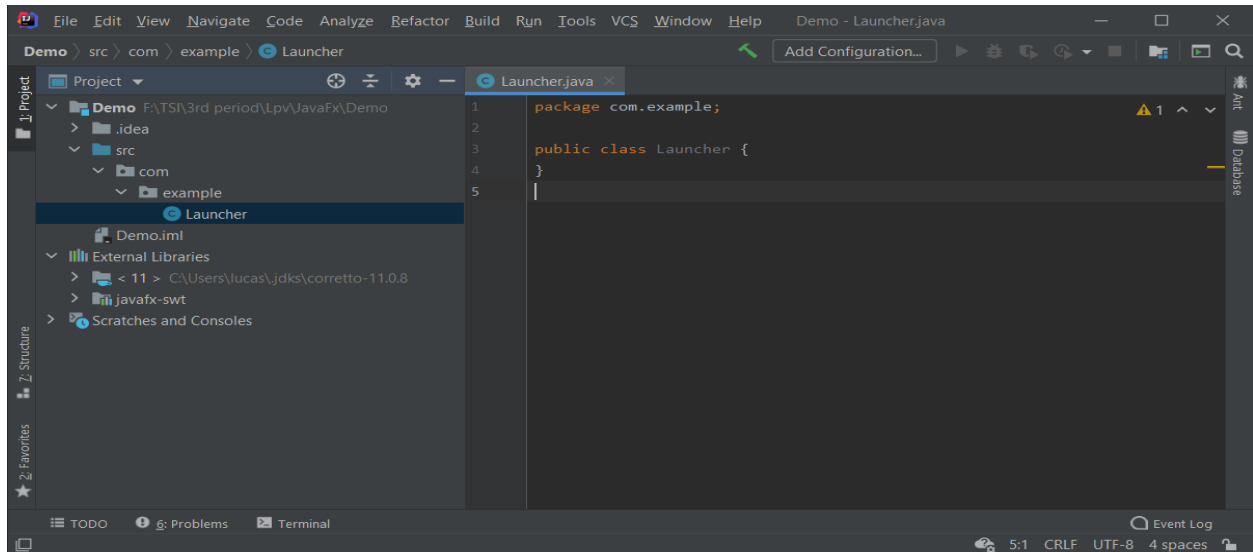


Figura 10: Pacote raiz e classe de inicialização criados.

Para que nosso aplicativo se inicie como um aplicativo JavaFx, precisamos dizer que a classe de lançamento é uma classe de aplicação, fazendo com que nossa classe de inicialização herde a classe Application. Precisamos também do método main que é o ponto inicial de qualquer aplicativo Java.

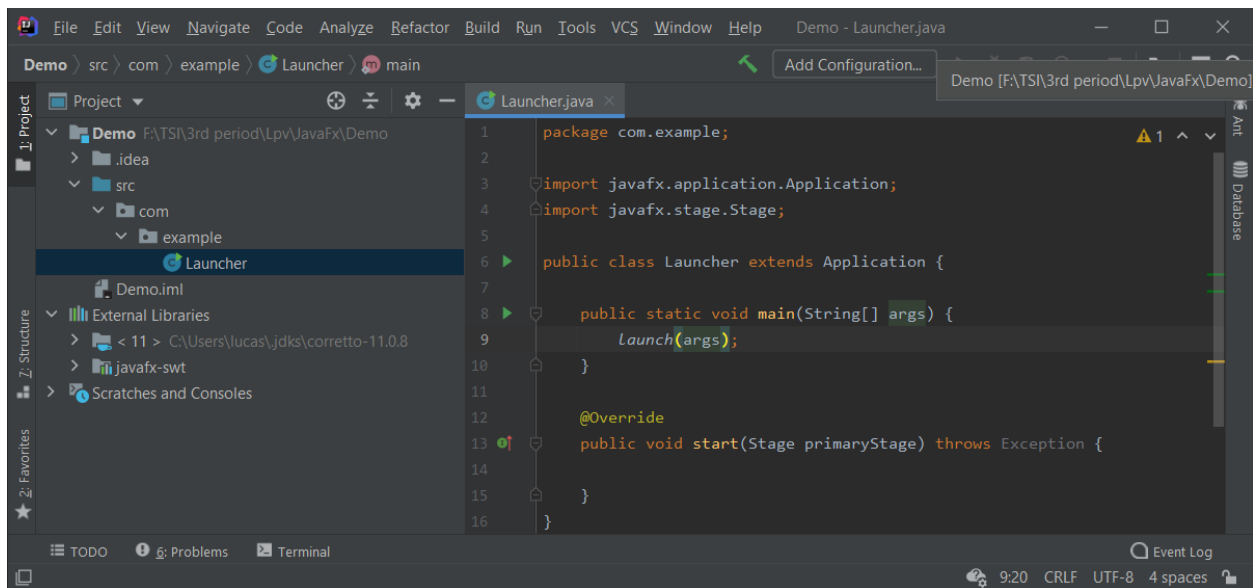


Figura 11: Herdando da classe Application e iniciando a aplicação através do main.

Vamos agora, inserir um botão com um evento de clique e exibir a janela.

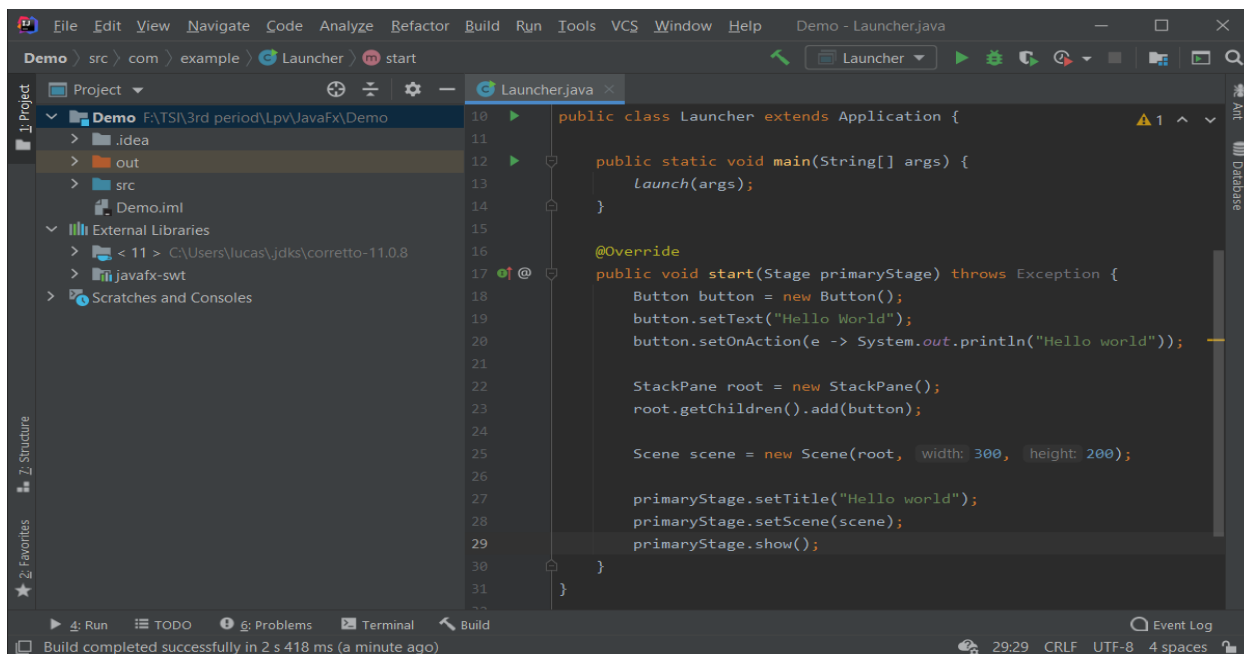


Figura 12: Criando uma janela simples.

Nossa primeira janela está criada. Nela, temos um botão que ao ser clicado, exibe no console, a mensagem “Hello world”. Vá em frente e execute seu programa, o que acontece?

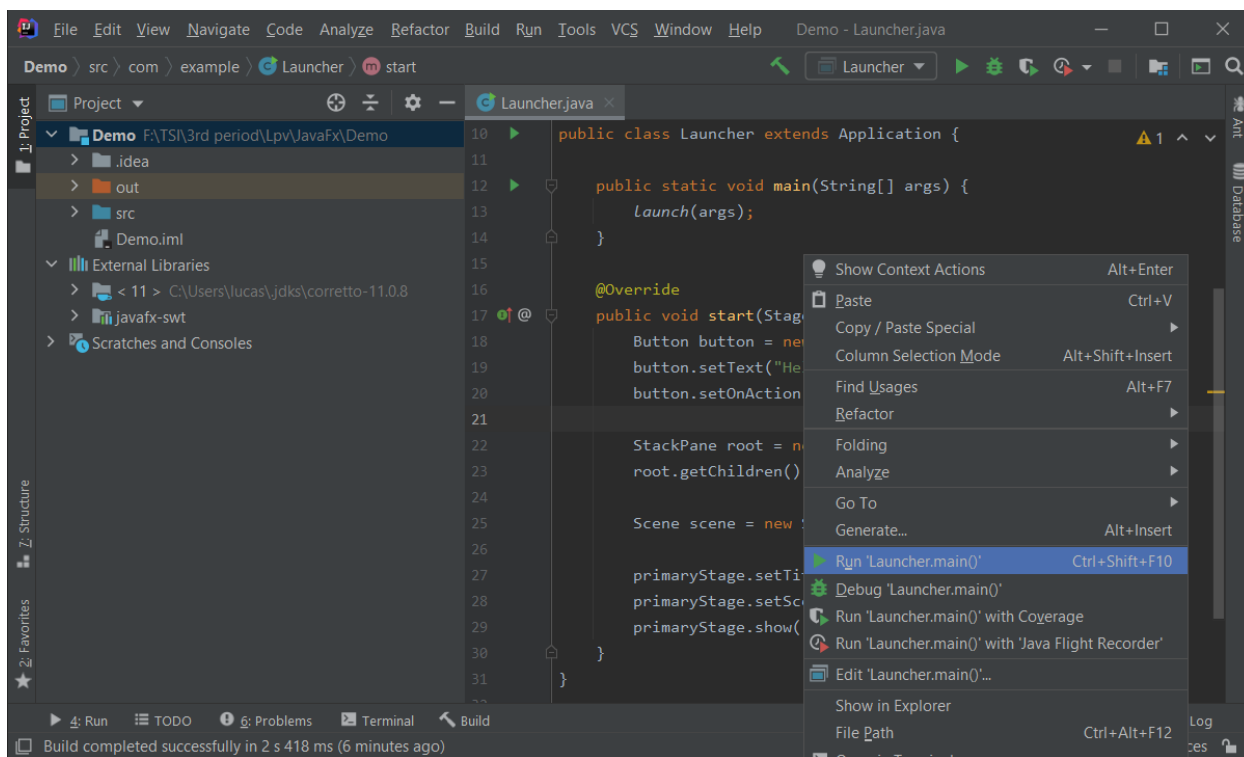


Figura 13: Executando o programa.

Para sua surpresa, a janela não é exibida e uma exceção ocorre:

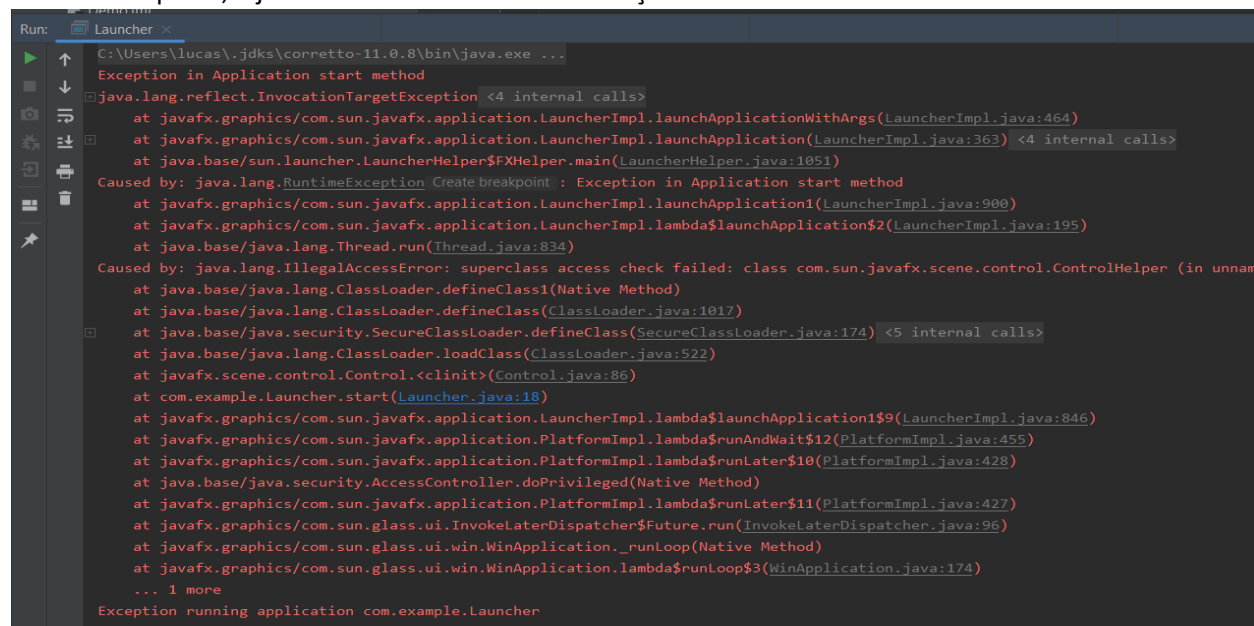


Figura 14: exceção de módulo não configurado

Por que isso acontece?

O problema é que ainda não configuramos os módulos de nosso projeto. Fazendo com que o aplicativo falhe em acessar as classes necessárias. Para resolver esse problema, vamos configurar nosso módulo. Clique com o botão direito do mouse no pacote `src` de seu projeto, vá em `new -> module-info.java`.

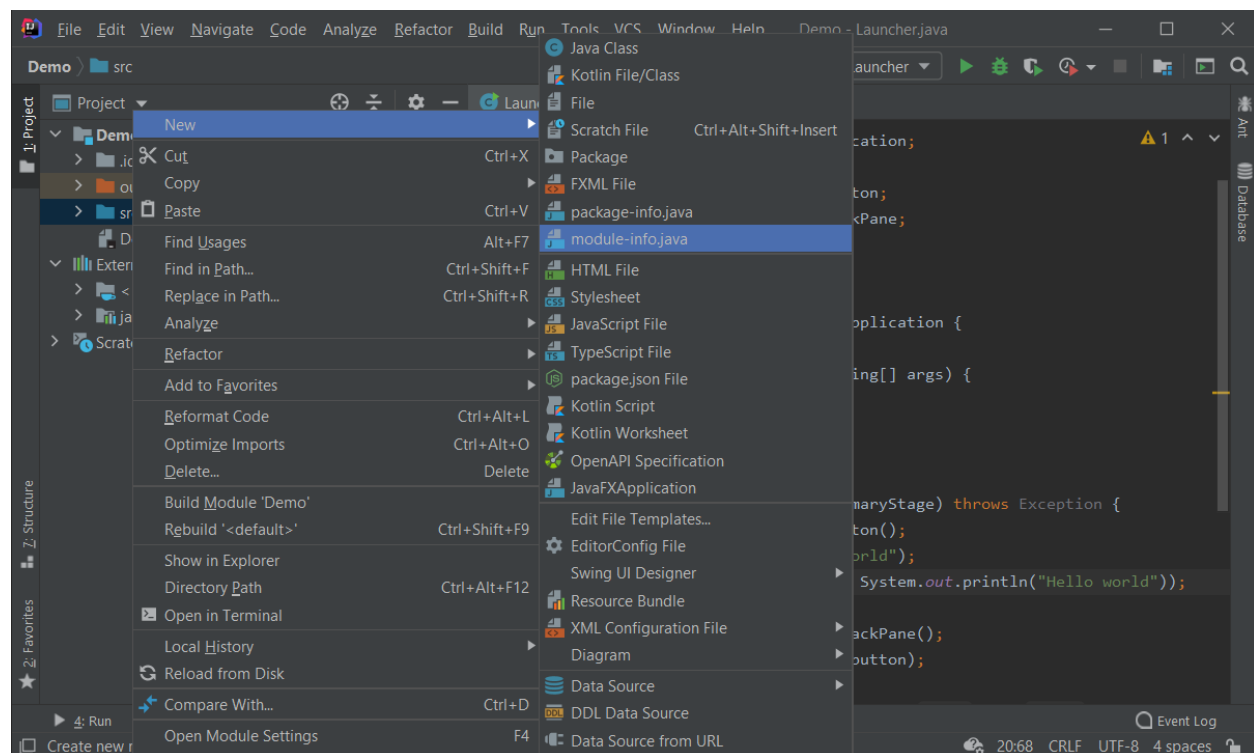


Figura 15: Criando o arquivo de módulo do projeto.

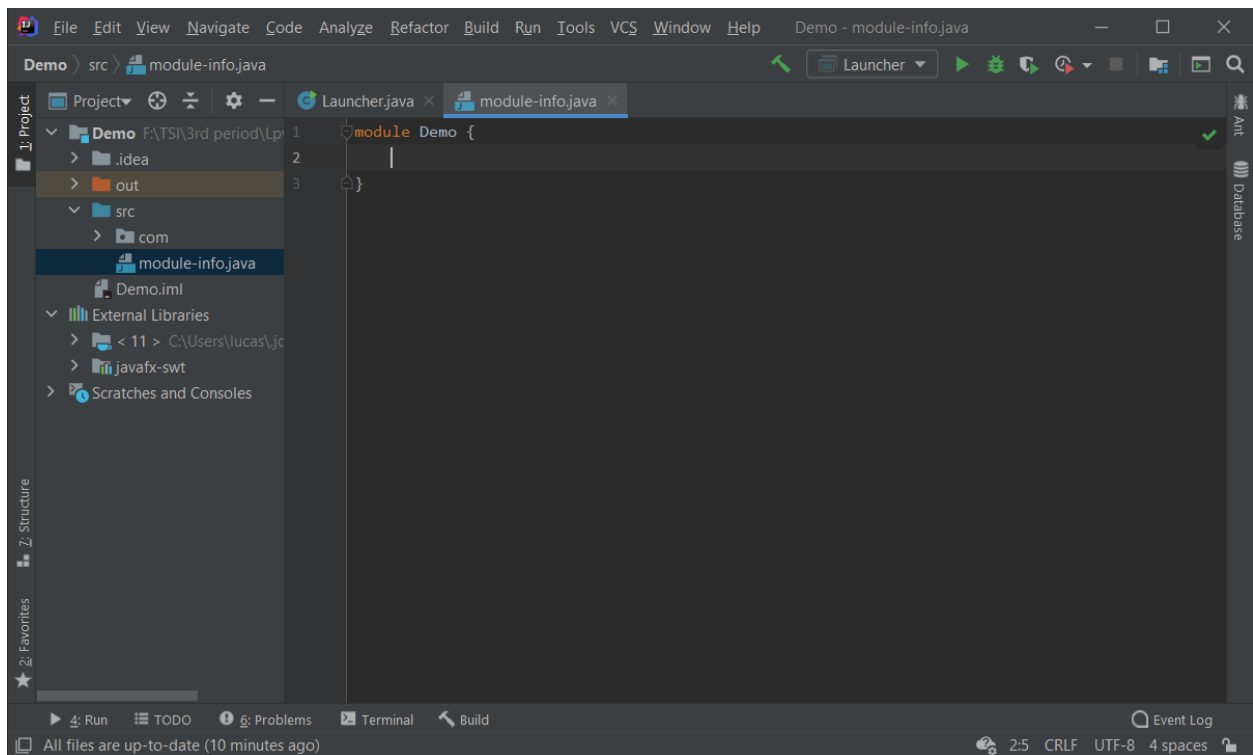


Figura 16: Arquivo de modulo criado.

Agora vamos popular este arquivo com os pacotes JavaFx que iremos usar e especificar o caminho de nossa classe de lançamento:

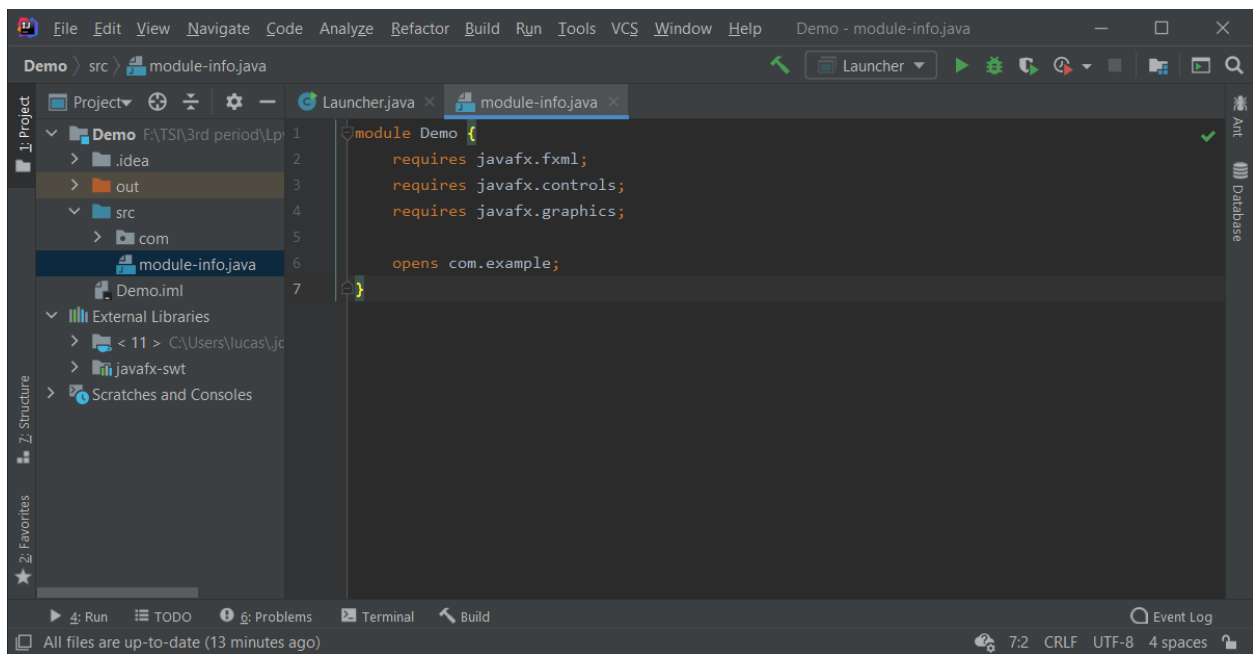


Figura 17: Arquivo de modulo populado.

Agora, ao rodar o projeto novamente, a janela será exibida. Clique no botão e veja o que acontece.

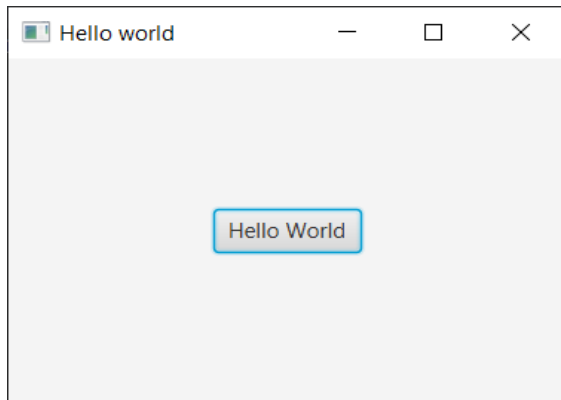


Figura 18: Primeira janela JavaFx exibida.

Migrando a view para o arquivo de FXML

Ao trabalharmos com janelas mais elaboradas, o processo de criação de uma janela via código se torna muito trabalhoso, maçante e ineficiente além de gerar um código de menor qualidade. Por essas e outras razões, é recomendado que a criação de telas seja feita através de arquivos FXML.

Para criar uma janela usando o FXML, siga os passos abaixo.

Clique com o botão direito sobre o pacote onde o arquivo deverá ser criado, clique em new -> FXML File, dê o nome desejado e confirme. Será criado um arquivo FXML com as definições iniciais do layout.

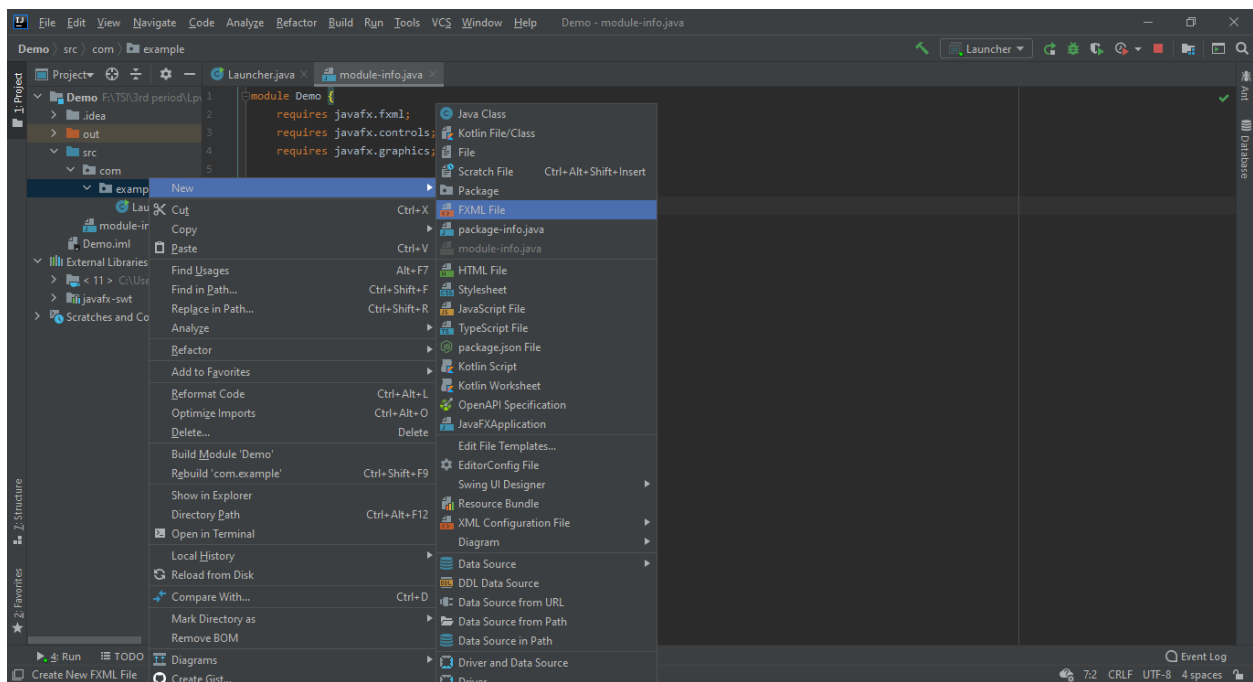


Figura 19: Criando um arquivo FXML – passo 1

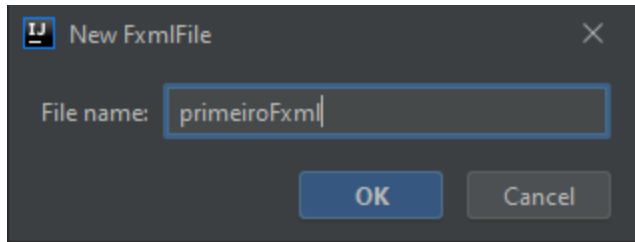


Figura 20: Criando um arquivo FXML – passo 2.

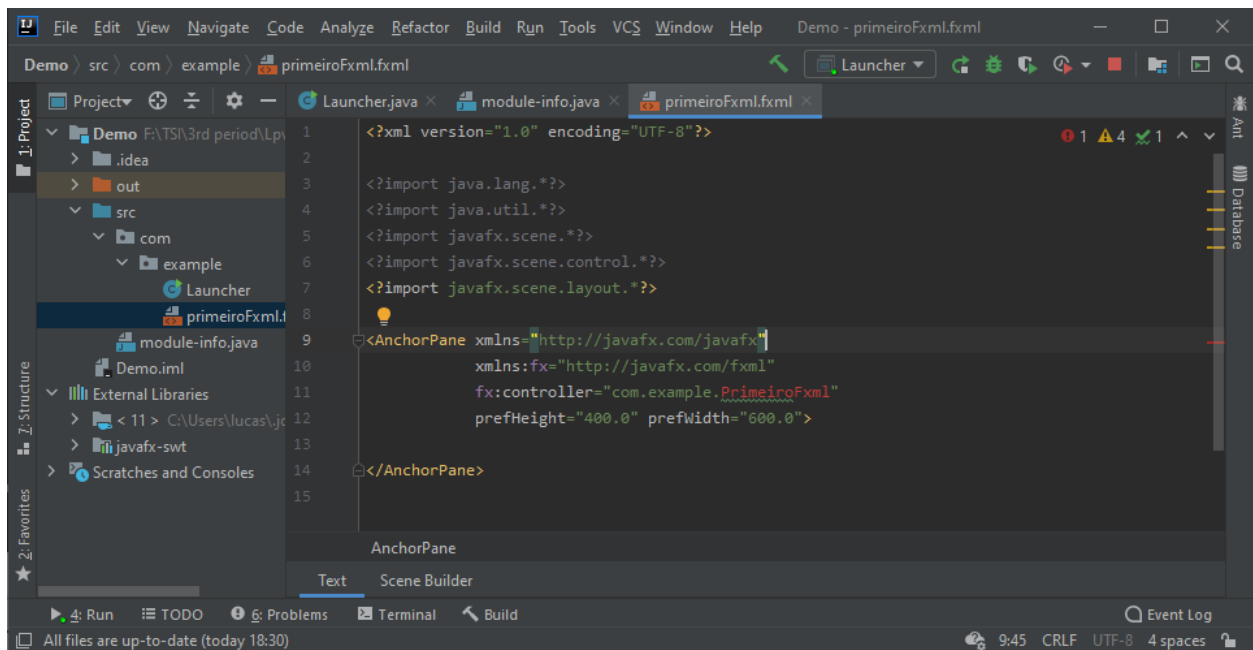


Figura 21:Arquivo FXML criado.

Veja que o FXML, assim como o Java, faz uso de imports porém, no estilo do próprio FXML e também que há um atributo `fx:controller`. Este atributo é muito importante, pois é através dele que definimos o que acontece com nossos elementos visuais. O controller aqui definido não existe, logo deverá ser criado. Você pode criá-lo de duas maneiras, criando uma classe como normalmente fazemos ou, usando a combinação de teclas `F2 ALT + ENTER` sendo que `F2` posiciona sobre o erro, e `ALT + ENTER` abre o menu de opções para correção do erro

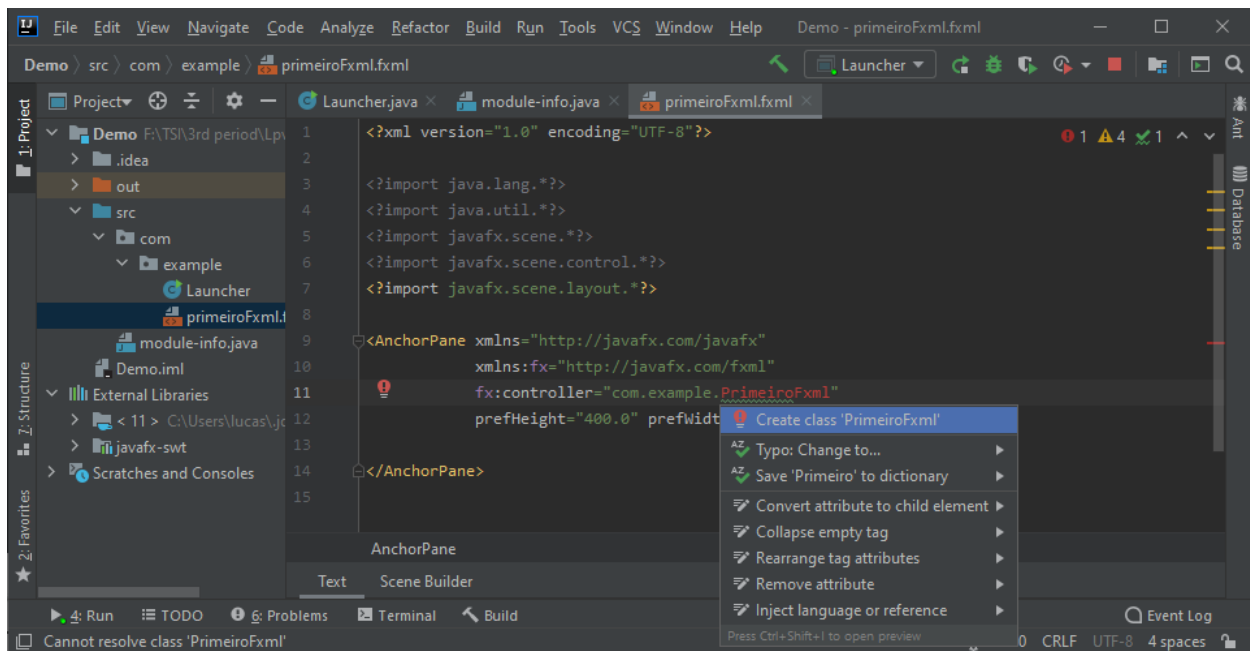


Figura 22: usando atalhos para implementar a classe controladora.

Feito isso, teremos uma view com um controller associado a ela. Mas ainda sem comportamento.

Vamos alterar a nossa classe Launcher para que ela inicie a aplicação a partir dessa nova view. Altere o método start para que fique da seguinte maneira

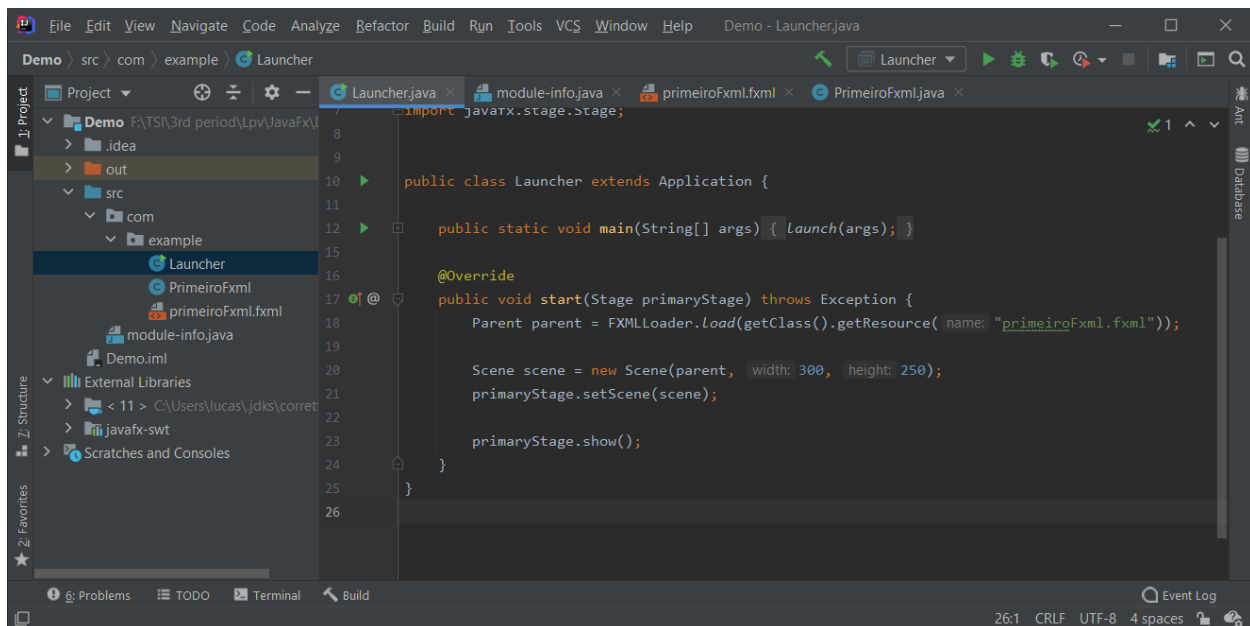


Figura 23: alterando a classe lançadora para que exiba a view criada via arquivo fxml.

Rode o programa. A janela abaixo deverá ser exibida:

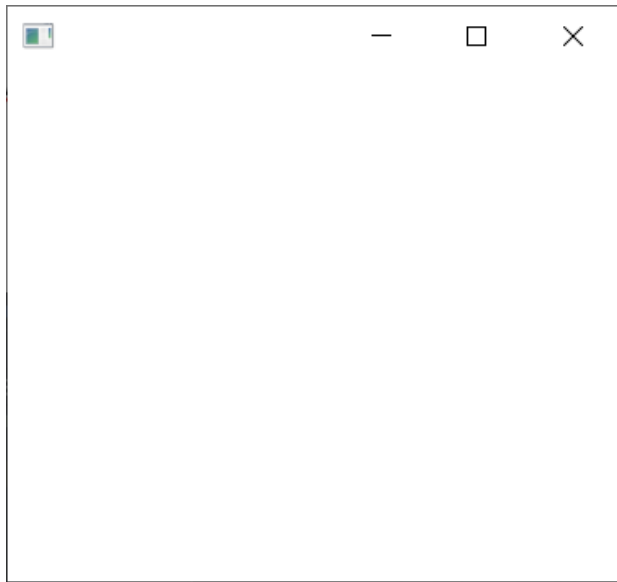


Figura 24: primeira janela via arquivo FXML

Vamos adicionar um botão na view criada. Assim como em HTML, os elementos de uma view são criados através de tags, tendo seus atributos. Essas tags devem ser colocados dentro de uma tag especial, a tag *children*.

No exemplo abaixo vamos criar um botão com uma ação similar ao que fizemos anteriormente:

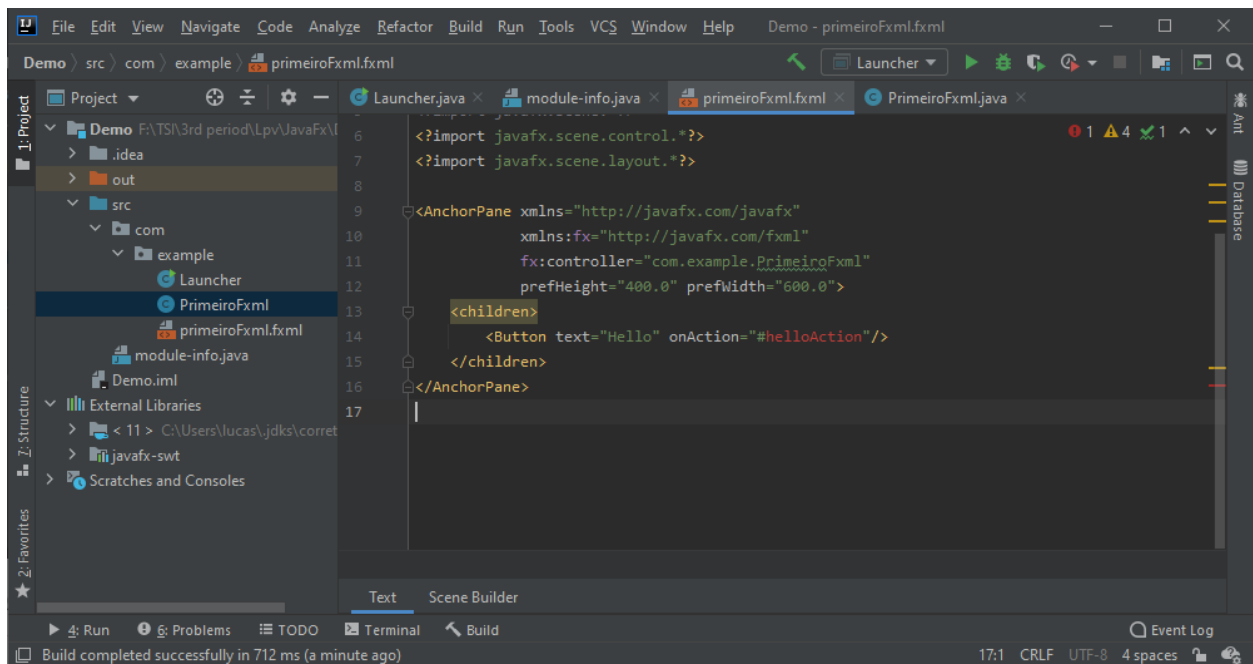


Figura 25: adicionando um botão através do editor FXML.

Note que o botão criado possui dois atributos, sendo o atributo **text**, o texto a ser exibido ao usuário e o atributo **onAction**, a ação a ser realizada ao clicar no botão sendo a última, implementada no controlador da view. Para implementar a ação do botão, use a combinação de teclas F2 + ALT + ENTER e selecione a opção *Create method*:

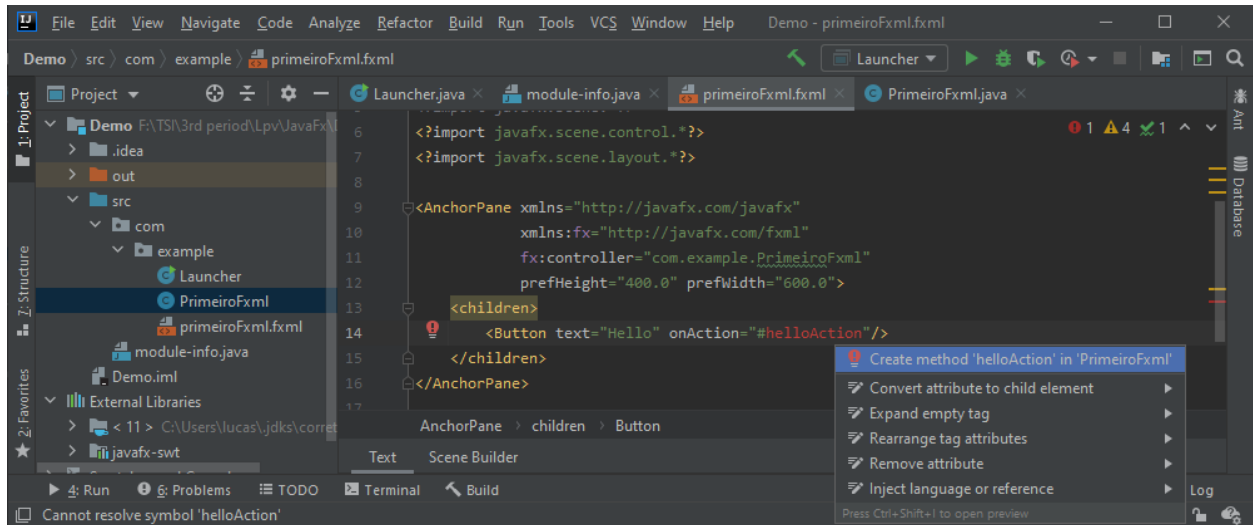


Figura 26: adicionando comportamento ao botão - passo 1

Um método será inserido à classe controladora da view:

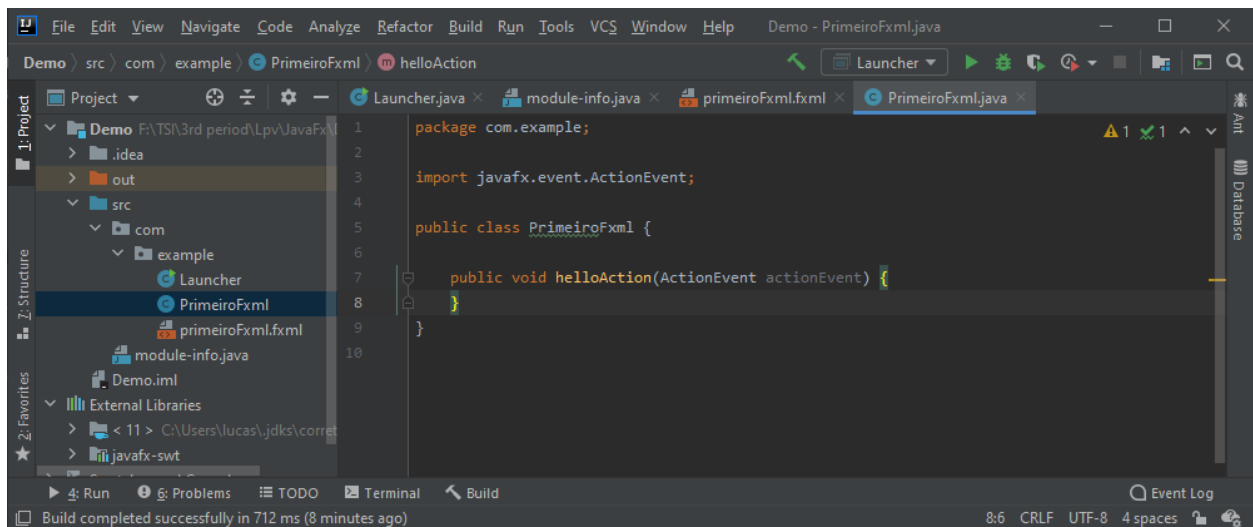


Figura 27: método adicionado à classe controladora.

Faça com que esse método imprima algo no console:

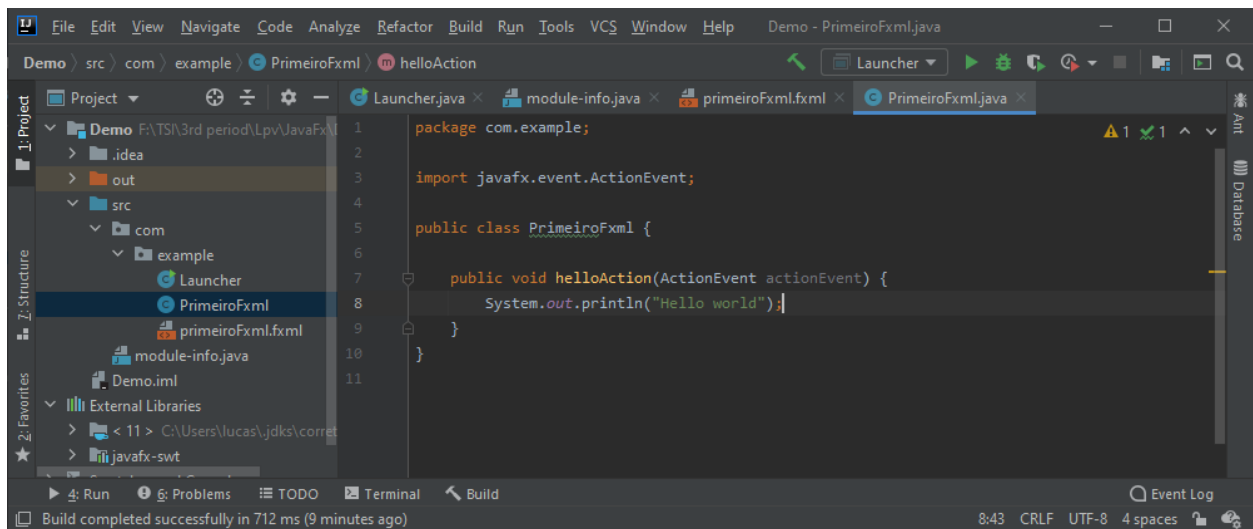


Figura 28: imprimindo no console.

Execute a aplicação e clique no botão, veja que o comportamento é o mesmo que a view implementada diretamente via código Java, mas agora, o layout de sua janela está implementado separadamente de seu comportamento, facilitando a manutenção e legibilidade.

Scene builder

A melhor maneira de trabalhar com layouts e a combinação do uso de arquivos FXML com um programa externo para a edição do arquivo FXML. Usando o Scene builder temos as seguintes vantagens:

- Adição de elementos nas views através do arrastar e soltar;
- Fácil alinhamento dos elementos no layout;
- Pré-visualização do layout.

O Scene builder pode ser encontrado para download [aqui](#). Após realizado o download, a instalação pode ser realizada através do instalador padrão do Windows ou extração dos arquivos em seu diretório de preferência em caso de sistemas Linux. Feito isso, o Scene builder estará pronto para ser usado.

Para usar o Scene builder no IntelliJ, dê um clique com o botão direito do mouse sobre o arquivo FXML que deseja editar e selecione *Open In SceneBuilder*

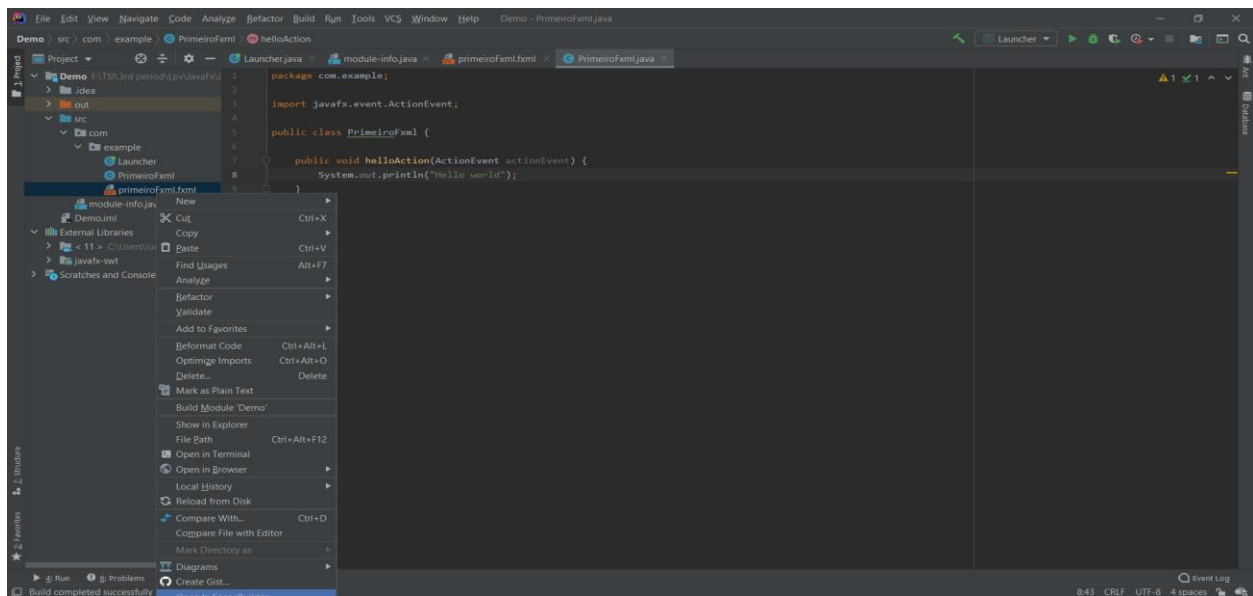


Figura 29: abrindo FXML com scene builder.

No primeiro uso do scene builder, o IntelliJ irá solicitar que seja definido o caminho do executavel do mesmo. Para isso navegue até o diretório de instalação (ou extração) do scene builder e selecione o executavel e clique em OK.

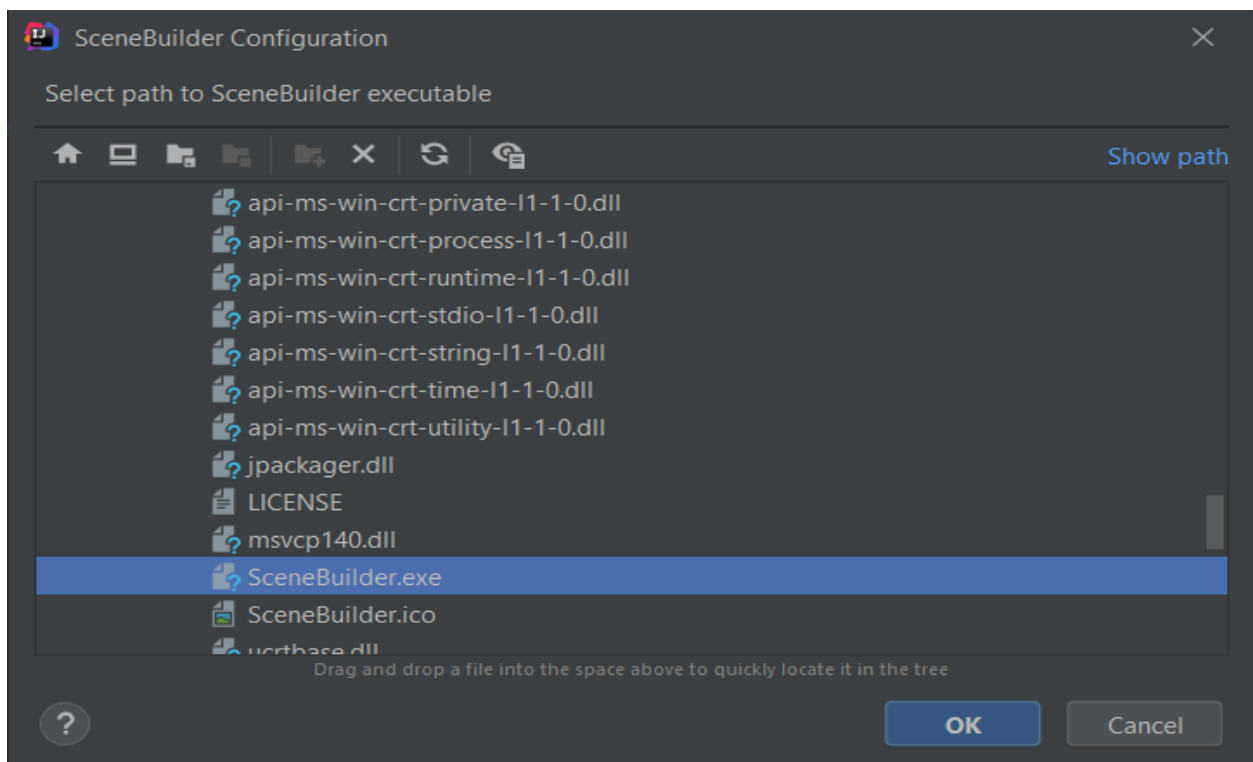


Figura 29: Definindo caminho de instalação do scene builder.

Será inicializado o scene builder com o arquivo de layout carregado:

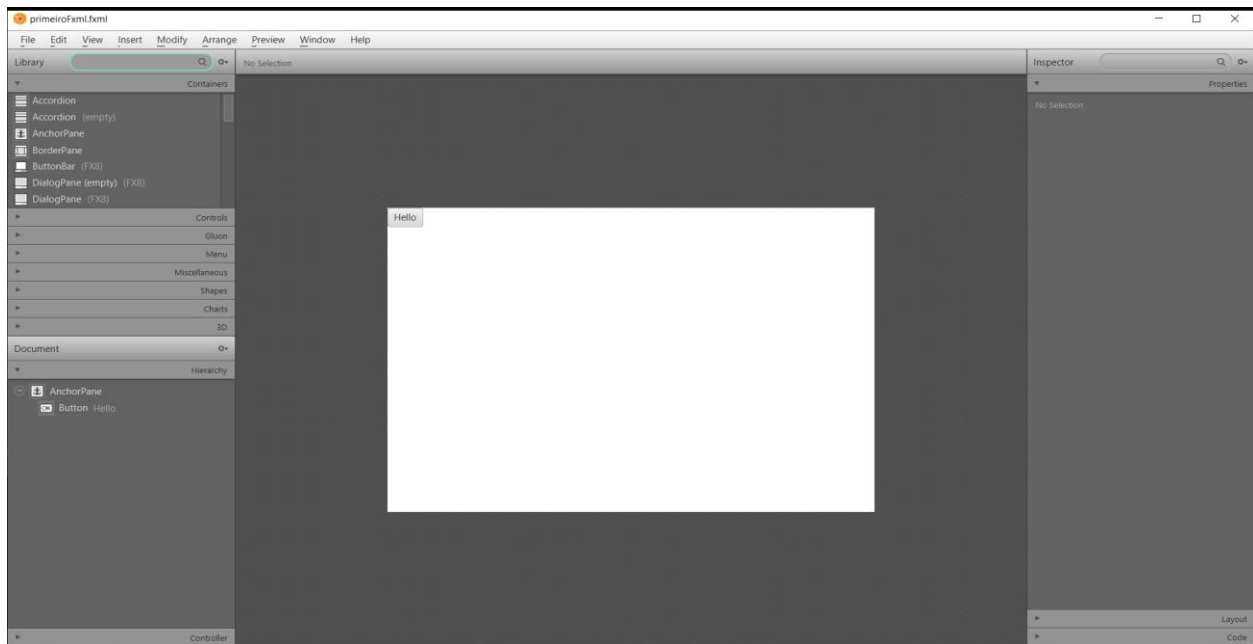


Figura 30: Arquivo FXML aberto via Scene Builder.

Agora podemos realizar todo o processo da criação do layout aqui que será aplicado no arquivo FXML.

Vamos agora conhecer a disposição das funcionalidades que você, como desenvolvedor JavaFx, frequentemente fará uso.

No topo à esquerda, se encontra a biblioteca de componentes visuais, nela são disponibilizados todos os componentes padrão do JavaFx para a criação de telas

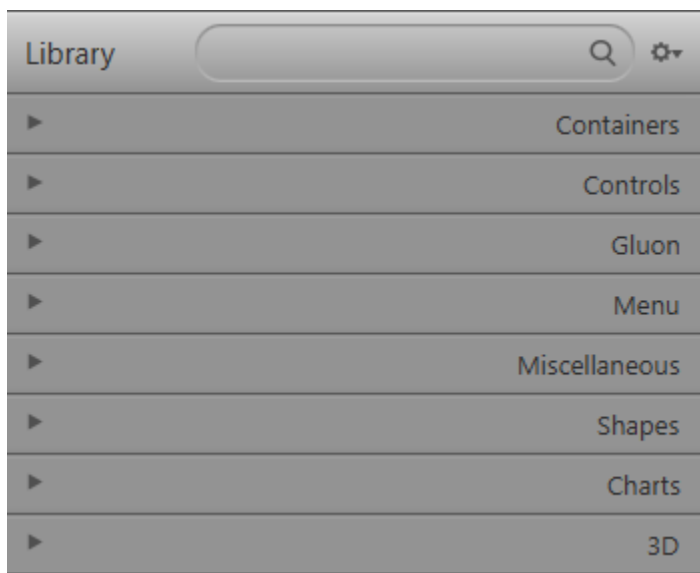


Figura 31: Biblioteca de componentes JavaFx.

No canto inferior à esquerda, encontra-se a região de informações do documento sendo editado. Nesta região pode ser alterado a classe controladora e visualizada a hierarquia dos componentes dentre outras ações que refletirão sobre o layout da view como um todo.

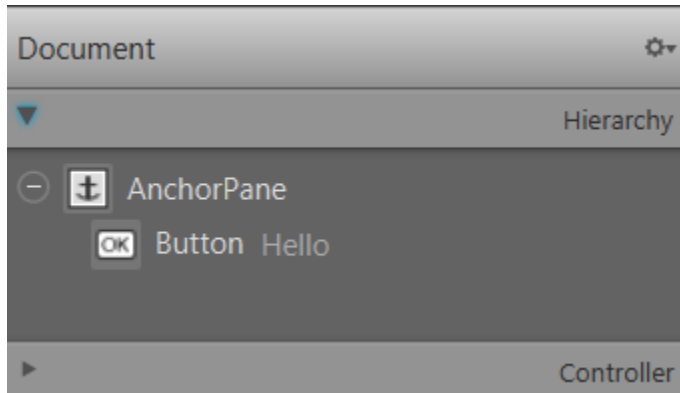


Figura 32: Hierarquia do layout.

No canto superior direito, há uma região para o gerenciamento de um componente específico dentro da view. Esta região está dividida em 3 sub regiões.

1. Propriedades – Essa região é destinada a definição e edição das propriedades do componente tais como texto, alinhamento do texto, visibilidade se está ligado ou desligado e etc;
2. Layout – Essa região é destinada a determinar a disposição e comportamento do elemento em relação ao seu container pai. Aqui pode ser definidas as margens, o padding, o tamanho, se o componente é redimensionável e outras.
3. Code – Aqui definimos a identidade do componente e como ele irá responder a diversos eventos tais como ao ser clicado, ao ser arrastado, etc.

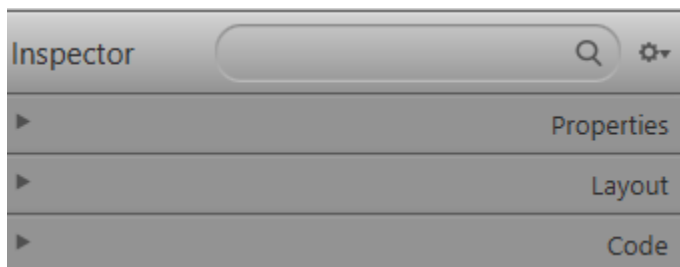


Figura 33: Região para gerenciamento de componente gráfico.

Nota: Ids e comportamentos criados através do Scene builder deverão ser adicionados manualmente à classe controladora da view. Para tal pode-se fazer o uso do menu *view -> show sample controler eskeleton* para copiar as definições:

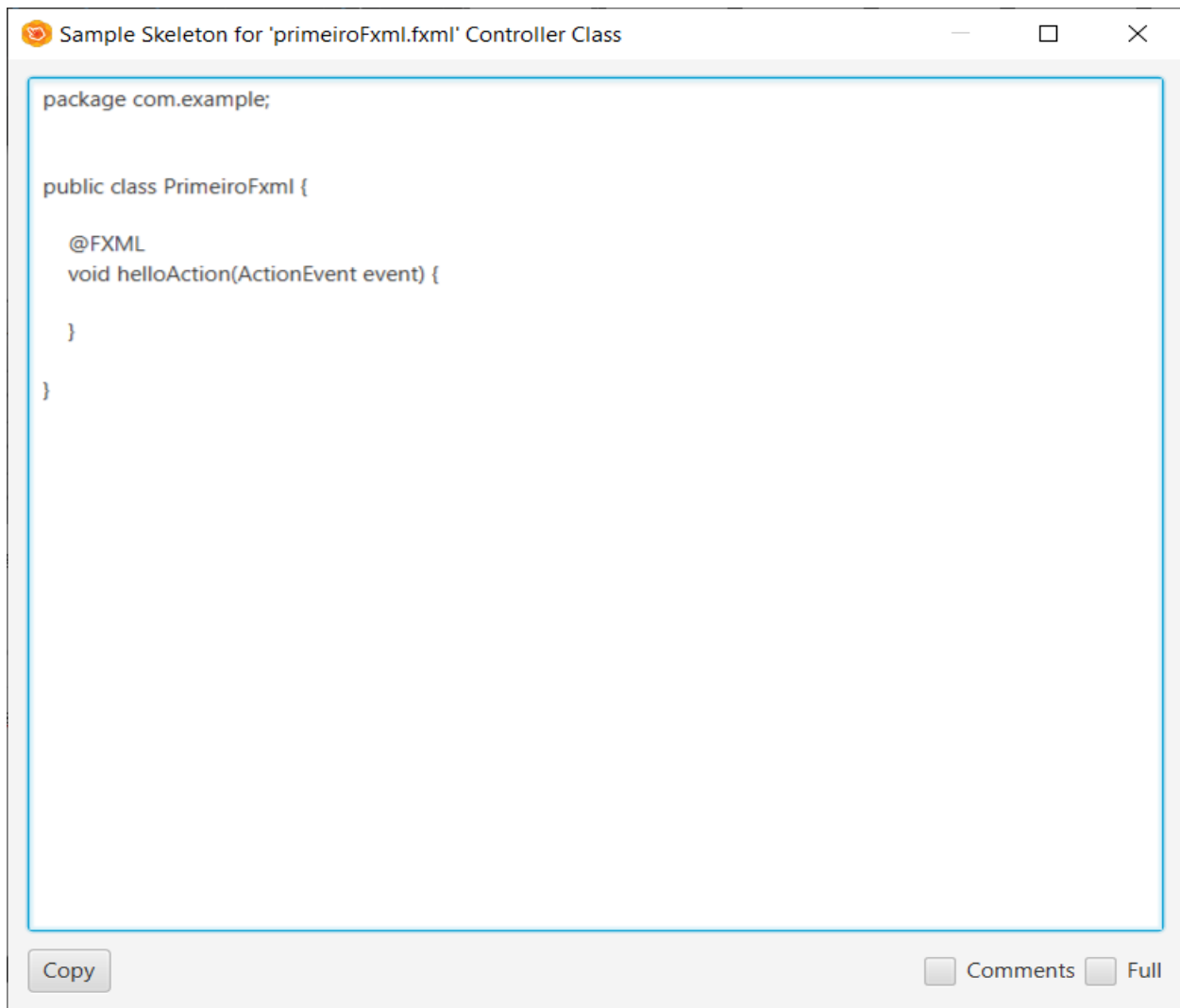


Figura 34: Esqueleto do controlador gerado via Scene builder.

Componentes Básicos

- **Botão** Para adicionar um botão à view, vá à região da *Biblioteca de componentes*, selecione a aba *Controls*, selecione o componente Button e arrastre para a posição desejada. Você pode editar o texto do botão dando dois cliques sobre ele e digitando o novo texto.
-

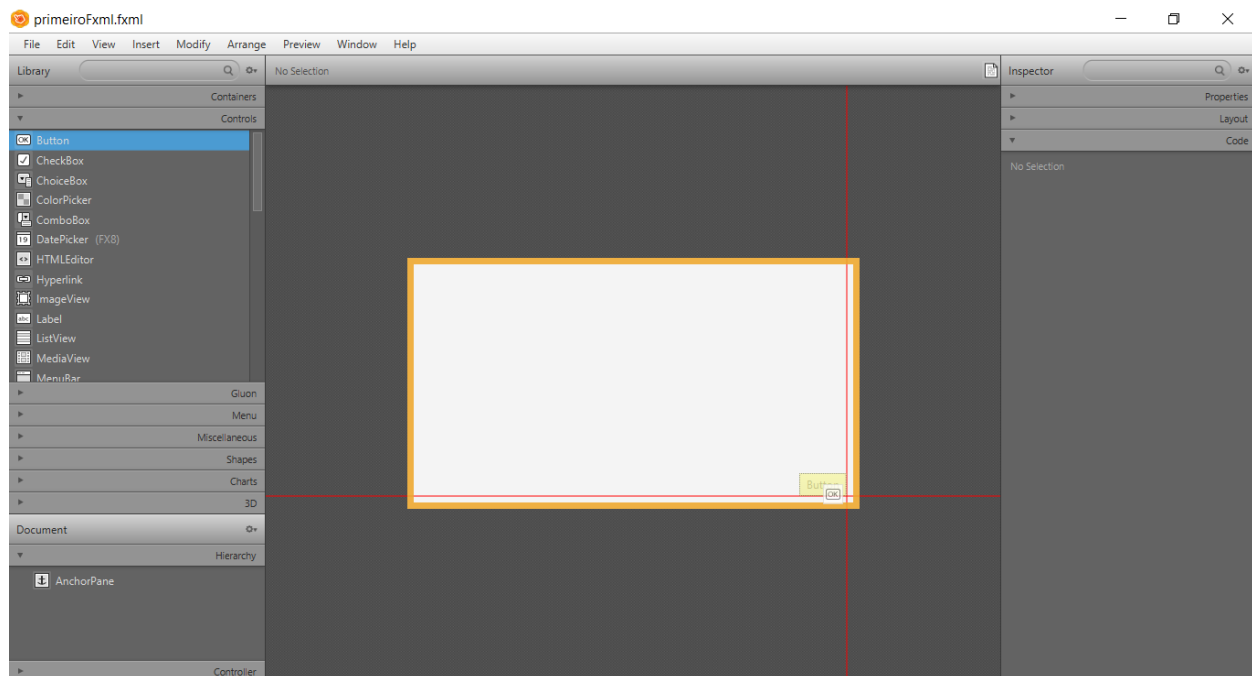


Figura 35: Adicionando botão à view.

- **Caixa para entrada de texto** Para adicionar uma caixa de texto, vá à região da *Biblioteca de componentes*, selecione a aba *Controls*, selecione o componente *TextField* e arraste para a posição desejada.

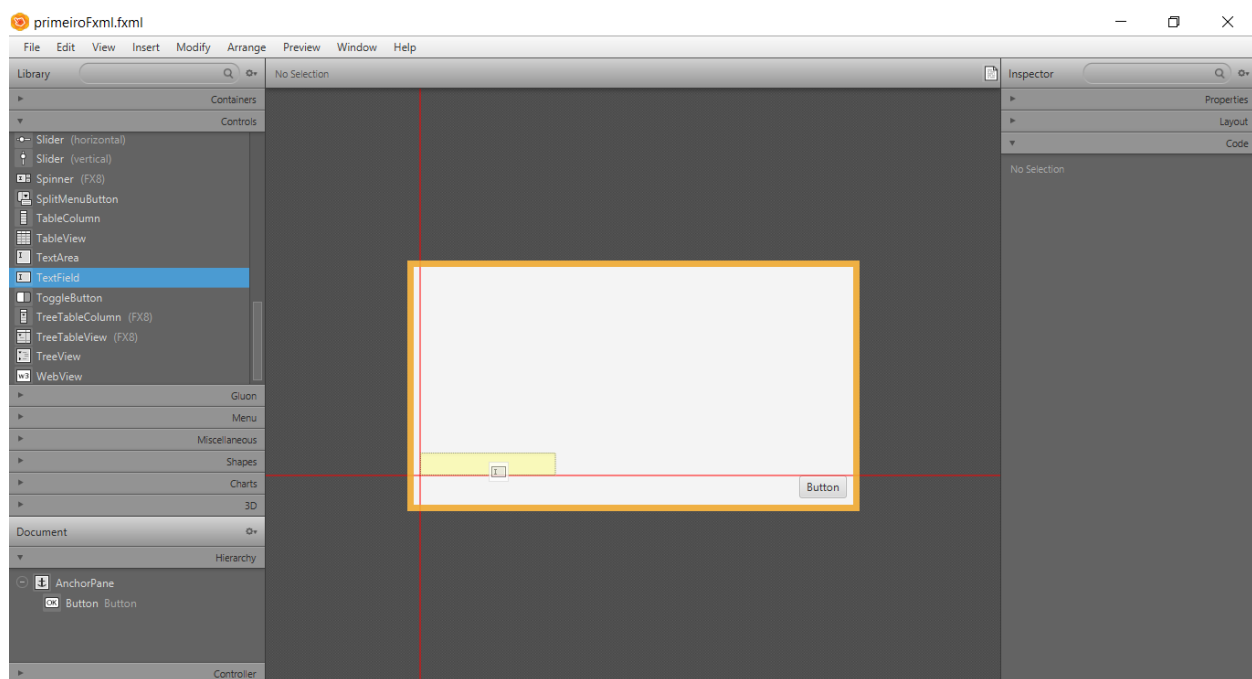


Figura 36: Adicionando caixa de texto à view.

- **Label de texto** Na região da *Biblioteca de componentes*, selecione a aba *Controls*, selecione o componente Label e arrastre para a posição desejada. Você pode editar o texto do label dando dois cliques sobre ele e digitando o novo texto.

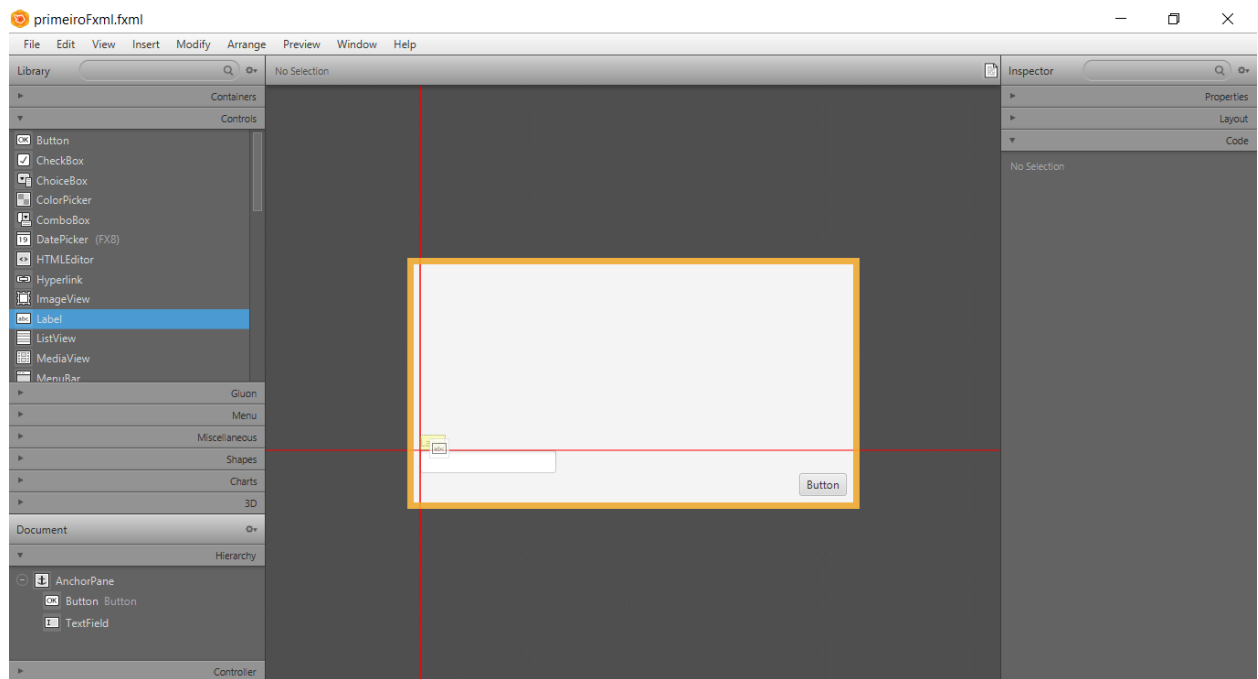


Figura 37: Adicionando label à view.

- **Radio button** na região da *Biblioteca de componentes*, selecione a aba *Controls*, selecione o componente RadioButton e arrastre para a posição desejada. Você pode editar o texto do Radio button dando dois cliques sobre ele e digitando o novo texto.

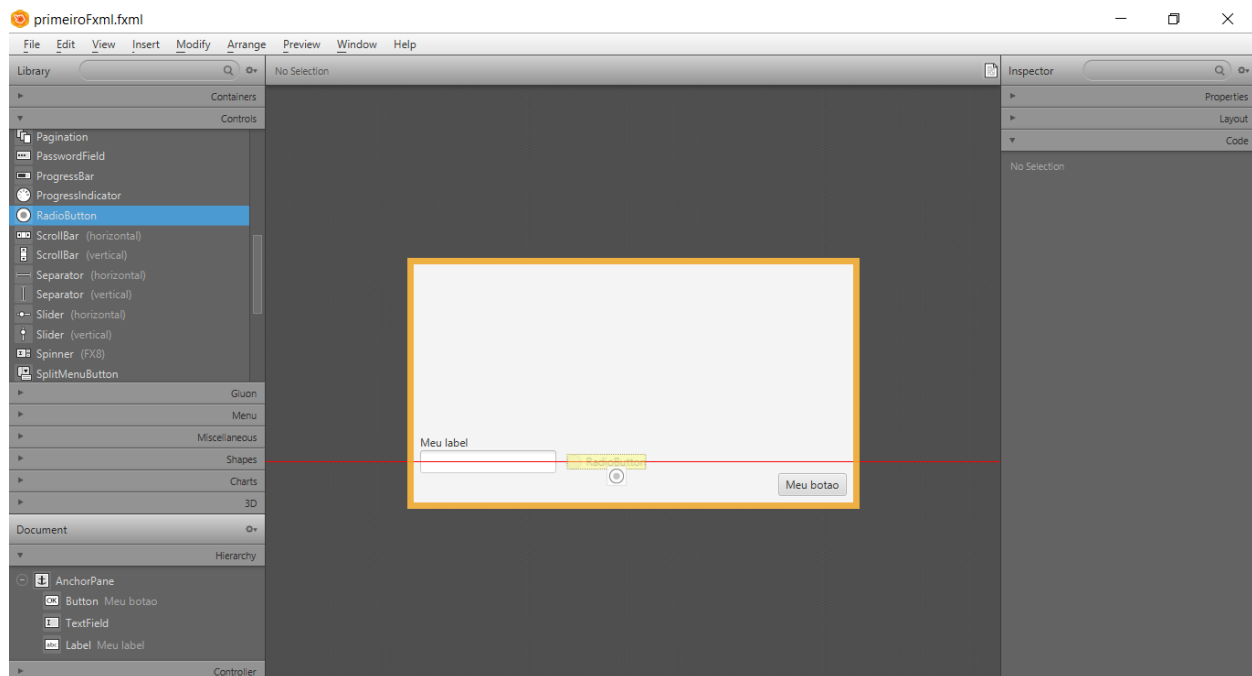


Figura 38: Adicionando RadioButton à view.

- **Checkbox** na região da *Biblioteca de componentes*, selecione a aba *Controls*, selecione o componente *CheckBox* e arraste para a posição desejada. Você pode editar o texto do *CheckBox* dando dois cliques sobre ele e digitando o novo texto.

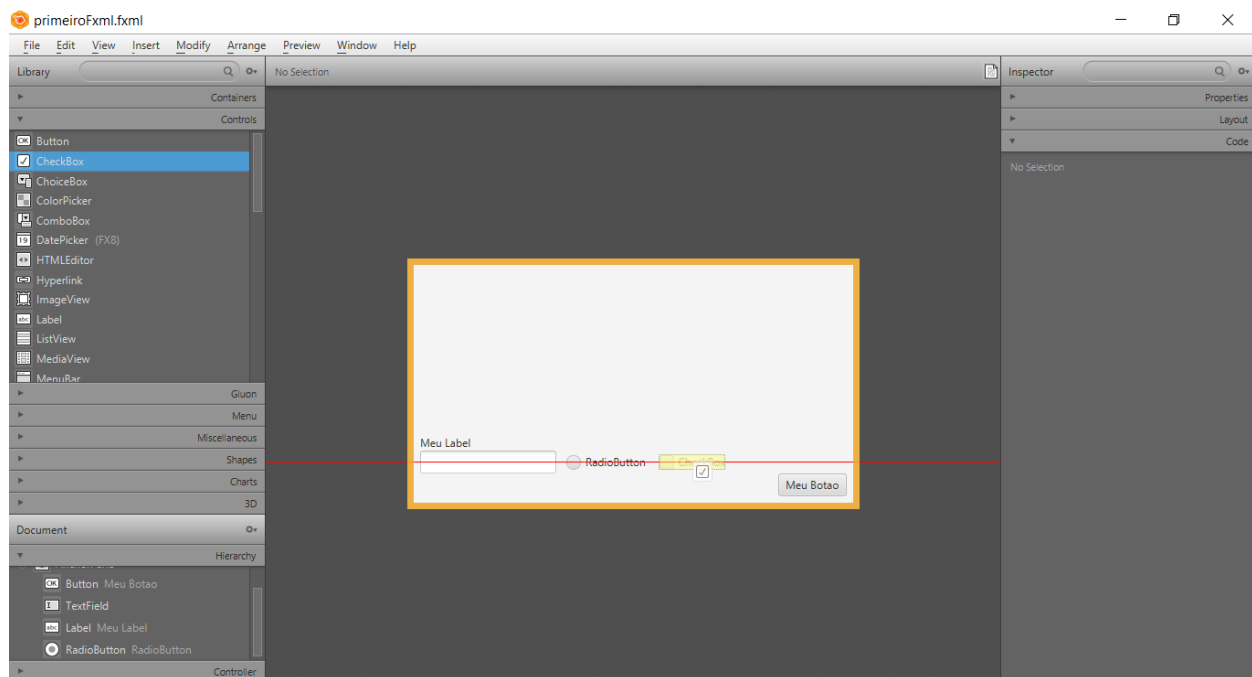


Figura 39: Adicionando Checkbox à view.

- **Lista Suspensa** na região da *Biblioteca de componentes*, selecione a aba *Controls*, selecione o componente *ChoiceBox* e arraste para a posição desejada.

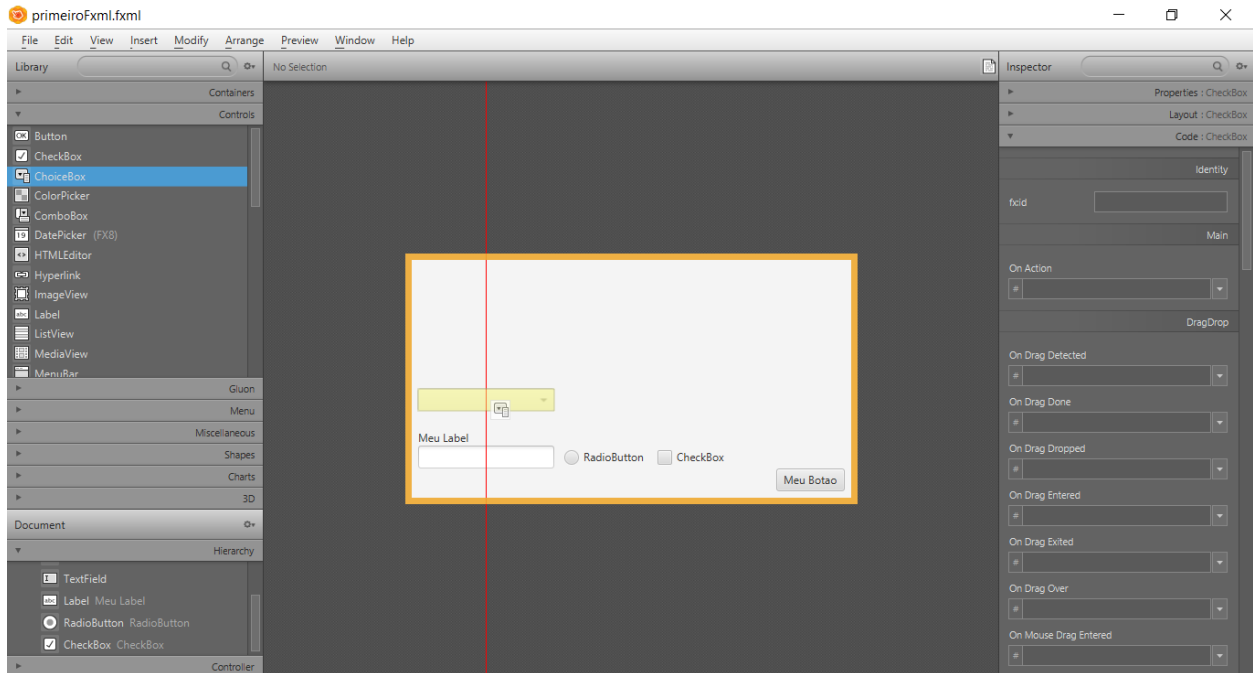


Figura 40: Adicionando lista suspensa à view.

- **Slider** na região da *Biblioteca de componentes*, selecione a aba *Controls*, selecione o componente *Slider* e arraste para a posição desejada.

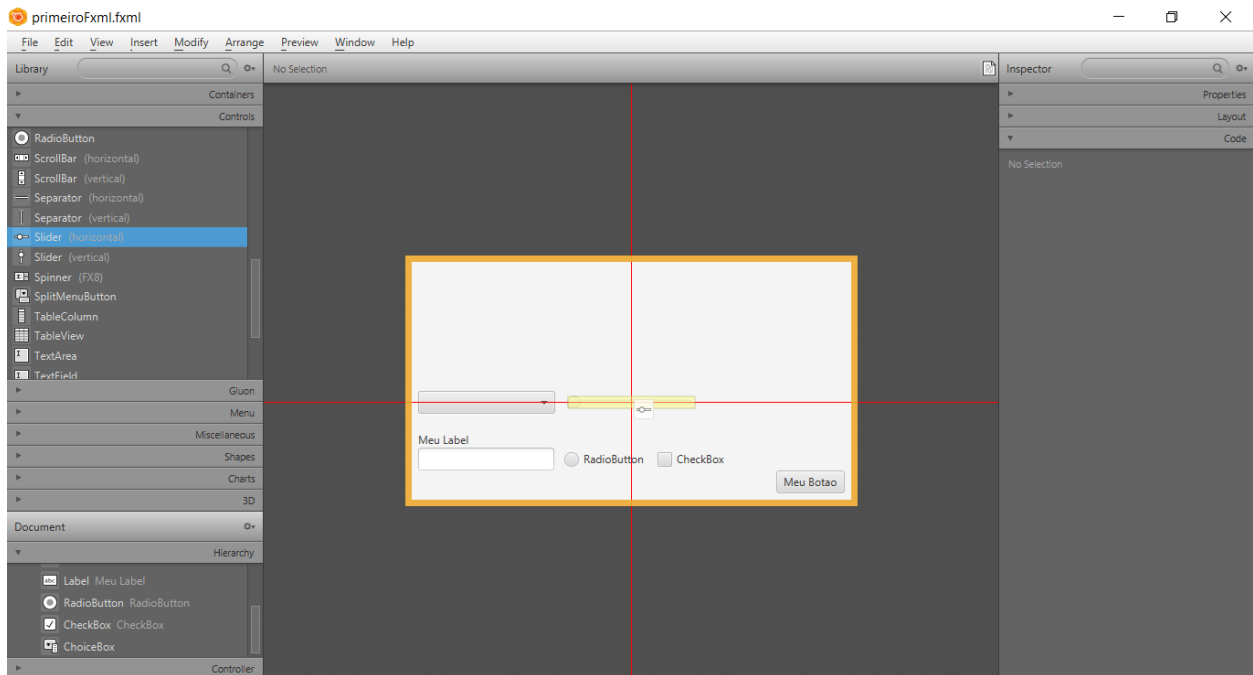
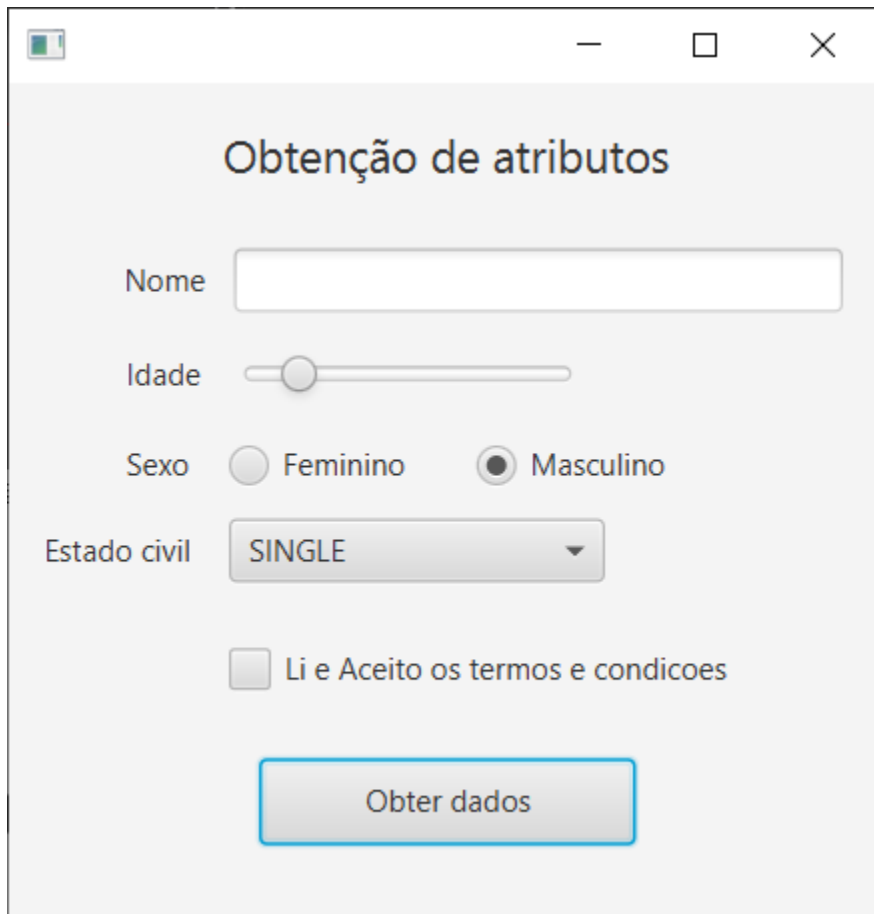


Figura 41: Adicionando slider à view.

Obtenção de atributos

Para demonstrar como é realizada a obtenção de conteúdo dos componentes, será usada a seguinte view:



Obtenção de atributos

Nome

Idade

Sexo ☐ Feminino ☒ Masculino

Estado civil

☐ Li e Aceito os termos e condicoes

Figura 42: view para demonstração de obtenção de conteúdo.

Como é apenas para demonstração, não haverá validações, consistindo apenas na recuperação dos dados inseridos pelo usuário e exibição em uma caixa de diálogo.

Como será necessária a manipulação dos componentes pelo controller da view, devemos definir os ids de todos os componentes a serem acessados. Como dito anteriormente, isso pode ser realizado na região Inspector -> Code

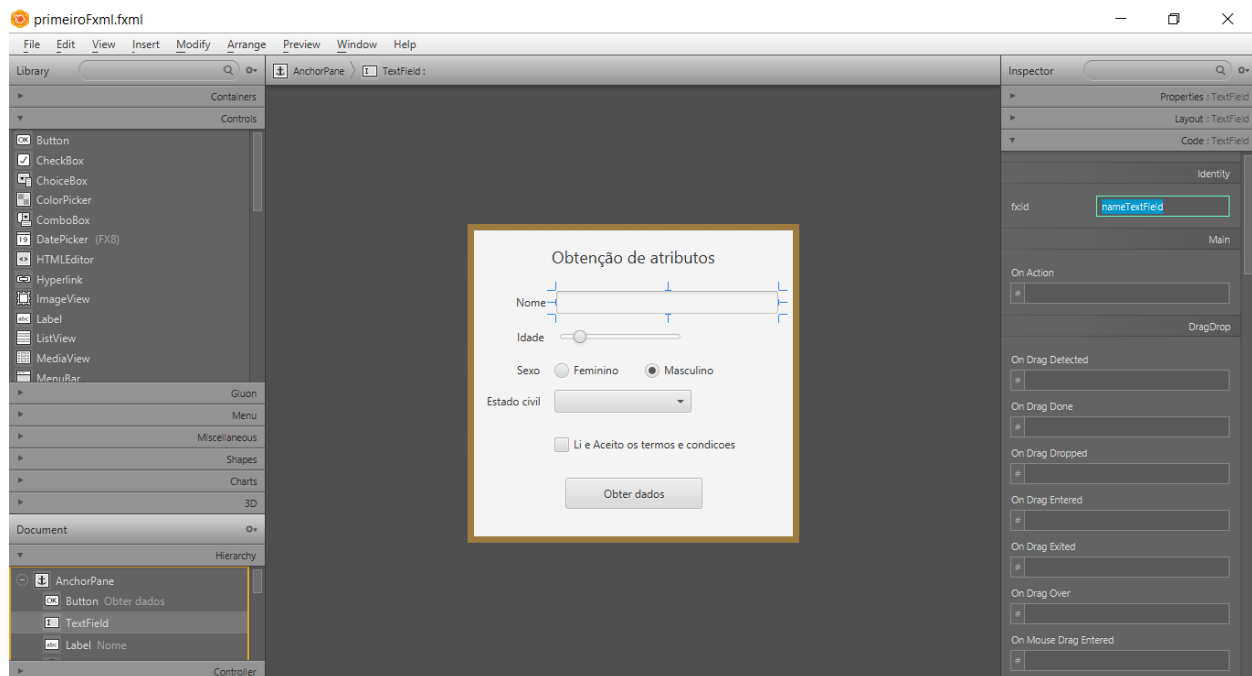


Figura 43: definindo o id do campo de texto.

Como a definição dos ids será um processo repetitivo, não será apresentado aqui para todos os elementos. Vamos passar direto para a criação do evento de clique no botão.

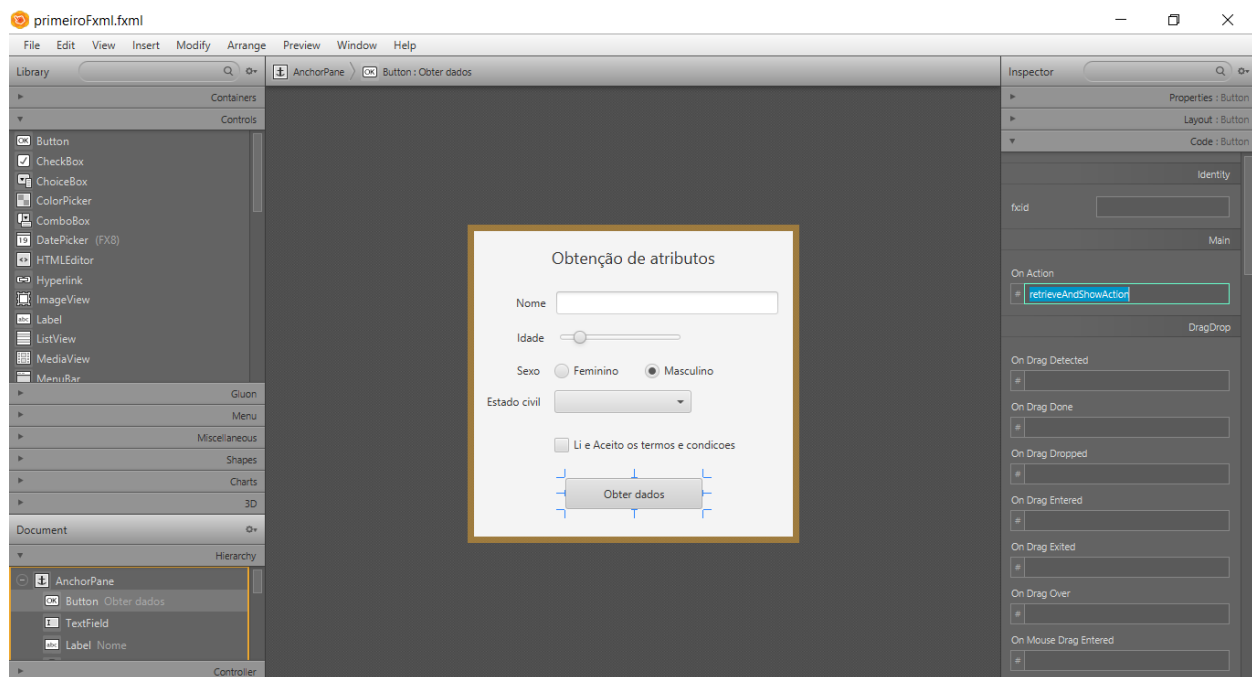


Figura 44: definindo o evento de clique para o botão.

Agora vamos copiar todo o código gerado para o controlador e colar na nossa classe controladora.

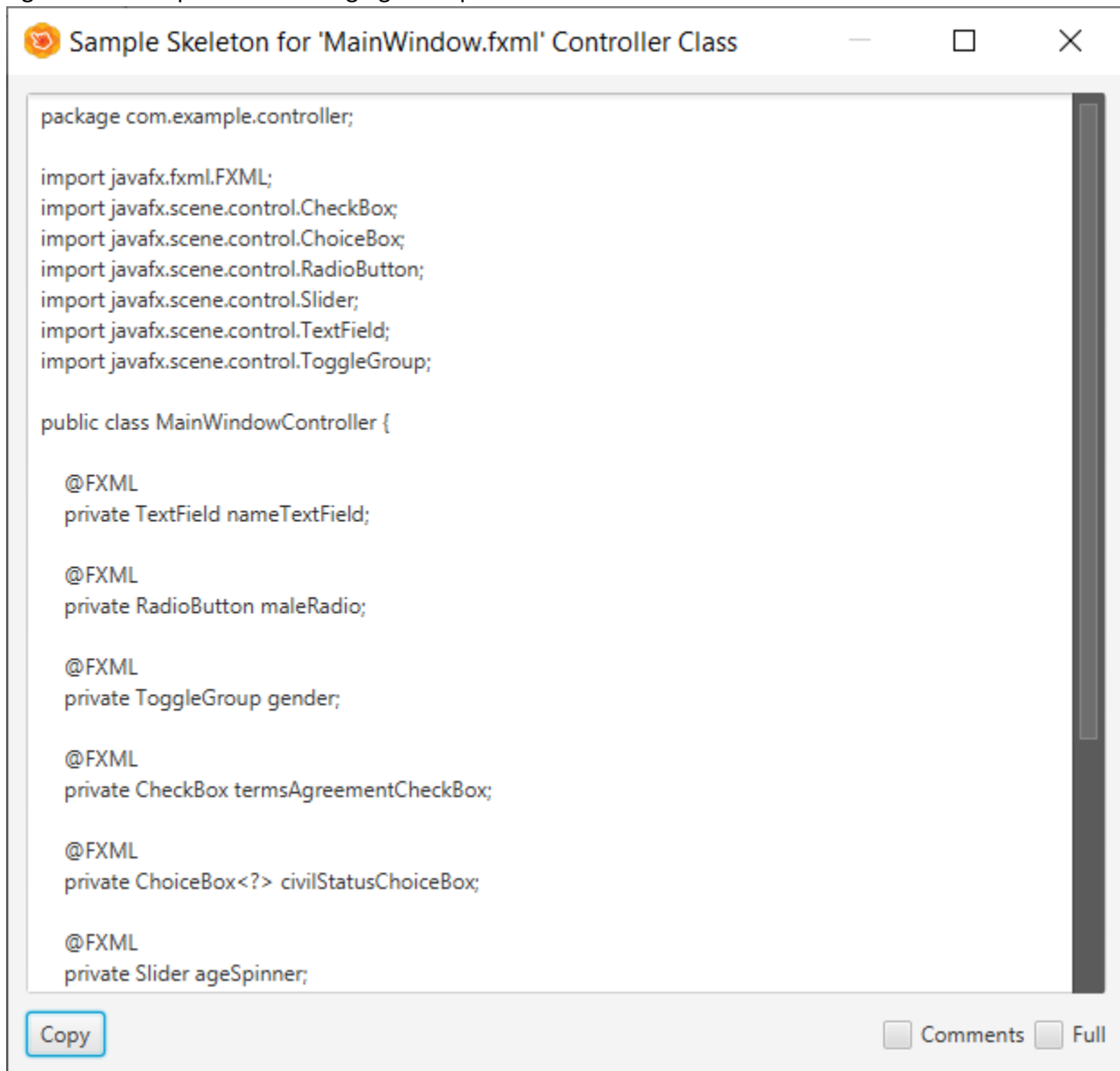


Figura 45: copiando o esqueleto do controlador.

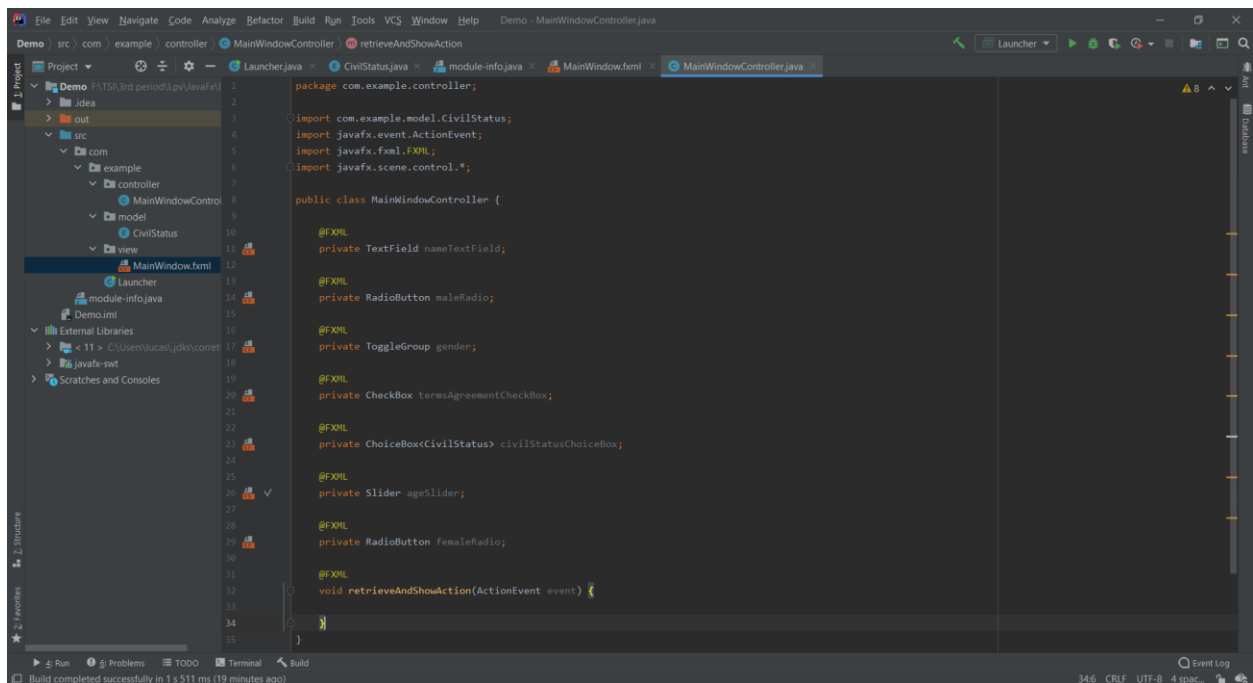


Figura 46: classe controladora com o código gerado com o Scene Builder.

Vamos configurar o ChoiceBox para que as opções sejam do tipo CivilStatus. Para isso devemos implementar a interface Initializable que nos permite realizar inicializações na view antes de exibi-la.

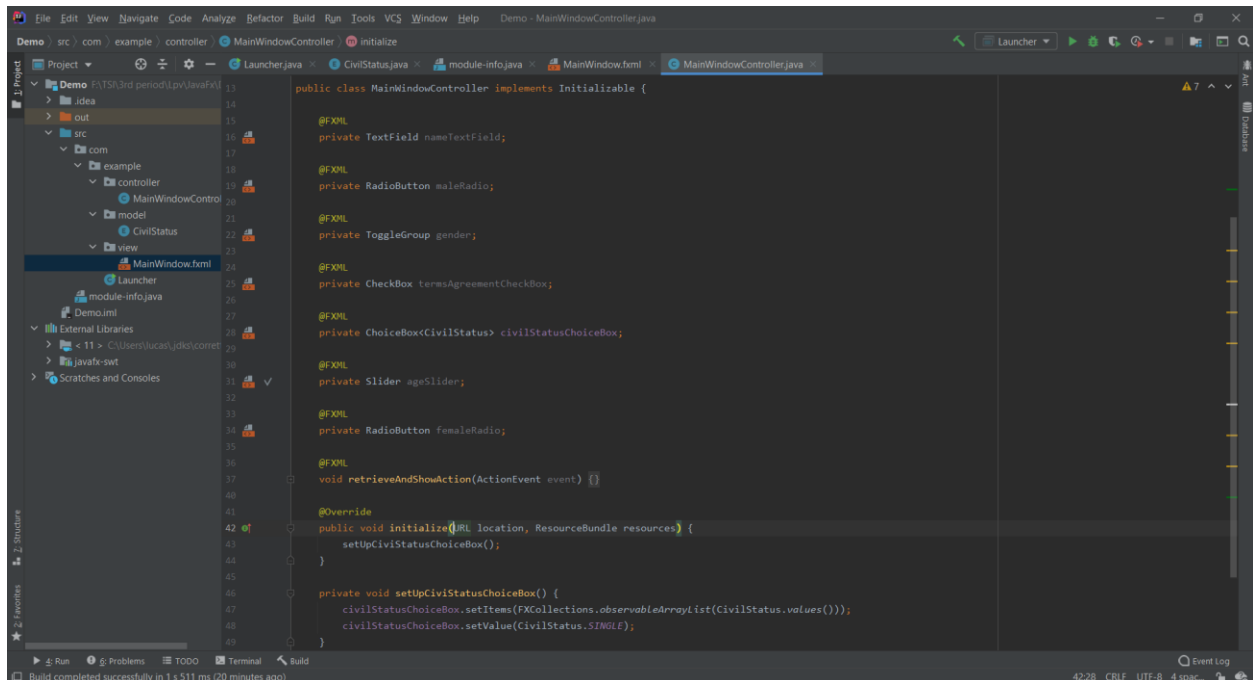


Figura 47: Choice box configurado.

Pronto. Agora podemos realizar a obtenção e exibição do conteúdo da view, mas antes vamos conhecer como pode ser realizada a recuperação dos dados:

- **Caixa de texto** – usa-se o método `getText()`;
- **Slider** – usa-se o método `getValue()`;
- **RadioButton** – usa-se o método `isSelected()`;
- **ChoiceBox(Lista suspensa)** – usa-se o método `getSelectionModel().getSelectedItem()`;
- **CheckBox** - usa-se o método `isSelected()`.

Logo, para recuperar os dados de nossa view, teremos o seguinte trecho de código:

```
@FXML
void retrieveAndShowAction(ActionEvent event) {
    var name :String = nameTextField.getText();
    var age :double = ageSlider.getValue();
    var gender :String = femaleRadio.isSelected() ? "Female" : "Male";
    var civilStatus :CivilStatus = civilStatusChoiceBox.getSelectionModel().getSelectedItem();
    var acceptedTerms :boolean = termsAgreementCheckBox.isSelected();
}
```

Figura 48:obtendo dados da interface e armazenando em variáveis locais.

Uma vez que a recuperação foi realizada, podemos concatenar estes dados em uma única String e exibir usando um Alert.

```
@FXML
void retrieveAndShowAction(ActionEvent event) {
    var name :String = nameTextField.getText();
    var age :double = ageSlider.getValue();
    var gender :String = femaleRadio.isSelected() ? "Female" : "Male";
    var civilStatus :CivilStatus = civilStatusChoiceBox.getSelectionModel().getSelectedItem();
    var acceptedTerms :boolean = termsAgreementCheckBox.isSelected();

    var message :String = String.format("Nome: %s\nIdade: %1.0f\nSexo: %s\nEstado civil: %s\nAceitou os termos: %s",
        name, age, gender, civilStatus, acceptedTerms);

    showData(message);
}

private void showData(String message) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Dados obtidos");
    alert.setHeaderText(null);
    alert.setContentText(message);

    alert.showAndWait();
}
```

Figura 49: concatenando e exibindo os dados via Alert.

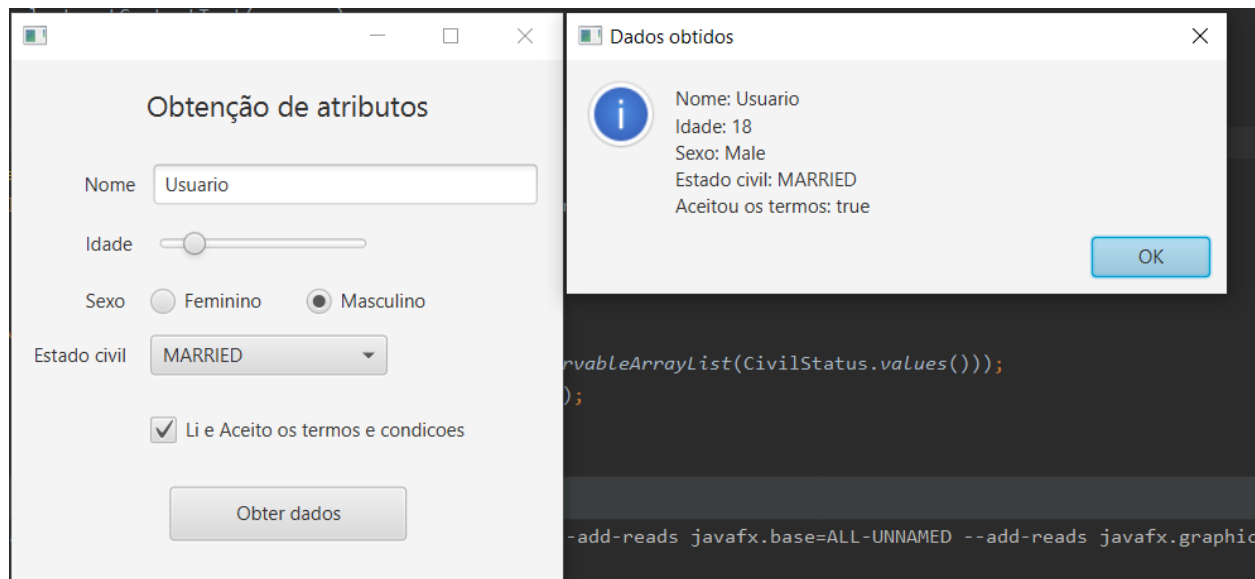


Figura 50: Exemplo de dados recuperados.

Nota: O código dessa demonstração pode ser encontrado nesta [página do github](#).

Tratamento básico de eventos

Veremos agora, como tratar eventos nos elementos.

Clique no botão – Basta adicionar um evento *onAction* e definir o que será realizado.

<ENTER> no input – Para iniciar uma ação após o usuário pressionar uma tecla no input, basta adicionar o evento *On Key Pressed*. Este evento é disparado toda vez que o usuário aperta uma tecla tendo o input selecionado. Este evento é disparado sobre qualquer tecla, logo para realizar uma ação somente quando o usuário teclar ENTER, deverá ser realizada uma checagem.

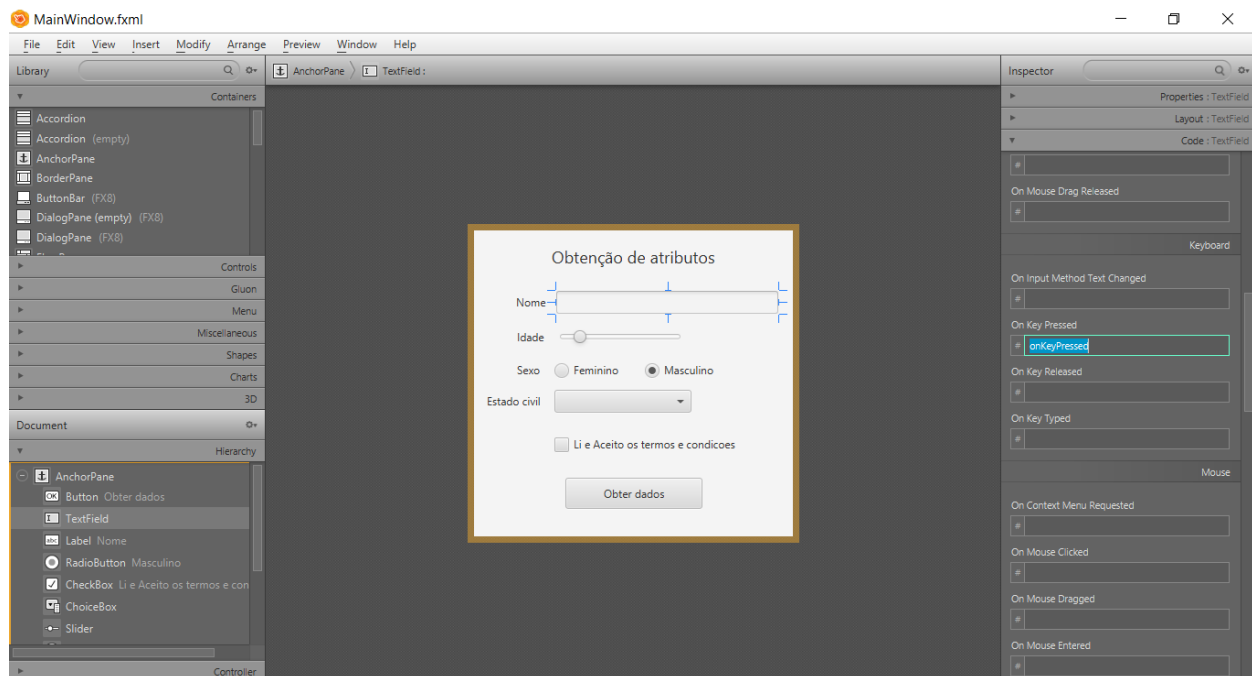


Figura 51: Associando evento de tecla pressionada ao text field via Scene builder – passo 1

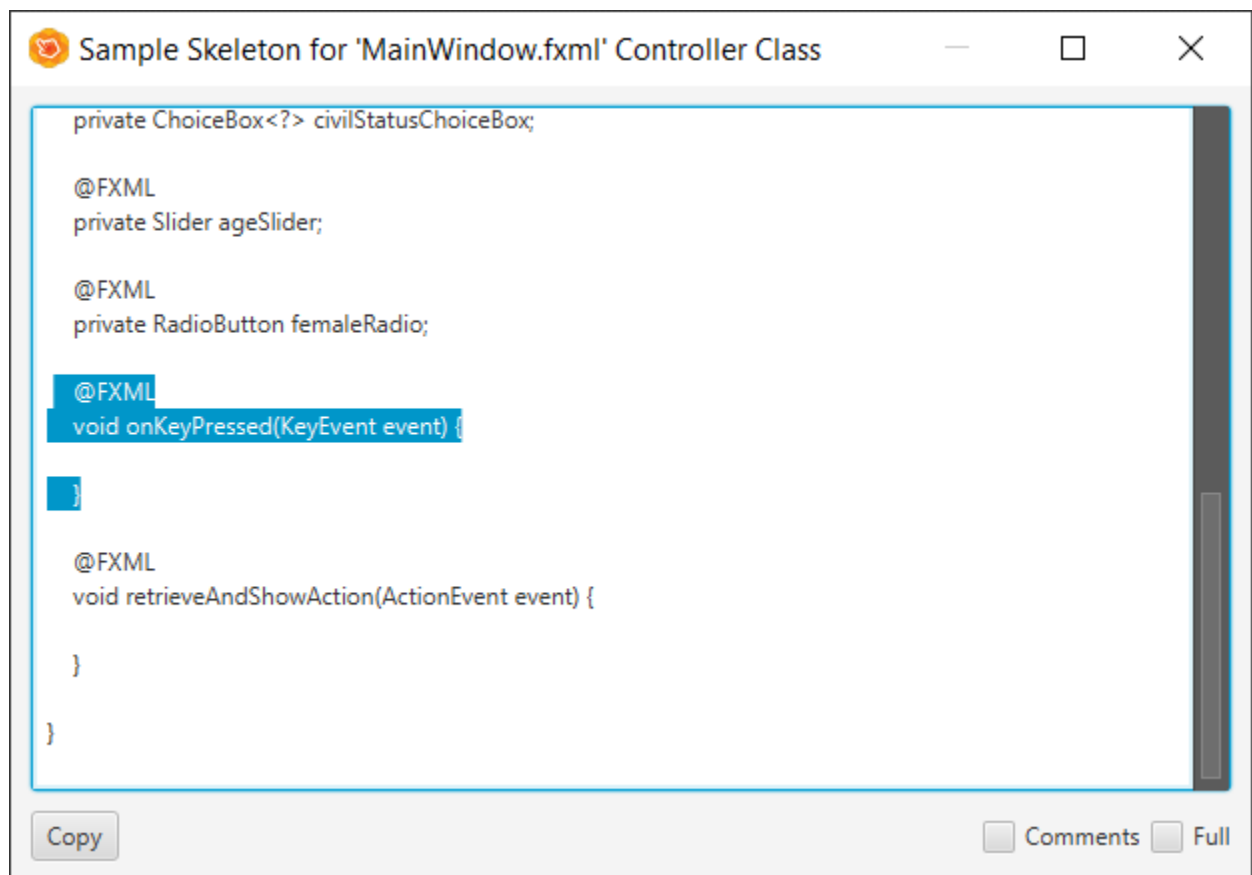


Figura 51: Associando evento de tecla pressionada ao text field via Scene builder – passo 2

```

@FXML
void onKeyPressed(KeyEvent event) {
    if(event.getCode() == KeyCode.ENTER) // crie sua ação
        System.out.println("Enter pressed");
}
}

```

Figura 51: Definindo ação a ser executada ao pressionar a tecla ENTER.

Marcar/desmarcar radio/checkbox Você pode definir o evento de alteração de valor via código Java, adicionando um ouvinte de mudança.

```

termsAgreementCheckBox.selectedProperty().addListener((observable, oldValue, newValue) ->
    System.out.println("Aceitou os termos? " + newValue));

femaleRadio.selectedProperty().addListener((observable, oldValue, newValue) ->
    System.out.println("Feminino? " + newValue));

```

Alteração do valor da lista suspensa (ChoiceBox) Assim como no checkBox e RadioButton, pode se definir o evento de alteração de valor para um ChoiceBox via código Java. É possível disparar o evento baseando se no índice, ou item da lista.

```

civilStatusChoiceBox.getSelectionModel().selectedItemProperty().addListener((observable, oldValue, newValue) -> {
    System.out.println("Novo item selecionado: " + newValue);
});

civilStatusChoiceBox.getSelectionModel().selectedIndexProperty().addListener((observable, oldValue, newValue) -> {
    System.out.println("Novo índice selecionado: " + newValue);
});

```

Alteração do valor do slider O evento de alteração pode ser definido através do código Java baseando-se no mouse ou na propriedade de valor.

```

ageSlider.valueProperty().addListener((observable, oldValue, newValue) ->
    System.out.println("New age:" + newValue));

ageSlider.setOnMouseReleased(event -> {
    System.out.println(ageSlider.getValue());
});

```

Nota: O código para o tratamento de eventos básicos pode ser encontrado [aqui](#).

Exemplo prático básico:

- Janela gráfica com um slider de 0 a 100 e um botão;
- O usuário marca no slider a nota que ele acredita que um teste de usabilidade
- System Usability Scale irá dar (hipótese);
- Depois, clica no botão e seleciona o CSV gerado por um formulário de entrevista do SUS;
- O score SUS é calculado pelo programa, e é exibido na interface o resultado, e o quanto a hipótese (marcada no slider) se aproximou do mesmo.

Os códigos dos projetos implementados para o desenvolvimento dessa apostila podem ser encontrados nos links:

<https://github.com/lucaschf/DemoJavaFx/commit/ea7c661b92d5601bc70f0e79a028e51c5044663>

<https://github.com/lucaschf/SusScoreCalculator>

<https://github.com/lucaschf/EmailClient>