Programação Orientada a Objetos

Java FX, FXML, Scene Builder

Prof. Gabriel M. Alves

May 5, 2023

versão: `70d70cc`

Sumário

Introdução

- Projetos podem conter vários arquivos, de código e de configurações. Além disso, podem apresentar uma extensa estrutura de diretórios para melhor organização. Um exemplo, são projetos que utilizam JavaFX.
- Existem diferentes ferramentas e recursos que auxiliam os desenvolvedores a organizar melhor os projetos.
- Exemplos de ferramentas: Apache Maven, Gradle e Apache Ant.
- Neste sentido, a comunidade Java desenvolveu o Projeto Jigsaw que tinha por objetivo modularizar a plataforma Java.
- O resultado deste projeto é conhecido como Java Plataform Module System (JPMS).

Java Módulos

- O objetivo do JPMS é facilitar a criação, manutenção e distribuição de aplicativos java.
- Para isso, o JPMS possibilita dividir a plataforma Java em módulos menores e mais independentes.
- Assim é possível definir dependências entre módulos e melhorar segurança.
- É importante observar que os módulos também promovem a característica de *encapsulamento*.

Java Módulos

- Ao dividir o código em módulos, a aplicação pode se tornar mais fracamente acoplada e menos monolítica.
- Ao alcançar um bom encapsulamento, um módulo pode ser exposto como API (*Application Programming Interface*) de maneira segura para outros módulos.
- Antes do JPMS, o termo "*JAR Hell*" era frequentemente utilizado para se referir a resolução do *classpath* e das dependências.
- Uma das motivações para o surgimento do Apache Maven foi para minimizar o *JAR Hell*.

Java Módulos

- Em orientação a objetos, a classe é a unidade fundamental de código.
- Uma classe provê encapsulamento.
- Um pacote é um conjunto de classes que, geralmente, estão relacionadas entre si.
- Um pacote provê encapsulamento em um nível maior que uma classe.
- Em sistemas maiores podem conter diversos pacotes e, mesmo assim, esse nível de controle de acesso pode não ser suficiente.
- Um módulo é um conjunto de pacotes, que provê um nível de encapsulamento maior que a de um pacote.

Java Módulos

- Um módulo consiste de:
  - uma coleção de pacotes;
  - opcionalmente, arquivos adicionais como bibliotecas, imagens e arquivos de configurações;
  - uma lista de pacotes acessíveis dentro do módulo;
  - uma lista de módulos dos quais ele depende.

Vantagens

- Forte encapsulamento: possibilidade de controlar quais pacotes serão acessíveis e não precisa se preocupar com código que não será consumido externamente.
- Configuração confiável: evita problemas de *classpath*.

Java Módulos

- *Definição do módulo*: é um arquivo de configuração que lista as dependências e exporta o módulo.
- Abaixo a estrutura básica de uma definição de módulo que, geralmente, fica no arquivo `module-info.java`.

```
1 module <nome do módulo> {
2     requires [public] <módulo dependente>;
3     exports <package name>;
4 }
```

Java Módulos

- Exemplo de definição de módulo usando JavaFX

```
1  module br.edu.ifsp.sbv.hellofxml {
2      requires javafx.controls;
3      requires javafx.fxml;
4      requires java.base;
5
6      opens br.edu.ifsp.sbv.hellofxml to javafx.fxml;
7      exports br.edu.ifsp.sbv.hellofxml;
8  }
```

Java Módulos

- Nome do módulo: a sugestão é utilizar o domínio reverso, como geralmente é feito com pacotes.
- `requires`: essa palavra-chave refere-se a declarar as dependências de um módulo. A palavra-chave `public` permite o acesso implícito entre módulos (*implied readability*).
- `exports`: expõe o módulo para ser acessado externamente.
- `opens` e `to`: um pacote torna-se "aberto" apenas para o módulo informado. Todos os pacotes estão fechados por padrão.

Recursos de JavaFX

- Um programa JavaFX é chamado de application, pois é uma subclasse que estende a superclasse `Application`.
- JavaFX fornece duas maneiras de criar interfaces gráficas:

  Programática todos os componentes da interface gráfica são programados pelo desenvolvedor por meio de recursos disponibilizados pela plataforma JavaFX.

  Declarativa todos os componentes da interface gráfica são declarados. A plataforma JavaFX utiliza a FXML que é uma linguagem de marcação própria para declarar os elementos da interface gráfica. O arquivo em formato FXML pode ser gerado pela ferramenta Scene Builder.

Recursos de JavaFX

- Relembrando outros termos associados a JavaFX:

  stage janela na qual estão todos os elementos da aplicação.

  scene uma cena em que estão os conteúdos da *stage*, em forma de grafo.

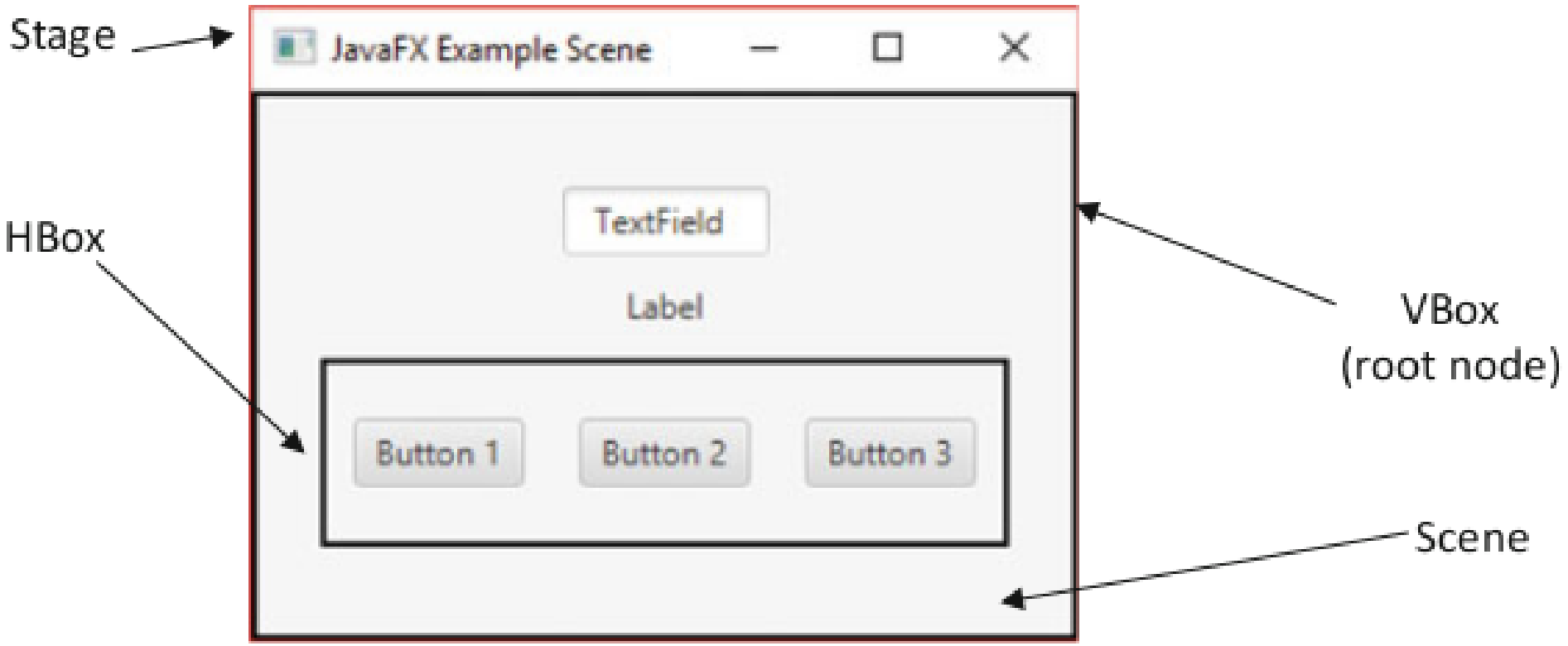
  nodes itens presentes em uma cena.

  controls componentes comuns que permitem interação com usuário (botões, campos, rótulos, etc).

  containers componentes que reúnem outros *nodes*. Permite organizar uma hierarquia de componentes.

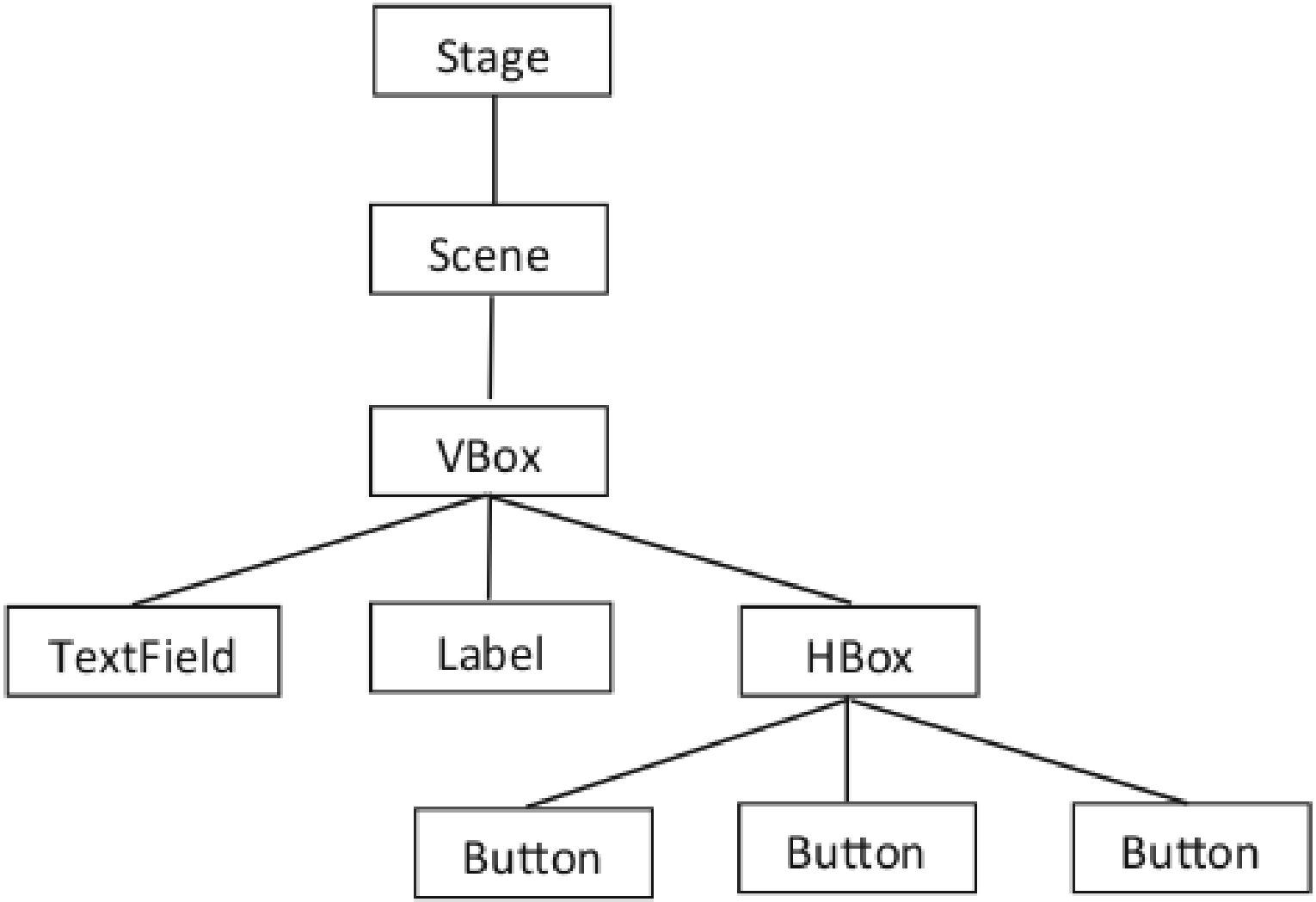
- Normalmente colocamos um *node* principal e o chamamos de root (raiz). Utilizamos os termos parent e children para referenciar os objetos dentro de uma hierarquia.

Recursos de JavaFX



Recursos de JavaFX

- Hierarquia dos componentes da interface gráfica apresentada.



Recursos de JavaFX

- JavaFX baseia se no modelo de programação *Node-centric UIs*, no qual uma interface consiste em um conjunto de nós. Cada nó representa um elemento da interface e esses nós são organizados em uma hierarquia.
  - Vantagens flexibilidade, facilidade para incluir e remover nós, escalabilidade.
- JavaFX usa a metáfora do teatro, portanto criar uma interface gráfica consiste nos seguintes passos:
  - Criar o palco (*stage*) em que sua aplicação irá rodar (desktop, celular, etc?)
  - Criar uma cena (*scene*) na qual os "atores" (componentes) irão visualmente interagir com a platéia (usuários);
  - Criar os nós da cena, ou seja, os elementos da interface gráfica;
  - Criar variáveis e classes que representam o modelo para os nós da cena;
  - Criar eventos que permitem os usuários interagirem com a interface gráfica;
  - Criar *timelines* e *transitions* que permitem realizar a animação da cena.

Recursos de JavaFX

- Quando uma aplicação JavaFX inicia, três métodos são chamados na seguinte ordem:
  - `void init()` : método em que são programadas rotinas necessárias e que devem ser executadas antes da aplicação iniciar.
  - `abstract void start(Stage stage)`: método no qual a aplicação é programada e iniciada. É importante observar que, se necessário, métodos auxiliares podem ser desenvolvidos e invocados dentro deste método.
  - `void stop()` : método em que são executados códigos após o encerramento da aplicação.
- A classe `Application` possui o método estático `launch()` a qual deve ser invocada a partir do `main()`:

```
1 Application.launch(args); // Opção 1
2 Application.launch(MinhaTela.class, args); // Opção 2
```

Recursos de JavaFX

- Os principais componentes da plataforma JavaFX são:
  - biblioteca de gráficos: responsável por renderizar gráficos na tela, como formas, imagens, textos e animações. Ela permite a transformação de imagens, efeitos visuais e interação com usuários.
  - biblioteca de controles: fornece uma série de componentes prontos como botões, caixas de texto, listas, tabelas, menus, etc.
  - sistema de layout: permite que os desenvolvedores definam como os elementos serão organizados na interface gráfica.
  - biblioteca de mídia: permite a reprodução de áudio e vídeo em uma interface gráfica.
  - API de acesso a dados: permite que os desenvolvedores acessem e manipulem dados em diversas fontes, como banco de dados e serviços web.
- JavaFX ainda interage com IDEs como Netbeans e Eclipse e a ferramenta SceneBuilder que permite a programação declarativa da interface gráfica.

Programação declarativa com FXML

- FMXL (*FX Markup Language*) é uma linguagem de marcação baseada em XML. Exemplo de código *fxml*:

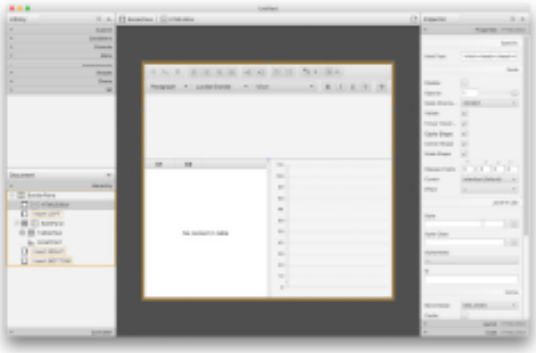
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?import javafx.scene.control.Button?>
3  <?import javafx.scene.control.Label?>
4  <?import javafx.scene.layout.VBox?>
5
6  <VBox>
7      <children>
8          <Label text="Primary View" />
9          <Button fx:id="primaryButton"
10                 onAction="#switchToSecondary"
11                 text="Switch to Secondary View" />
12     </children>
13 </VBox>
```

Programação declarativa com FXML

Importante

O código FXML é separado da lógica do programa que é definida no código-fonte Java. Tal separação entre interface gráfica e implementação faz com que seja mais fácil depurar, modificar e manter aplicações GUI do JavaFX.

Programação declarativa com FXML



Scene Builder

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>

<VBox alignment="CENTER" prefHeight="217.0" prefWidth="227.0" spacing="20.0"
      xmlns="http://javafx.com/javafx/19" xmlns:fx="http://javafx.com/fxml/1"
      fx:controller="br.edu.ifsp.sbv.hellofxml.PrimaryController">
    <children>
        <Label text="Primary View" />
        <Button fx:id="primaryButton" onAction="#switchToSecondary"
                text="Switch to Secondary View" />
        <Button fx:id="buttonTerceiraTela" mnemonicParsing="false"
                onAction="#IrParaTerceiraTela" text="Ir para Terceira Tela" />
    </children>
    <padding>
        <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
    </padding>
</VBox>
```

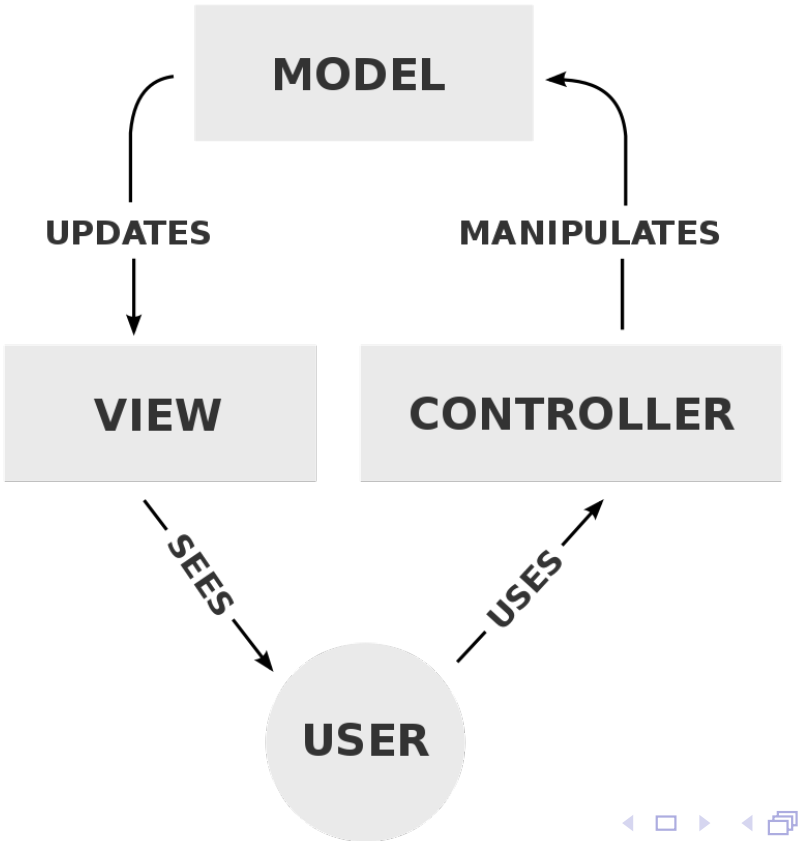
FXML

+

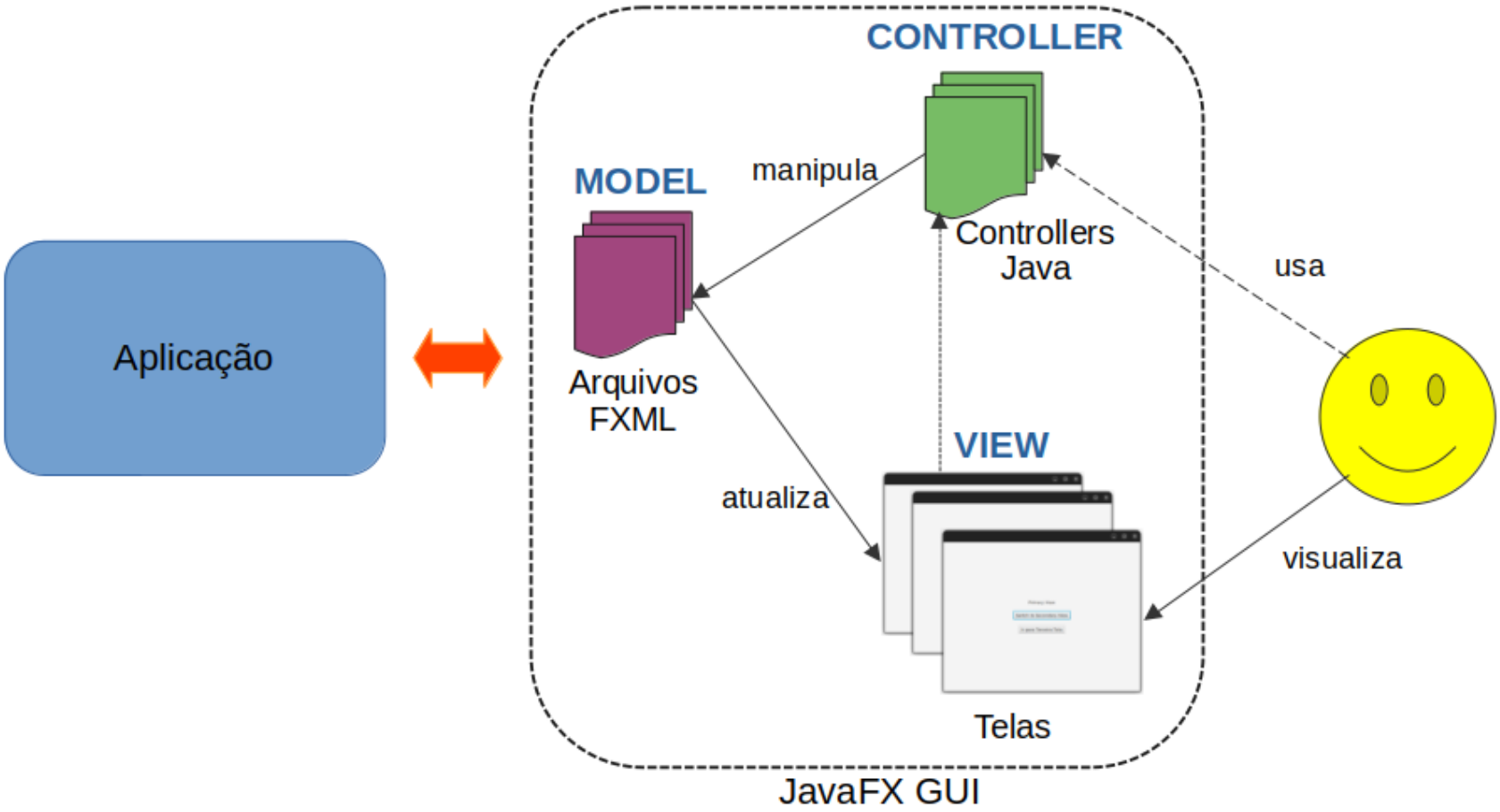
JavaFx

Programação declarativa com FXML

- Os aplicativos JavaFX seguem o padrão de design Model-View-Controller (MVC), que separa os dados de um aplicativo contidos no modelo de um lado e, de outro, a GUI do aplicativo (a visualização) e a lógica de processamento do aplicativo (o controlador).



Programação declarativa com FXML



Programação declarativa com FXML

- Uma classe *controller* é responsável por acessar e manipular arquivos `.fxml`
- Ela gerencia a interação do usuário com a interface gráfica
- Ela executa a lógica da aplicação de acordo com as interações do usuário
- Geralmente o tratamento de eventos será implementado em uma classe *controller*

Programação declarativa com FXML

- Exemplo de vinculação entre um arquivo `.fxml` e uma classe *controller*:

```
1  <AnchorPane id="AnchorPane" fx:id="rootPane" prefHeight="400.0"
2              prefWidth="600.0" xmlns="http://javafx.com/javafx/19"
3              xmlns:fx="http://javafx.com/fxml/1"
4              fx:controller="br.edu.ifsp.sbv.gui.PrimeiraTelaController">
5    <!-- demais declarações -->
6  </AnchorPane>
```

- Atributo `fx:controller` associa `.fxml` e *controller*.
- Atributo `fx:id` define um identificador que poderá ser manipulado no código da classe *controller*.

Programação declarativa com FXML

- Trecho de uma classe *controller*

```
1  // a implementação da interface é opcional
2  public class PrimeiraTelaController implements Initializable {
3
4      @FXML
5      private AnchorPane rootPane; //fx:id="rootPane"
6
7      @Override
8      public void initialize(URL url, ResourceBundle rb){
9          // código utilizado para inicializar e configurar
10         // as propriedades da classe ou para executar outras
11         // operações que sejam necessárias antes da interface
12         // gráfica ser exibida para o usuário.
13     }
14
15 }
```

Programação declarativa com FXML

- A interface `Initializable` define o método `initialize()` que é chamado automaticamente após carregas os elementos da interface gráfica definidos no arquivo `.fxml`.
- Um ponto de destaque em uma classe *controller* é a anotação @FXML.
- A anotação (Java Annottaion) pertence ao pacote `jafavx.fxml`.
- Ao definir um elemento no Scene Builder e definir a propriedade `fx:id`, ele se tornará acessível no código como atributo da classe *controller* se receber a anotação @FXML.
- Métodos da classe *controller* que recebem a anotação são usados, geralmente, para tratar eventos da interface gráfica.
- Importante mencionar que a classe *controller* pode conter outros métodos que não recebem tal anotação.

Programação declarativa com FXML

- Como ocorre essa mágica com o @FXML? Injeção de dependência
- A "*injeção de dependência*" é um conceito utilizado em programação que visa a reduzir o acoplamento entre as classes de um sistema.
- Ao invés da classe criar ou instanciar um objeto que ela precisa para realizar alguma tarefa, ela recebe esse objeto (dependência).
- Neste caso a dependência não precisa saber como criar ou instanciar essa dependência.
- Em Java, a injeção de dependências pode ser realizada com o uso de anotações. Tais anotações são definidas pelo *framework* de injeção dependência utilizado no projeto.

Programação declarativa com FXML

Programação declarativa com FXML

- A classe `FXMLLoader` é responsável por ler um arquivo `.fxml` em tempo de execução.
- Além de ler, a classe é responsável por realizar o *parsing* e criar os elementos que compõe a interface gráfica.

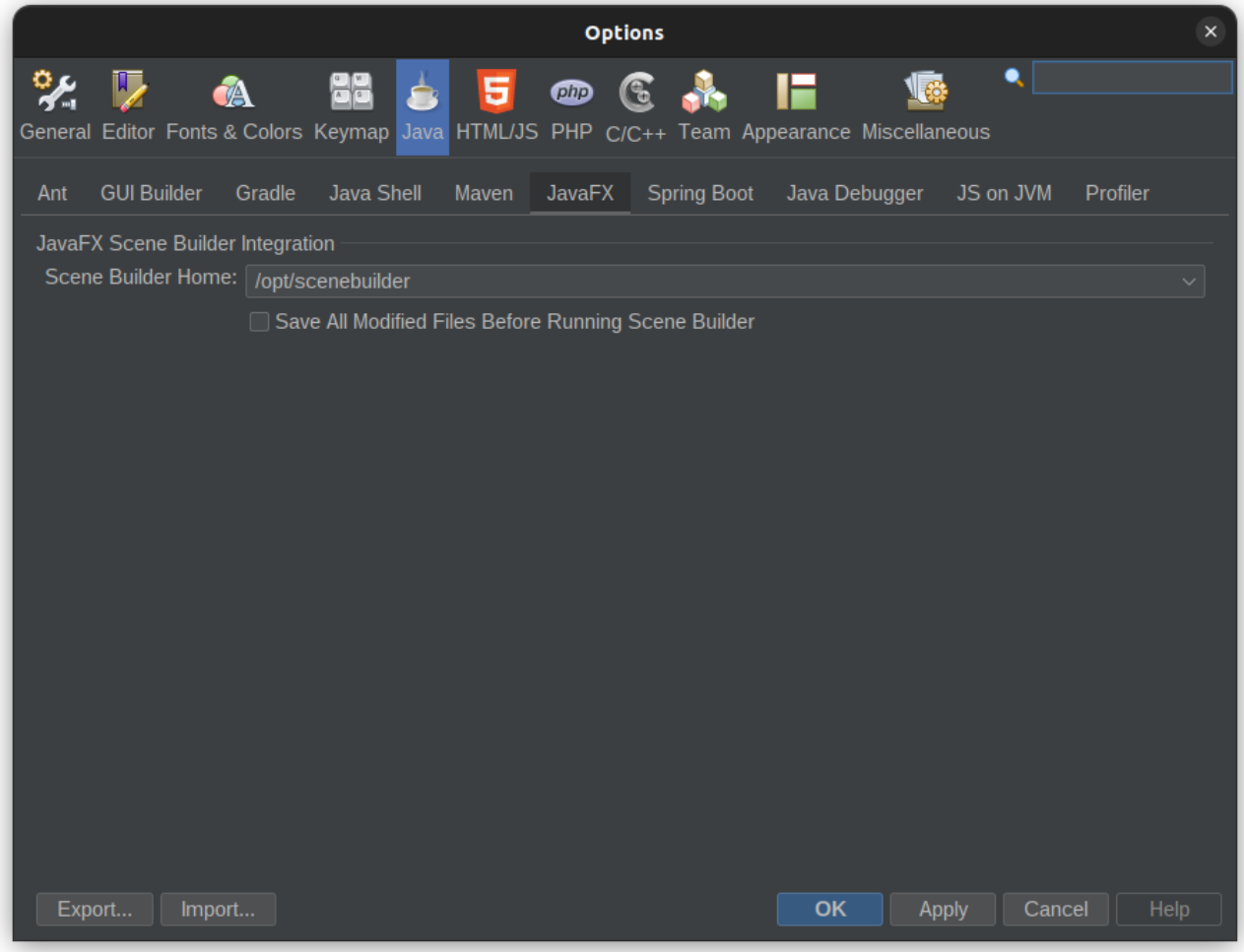
```
1  @Override
2  public void start (Stage stage) throws IOException {
3      URL url = getClass().getResource("PrimeiraTela.fxml");
4      Parent raiz = FXMLLoader.load(url);
5      Scene cena = new Scene(raiz);
6
7      stage.setTitle("Planeta Terra");
8      stage.setScene(cena);
9      stage.show();
10 }
```

Scene Builder

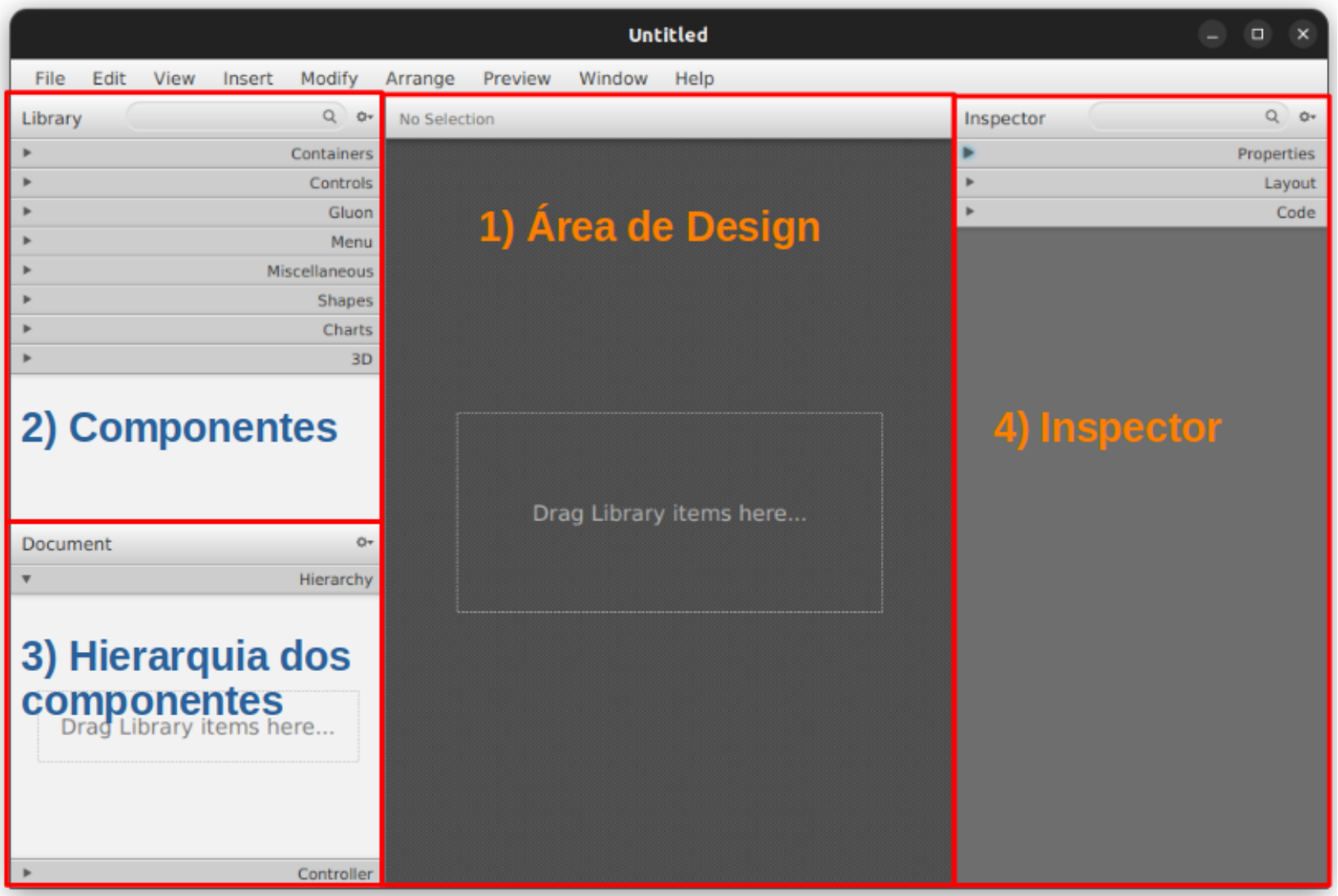
- A ferramenta Scene Builder é livre e *open source* suportada pela empresa Gluon.
- Scene Builder trabalha com todo o ecossistema JavaFX.
- É uma ferramenta *standalone* que permite criar interfaces gráficas (GUI) por meio do recurso "*drag-and-drop*", ou seja, arrastar e soltar componentes em uma área de *design*.
- Utilizar o SceneBuilder permite criar interfaces gráficas de forma *declarativa*, pois ele gera arquivos FXML.
- A ferramenta integra-se com IDEs como Netbeans e Eclipse.

Scene Builder

- Para integrar o Scene Builder com o Netbeans, é preciso acessar o menu "*Tools* > *Options*".



Scene Builder

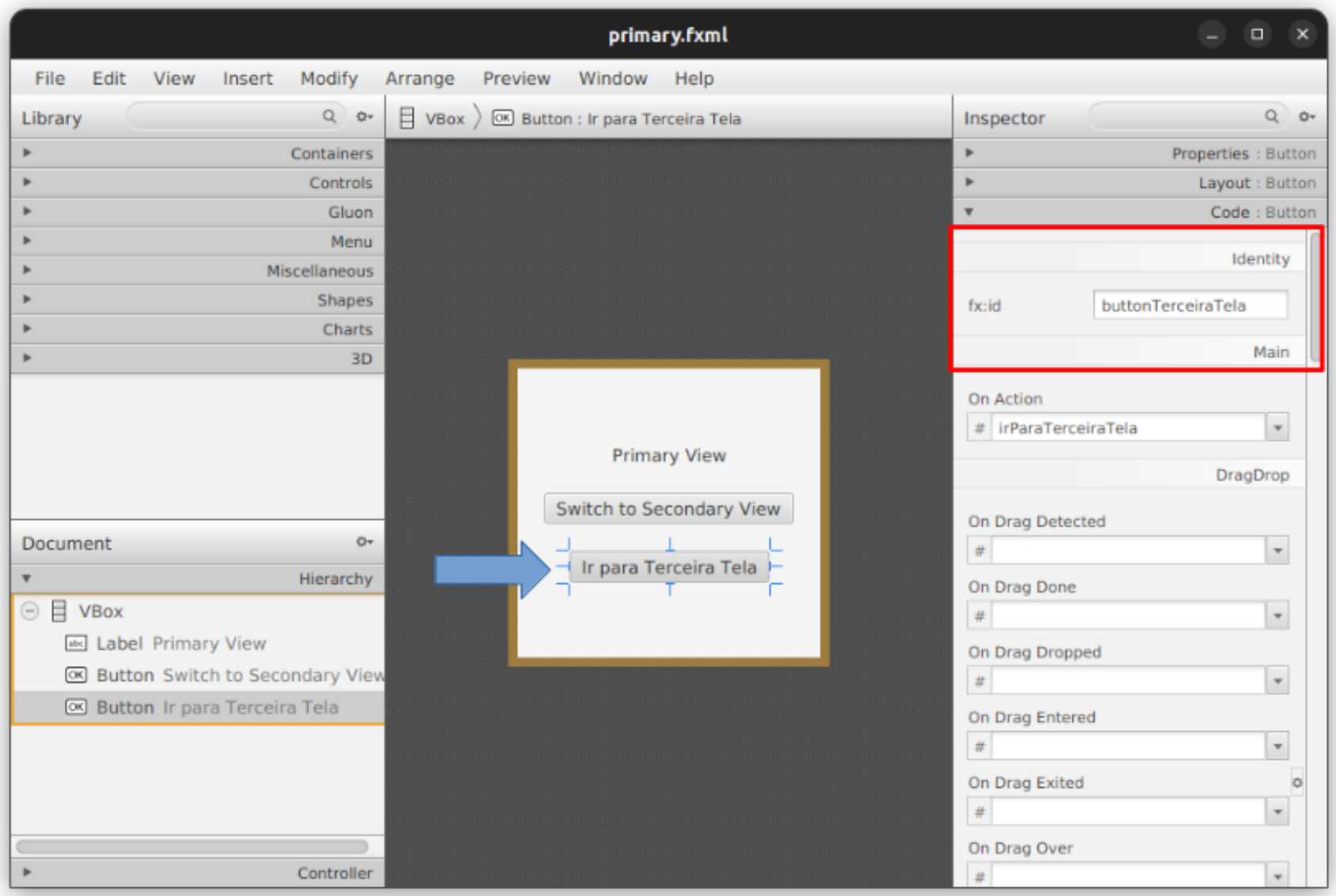


Scene Builder

- O Scene Builder gera a interface gráfica a partir da declaração dos elementos que a compõe.
- As entradas e eventos produzidos pelo usuário na interface gráfica são processados pelas classes *controllers*.
- Portanto, é necessário definir no Scene Builder um identificador para cada elemento da interface.
- Ao definir um identificador é gerado o atributo `fx:id` no elemento

```
1 <!-- trecho que destaca o fx:id -->
2 <Button fx:id="buttonTerceiraTela" mnemonicParsing="false"
3                 onAction="#irParaTerceiraTela"
4                 text="Ir para Terceira Tela" />
```

Scene Builder



Organização de um projeto JavaFX

- Utilizar o Apache Maven evita a necessidade de instalar o JavaFX JDK.
- Portanto, a proposta é organizar a aplicação em um Projeto Maven.
- O Apache Netbeans 17 oferece três, principais, tipos de projetos Maven:

  Java Application cria um Projeto Maven de cunho geral (exige muitas configurações para usar JavaFX)

  Simple JavaFX Maven Archetype cria um Projeto Maven para aplicações JavaFX sem uso de FXML.

  FXML JavaFX Maven Archetype cria um Projeto Maven para apicações JavaFX com uso de FXML.

Organização de um projeto JavaFX

- A recomendação é criar o projeto a partir do "*FXML JavaFX Maven Archetype*"
- Vale lembrar que o projeto cria um módulo (JPMS), que precisa ser levado em conta ao organizar um projeto.
- Ao preparar uma projeto é importante dedicar uma pasta para armazenar recursos.
- Recursos, em geral, são outros arquivos que não sejam de código, por exemplo: imagens, áudios, fxml, fontes, etc.

Organização de um projeto JavaFX

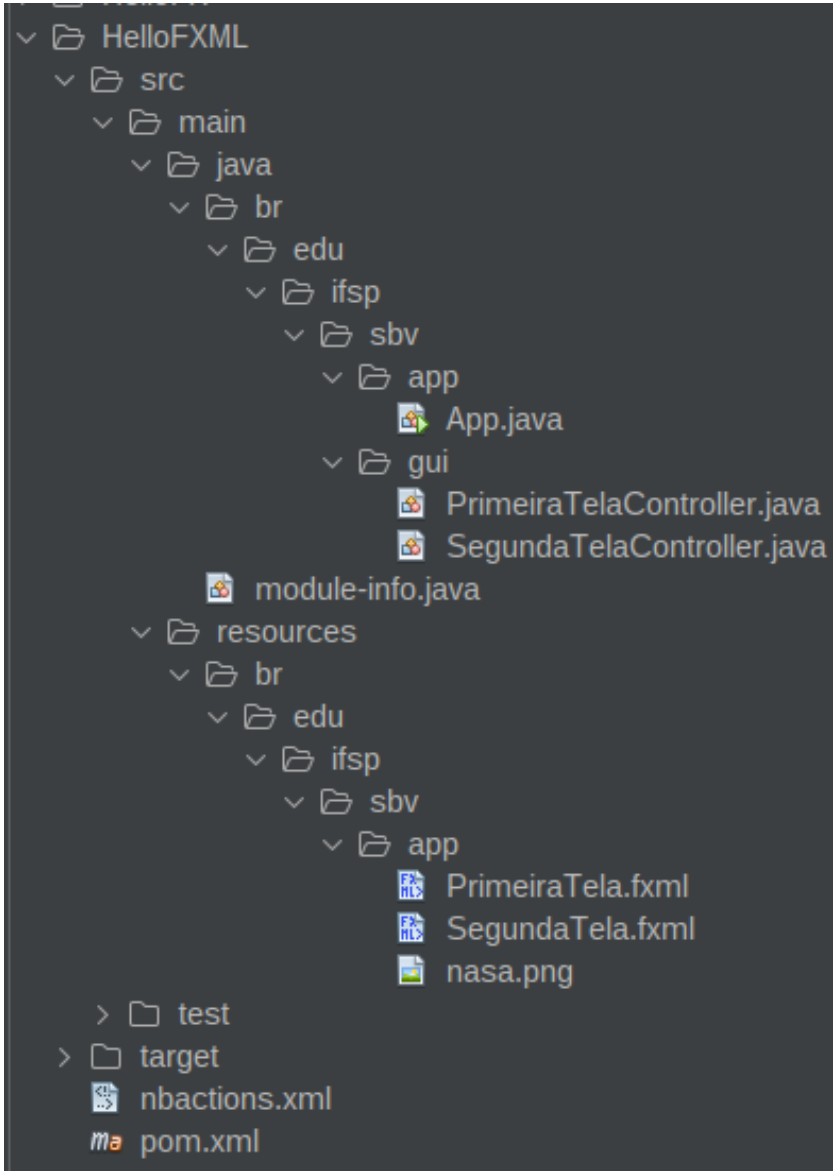
- A estrutura básica de um Projeto é:

```
-- Pasta Raiz do Projeto
    - src
        - main
            - java
                - pacotes da aplicação
                module-info.java
            - resources
        - test
    - target
    nbactions.xml
    pom.xml
```

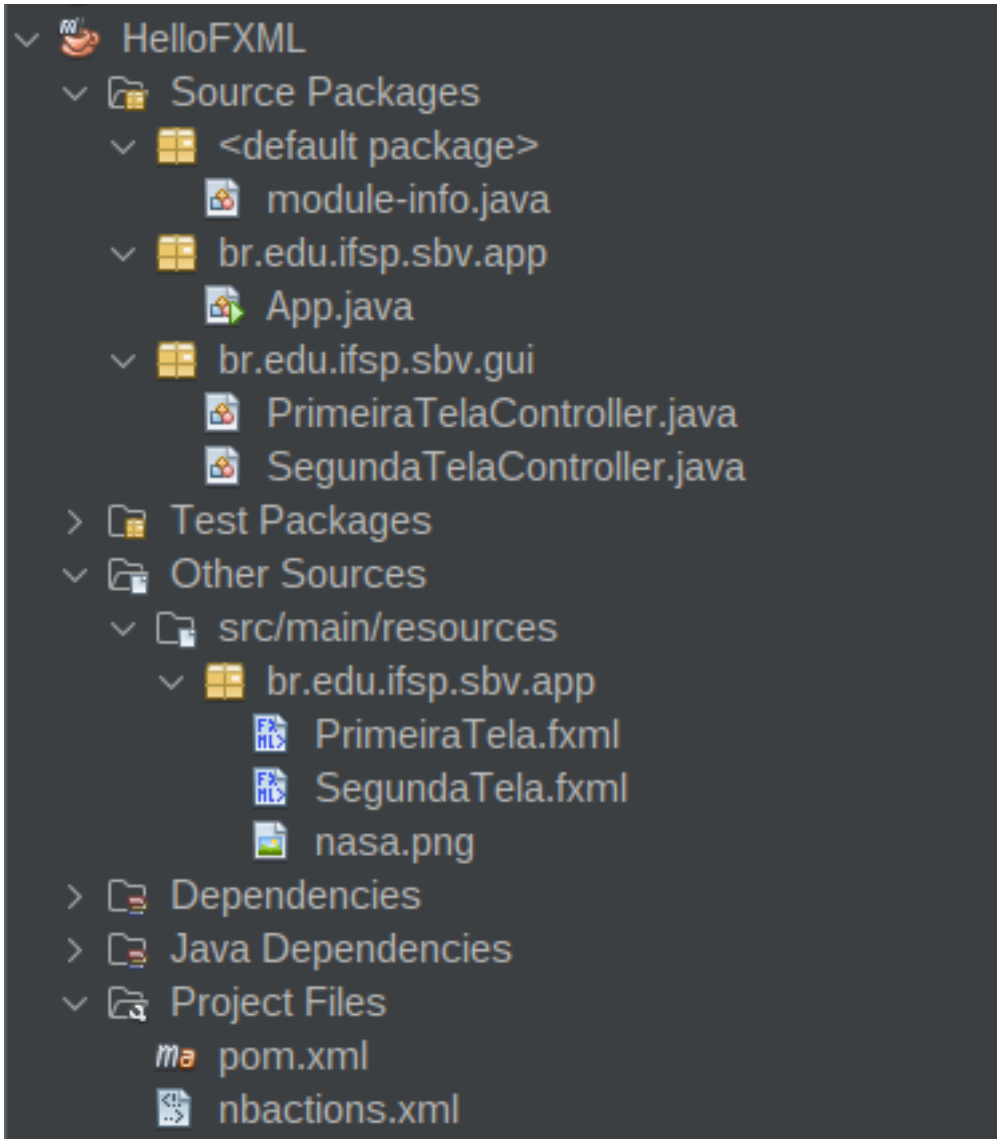
Organização de um projeto JavaFX

- o arquivo `nbactions.xml` são configurações do Apache Netbeans (evite editá-lo diretamente)
- O arquivo `pom.xml` é o arquivo de configuração do Apache Maven
- A pasta `java` contém o arquivo `module-info.java` com as configurações do módulo
- A pasta `java` contém um ou mais pacotes que contém a aplicação
- A pasta `test` contém arquivos de testes eventualmente realizados na aplicação
- A pasta `target` contém arquivos gerados após a compilação
- A pasta `resource` contém os arquivos de recursos
  - No Apache Netbeans deve ser gerada uma estrutura de pastas iguais ao pacote principal da aplicação.

Organização de um projeto JavaFX



Organização de um projeto JavaFX



Organização de um projeto JavaFX

- No exemplo do slide anterior, depois de criar o projeto "FXML JavaFX Maven Archetype", as classes *controllers* foram excluídas.
- O módulo possui dois pacotes: `br.edu.ifsp.sbv.app` e `br.edu.ifsp.sbv.gui`.
- O primeiro pacote contém a classe principal. Para alterar o nome da classe, precisa depois ajustar o arquivo `pom.xml`

```
1 <mainClass>br.edu.ifsp.sbv.app.App</mainClass>
```

- Os arquivos `.fxml` são criados e armazenados na pasta `resources`;
- As classes *controllers* são armazenadas no pacote `br.ifsp.edu.sbv.gui`

Organização de um projeto JavaFX

- Para utilizar as classes *controllers* em outro pacote é preciso ajustar o `module-info.java`

```
1  module br.edu.ifsp.sbv.app {
2      requires javafx.controls;
3      requires javafx.fxml;
4      requires java.base;
5
6      opens br.edu.ifsp.sbv.app to javafx.fxml;
7      // expõe o pacote gui para o módulo javafx.fxml
8      opens br.edu.ifsp.sbv.gui to javafx.fxml;
9
10     exports br.edu.ifsp.sbv.app;
11 }
```

Referências

- Harvey M. Deitel, Paul J. Deitel. Java como programar (Capítulo 25), Pearson Brasil. 10ª edição. 2017.
- Quentin Charatan, Aaron Kans. Java in Two Semesters: Featuring JavaFX, Springer International Publishing. 2019.
- StackOverflow. Learning JavaFX. Disponível em: http://riptutorial.com/ebook/javafx.
- JavaFX: sítio oficial. Disponível em: https://openjfx.io/.
- Oracle. Introduction to FXML. 2013. Disponível em: https://docs.oracle.com/javase/8/javafx/api/javafx/fxml/doc-files/introduction_to_fxml.html. Último acesso em 03 de maio de 2023.
- Oracle. Mastering FXML. 2014. Disponível em: https://docs.oracle.com/javafx/2/fxml_get_started/jfxpub-fxml_get_started.htm. Último acesso em 03 de maio de 2023.

Referências

- Johan Vos, Stephen Chin, Weiqi Gao, James Weaver, Dean Iverson. Pro JavaFX 9: a definitive guide to building desktop, mobile and embedded java clients. 4th edition. 2018.
- Carl Dea, Gerrit Grunwald, José Pereda, Sean Philips, Mark Heckler. JavaFX 9 by example: rich-client applications for any platform. 3th edition. 2017.
- Robert C. Martin. Arquitetura Limpa: o guia do artesão para estrutura e design de software. Rio de Janeiro: Alta Books, 2018.
- Paulo Silveira, Guilherme Silveira, Sérgio Lopes, Guilherme Moreira, Nico Steppat, Fábio Kung. Introdução à arquitetura e design de software: uma visão sobre a plataforma Java. Rio de Janeiro: Elsevier. 2012.

Encerramento

- Dúvidas?
- Comentários?

Contato

Gabriel Marcelino Alves
gabriel.marcelino@ifsp.edu.br