User Interfaces & Code Quality

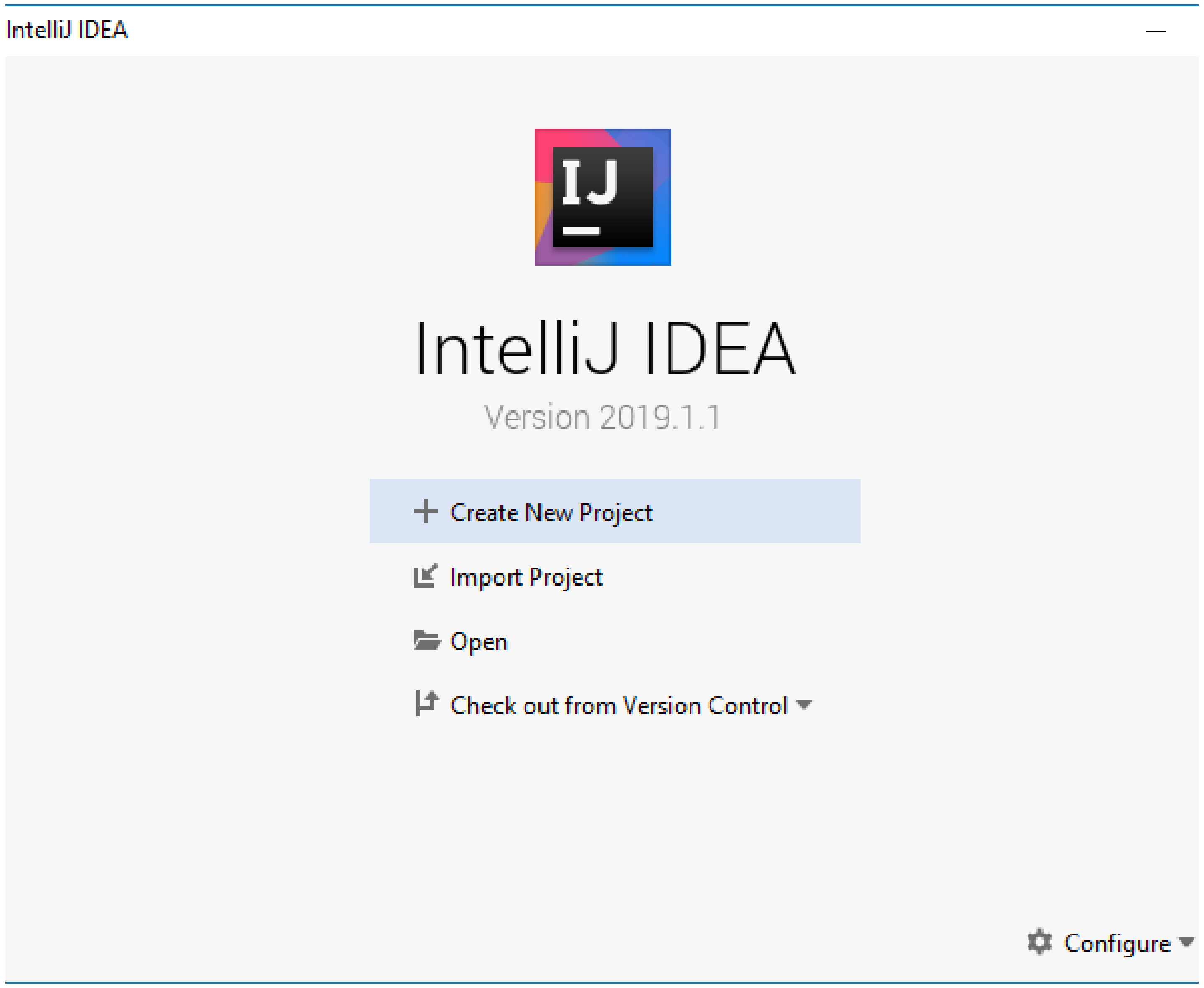
Programming Principles 02 - Design

MINOLI DE SILVA – minoli.d@iit.ac.lk

SULARI FERNANDO– sulari.f@iit.ac.lk

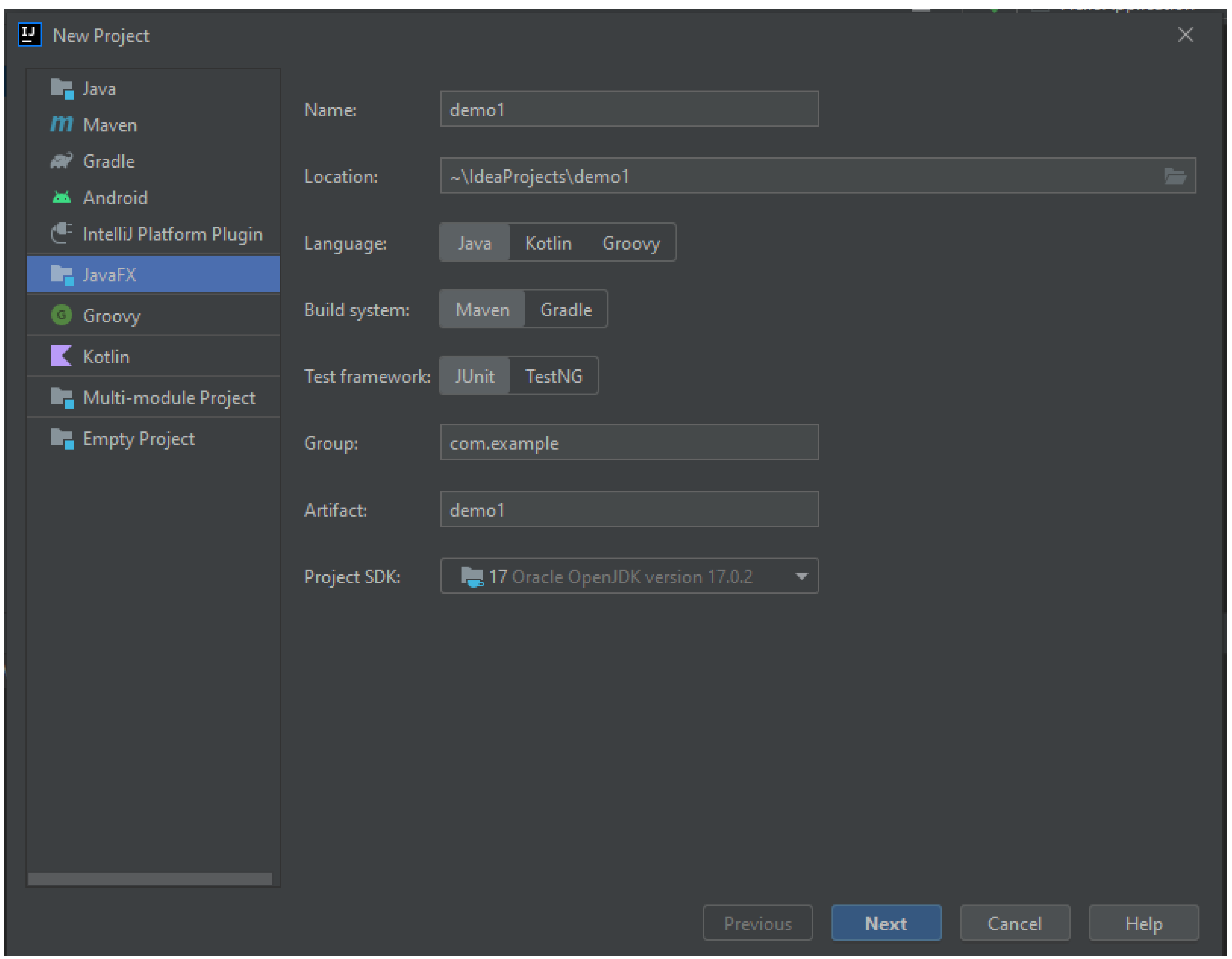
Creating a new JavaFX project

1. Open IntelliJ IDEA
2. Select Create New Project / File -> New -> Project



Selecting the project type

1. Select JavaFX from the sidebar menu
2. Make sure the java JDK is selected as the project SDK
3. If not given, click New.. And give the path to your JDK install location



Name: demo1

Location: ~\IdeaProjects\demo1

Language: Java | Kotlin | Groovy

Build system: Maven | Gradle

Test framework: JUnit | TestNG

Group: com.example

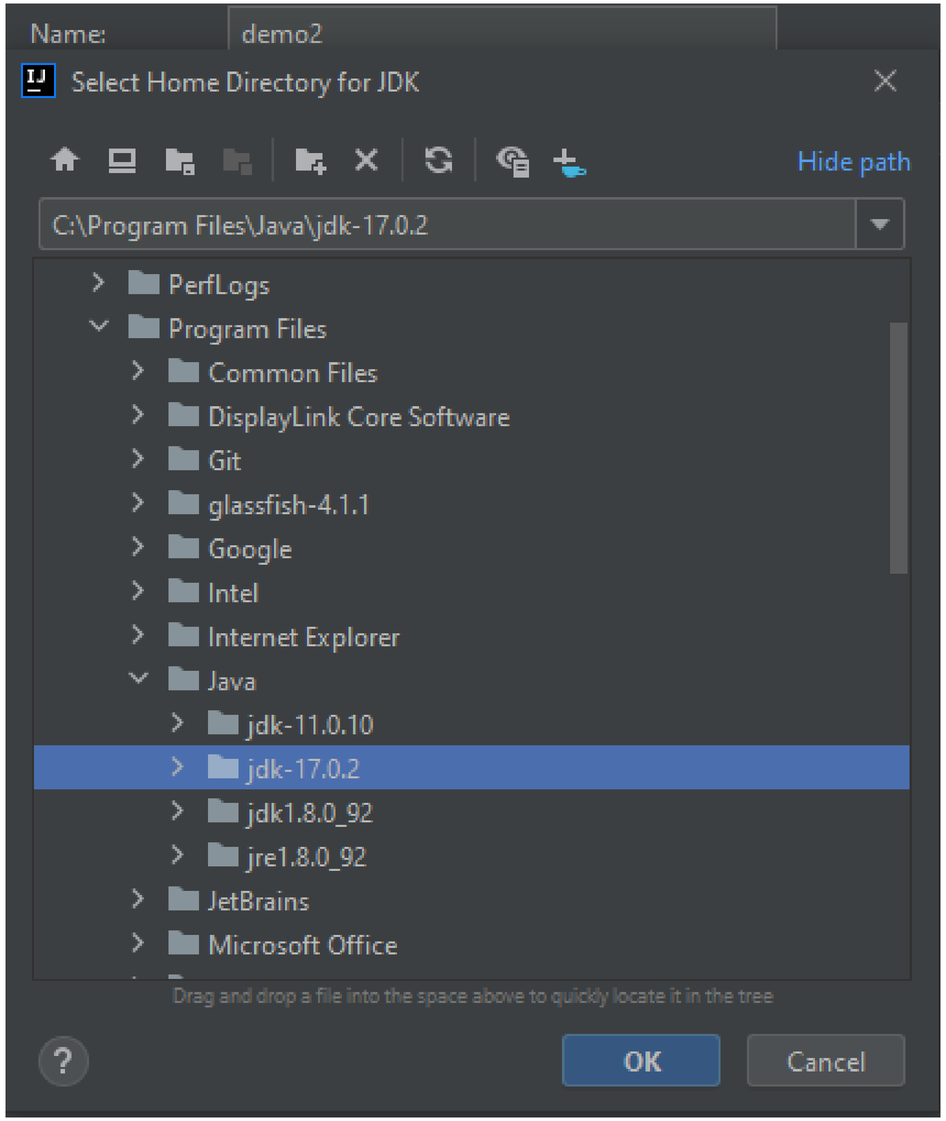
Artifact: demo1

Project SDK: 17 Oracle OpenJDK version 17.0.2

Setting path to JDK

1. Find the install location of the JDK
2. Select the JDK folder and click OK
3. If multiple Java versions are installed, select the preferred version’s JDK folder
4. Default location:

C:\Program Files\Java\jdk17.0.2



Name: demo2

Select Home Directory for JDK

Hide path

C:\Program Files\Java\jdk-17.0.2

- PerfLogs
- Program Files
  - Common Files
  - DisplayLink Core Software
  - Git
  - glassfish-4.1.1
  - Google
  - Intel
  - Internet Explorer
  - Java
    - jdk-11.0.10
    - jdk-17.0.2
    - jdk1.8.0_92
    - jre1.8.0_92
  - JetBrains
  - Microsoft Office

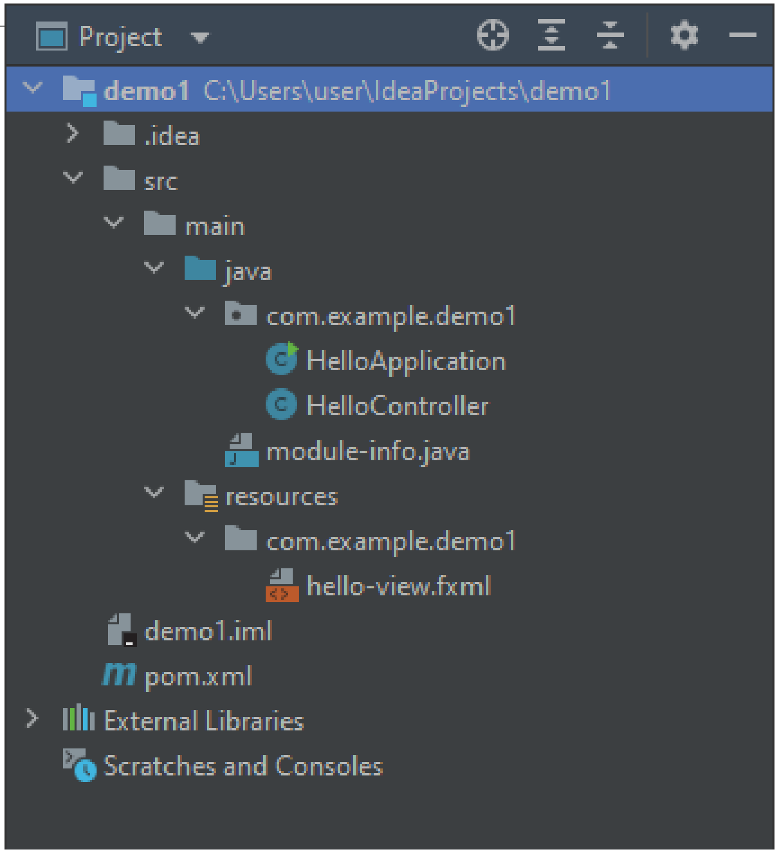
Drag and drop a file into the space above to quickly locate it in the tree

Default File Structure

- The JavaFX project created will have the following by default.
  - hello-view.fxml file
  - HelloApplication.java class
  - HelloController.java class

- .idea folder and .iml file is generated by IntelliJ IDEA.

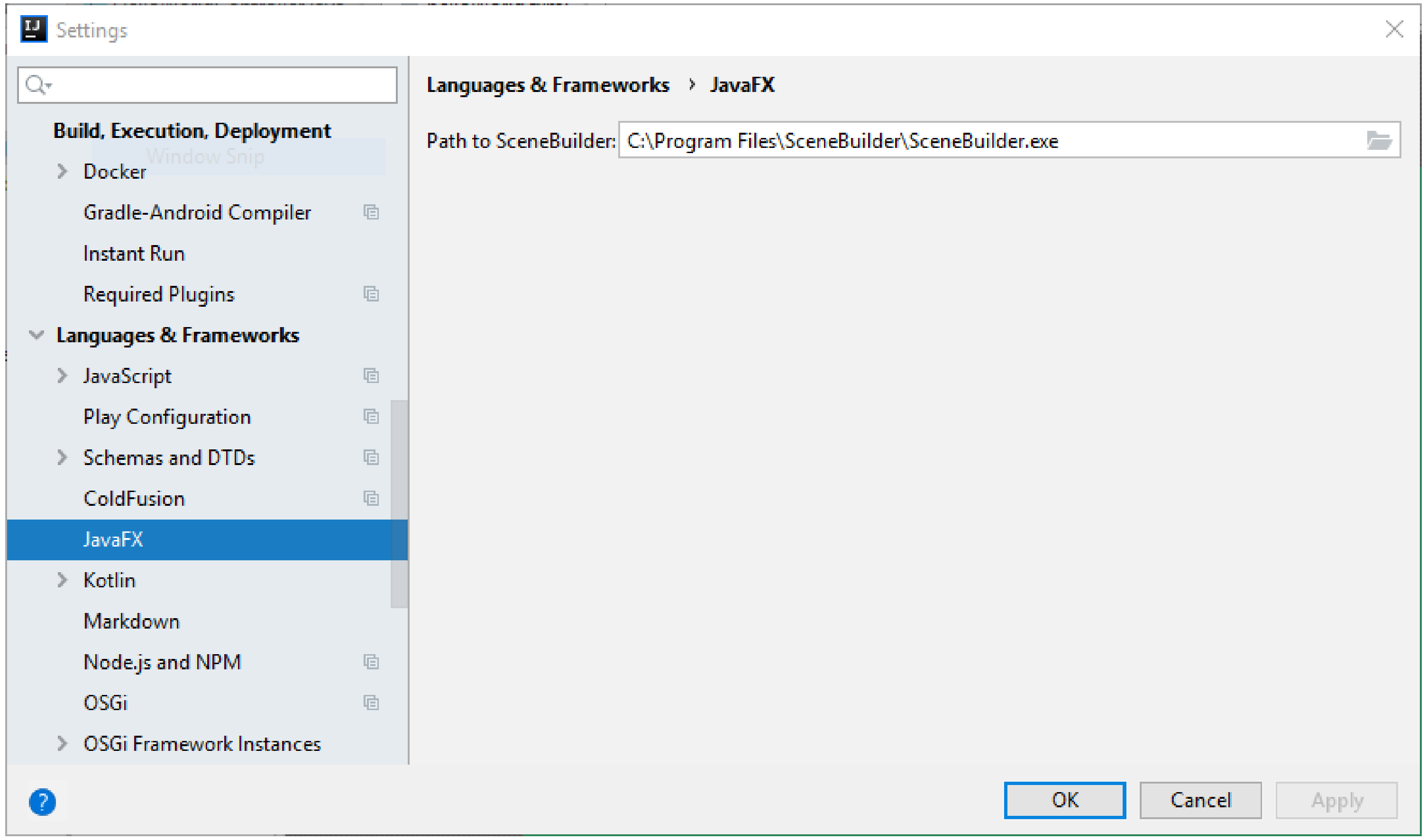
(If deleted, these files will be regenerated by IntelliJ upon reopening the project)



```
Project
demo1 C:\Users\user\IdeaProjects\demo1
  .idea
  src
    main
      java
        com.example.demo1
          HelloApplication
          HelloController
        module-info.java
      resources
        com.example.demo1
          hello-view.fxml
  demo1.iml
  pom.xml
External Libraries
Scratches and Consoles
```

Setting up Scenebuilder

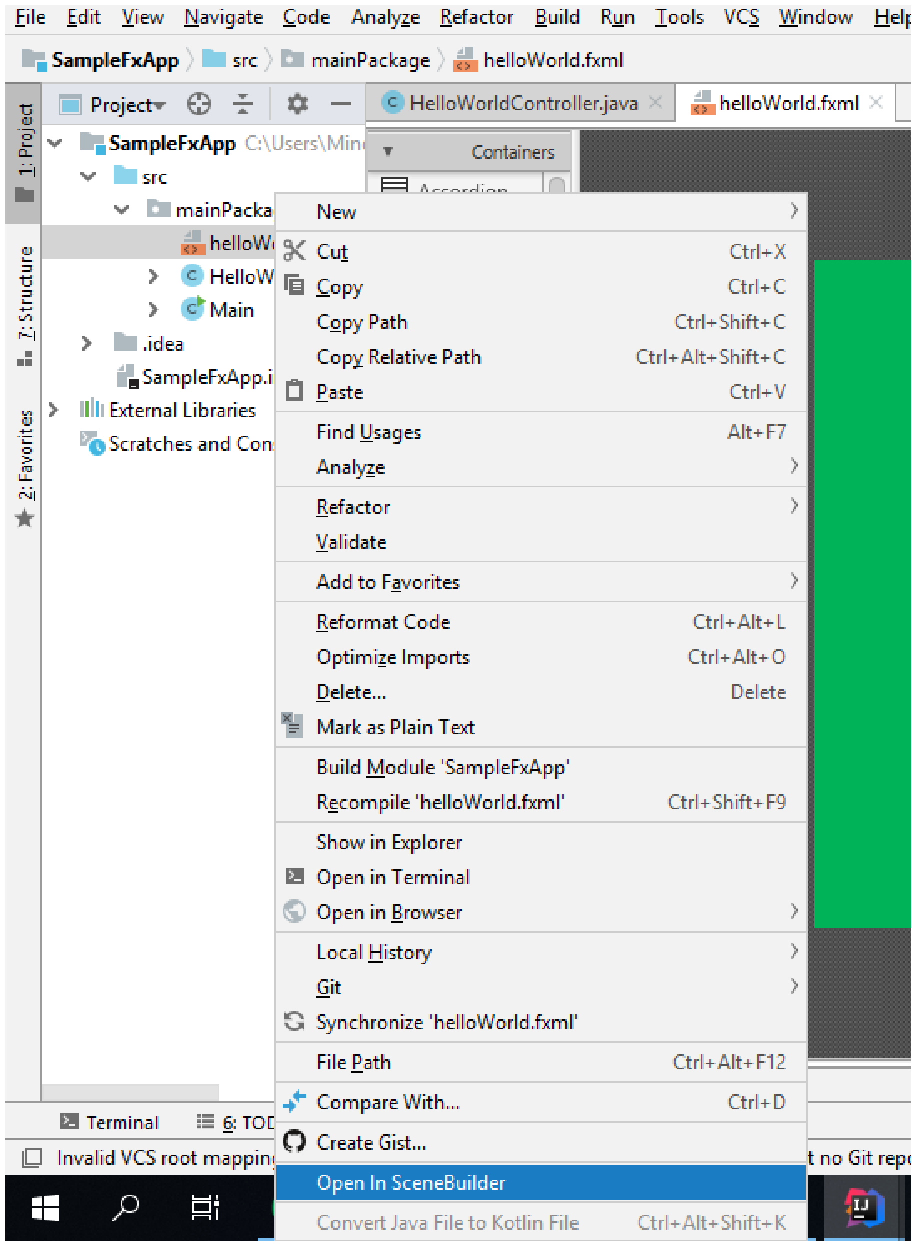
1. Go to File -> Settings
2. In the Settings window, select Languages and Frameworks -> JavaFX.
3. Give the path to SceneBuilder.exe from the location you installed.



Path to SceneBuilder: C:\Program Files\SceneBuilder\SceneBuilder.exe

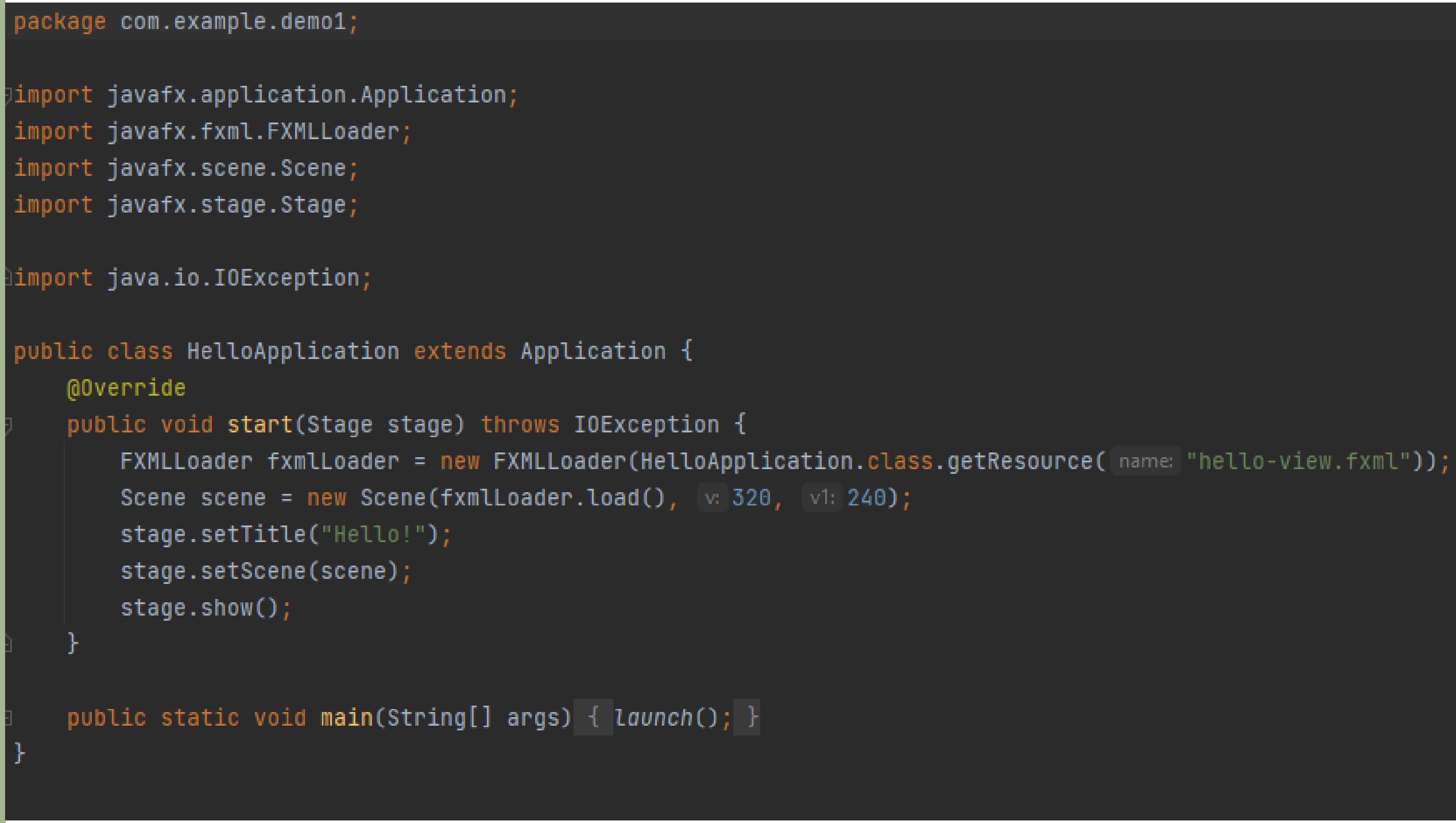
Opening a file in Scenebuilder

1. Right click on the FXML file you want to edit.
2. Select Open in SceneBuilder.
3. If you have already completed the previous step, the file will open in SceneBuilder.
4. If not, it will as you to give the path to SceneBuilder.

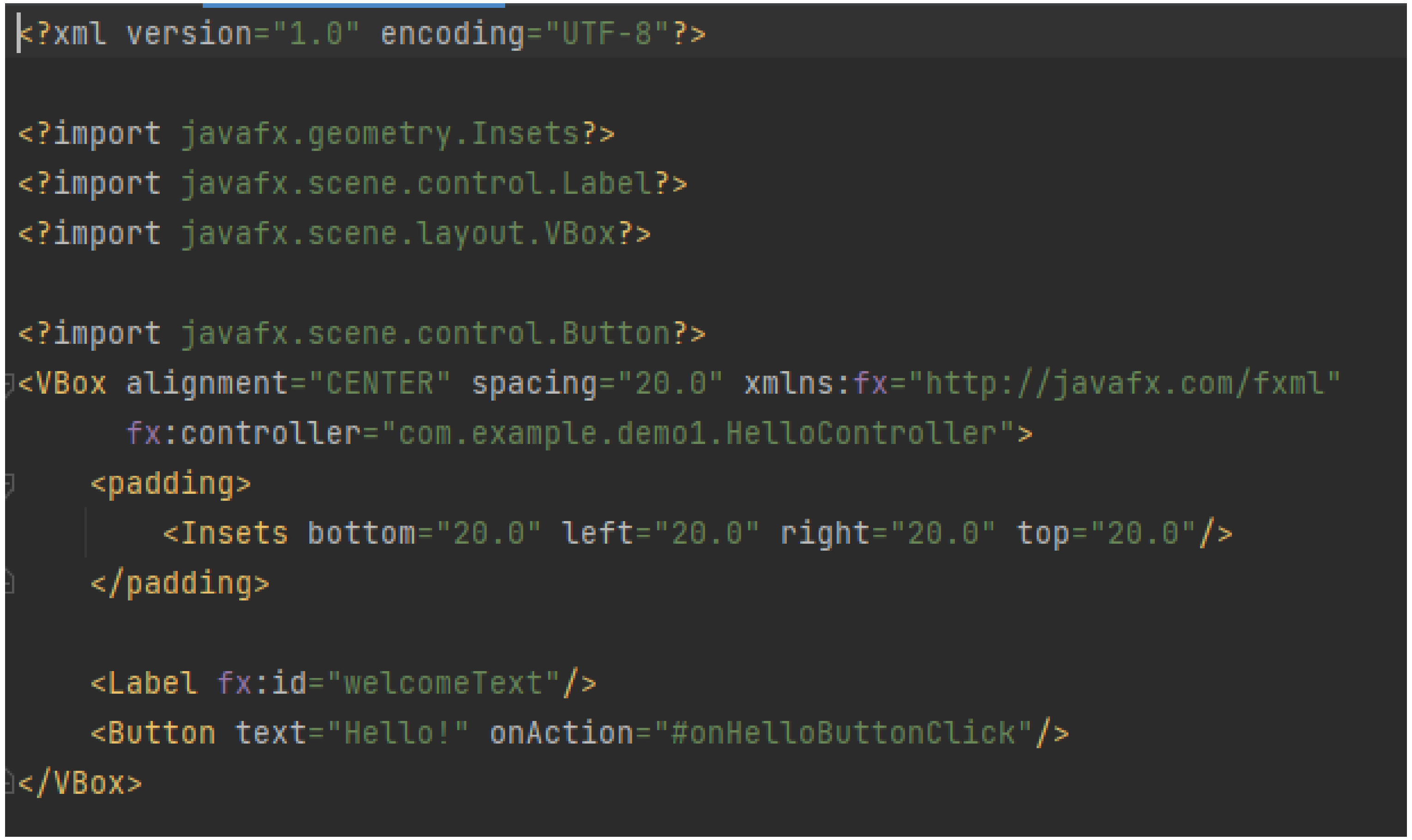


Main Class

- Contains the start method.
- Accesses the PrimaryStage.
- Creates the Scene with the FXML file as the root.
- Gives title to the Stage.
- Sets the Scene to the Stage.
- Shows the Stage on screen.



```java
package com.example.demo1;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource( name: "hello-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(),  v: 320,  v1: 240);
        stage.setTitle("Hello!");
        stage.setScene(scene);
        stage.show();
    }

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



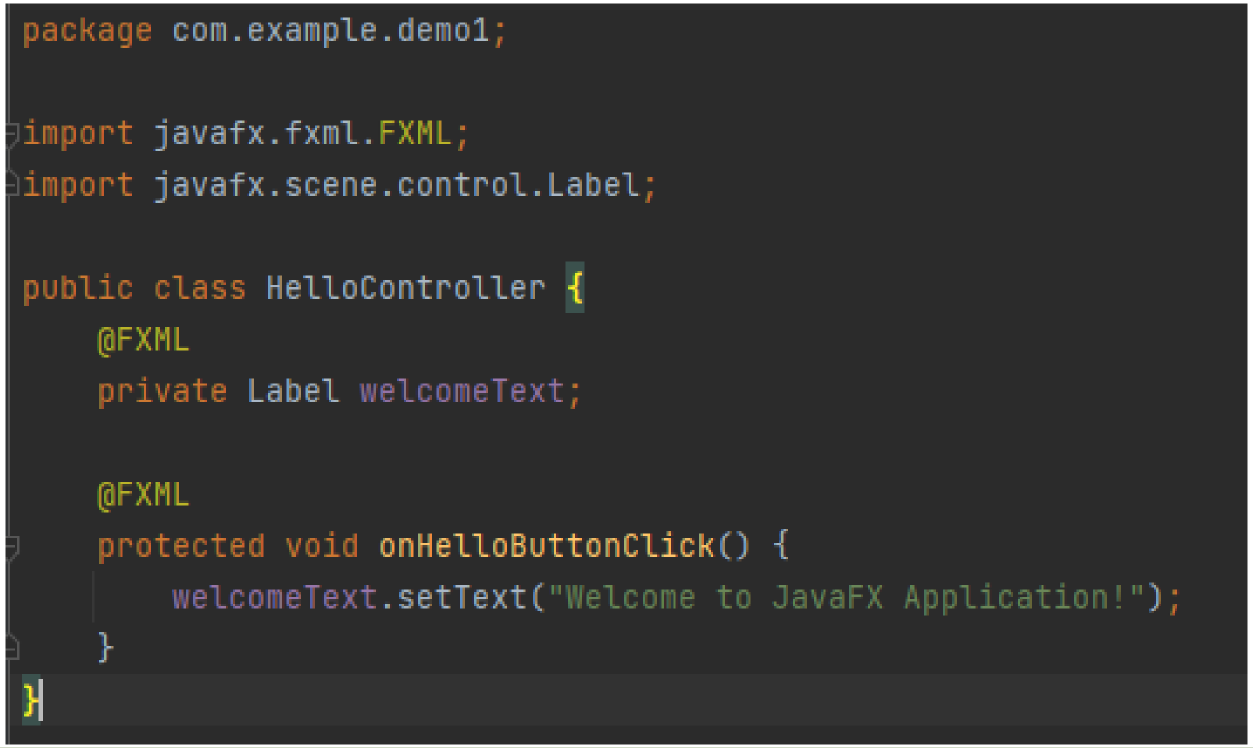
```
<?xml version="1.0" encoding="UTF-8"?>

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

<?import javafx.scene.control.Button?>
<VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"
      fx:controller="com.example.demo1.HelloController">
    <padding>
        <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
    </padding>

    <Label fx:id="welcomeText"/>
    <Button text="Hello!" onAction="#onHelloButtonClick"/>
</VBox>
```

FXML File

- Contains the Root node (A container).
- May contain Branch and Leaf nodes as children of the Root.
- Provides the View or the User Interface.
- Can be edited using the Text or SceneBuilder mode.

Controller class

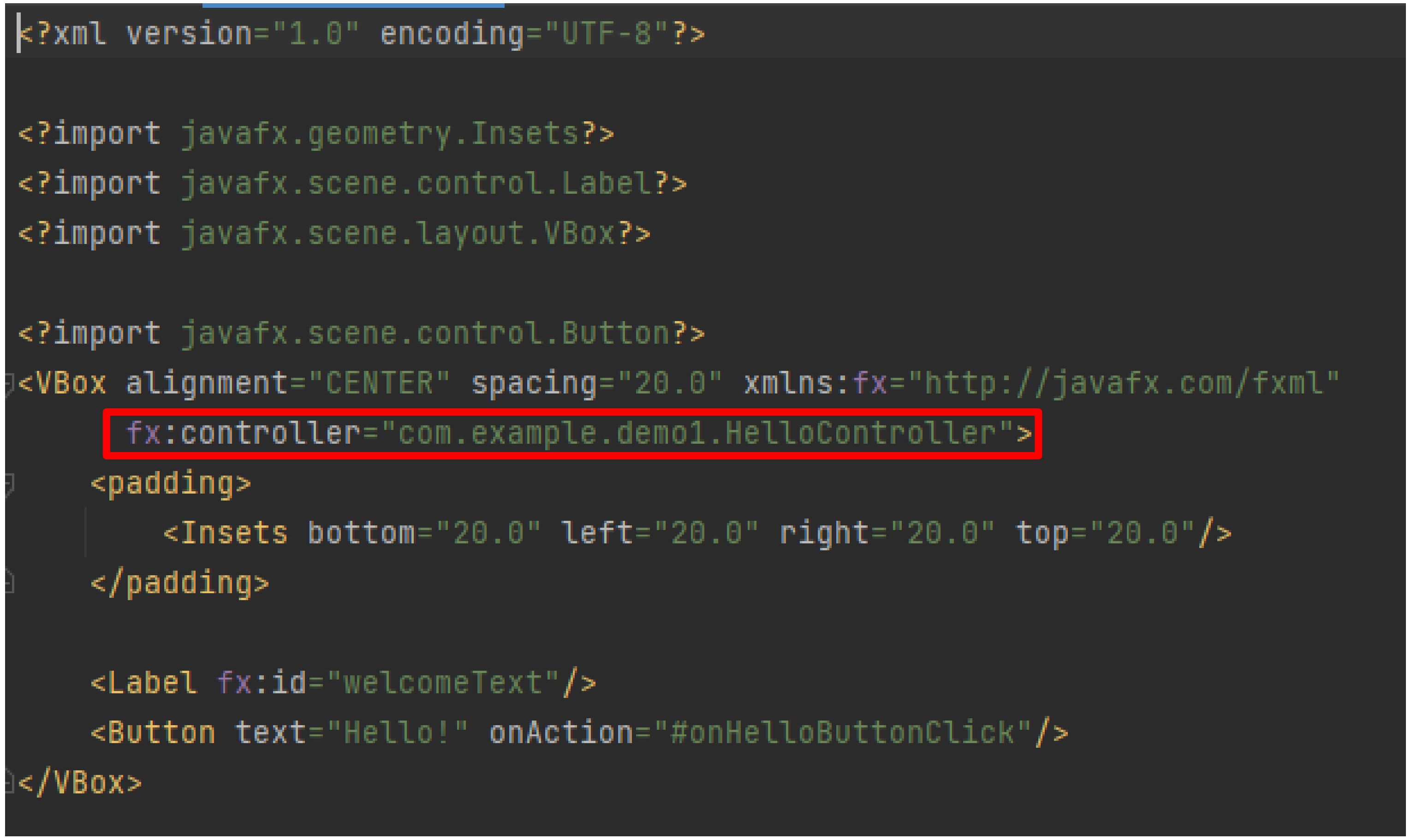
- Initially an empty Java Class.
- Used to Control the JavaFX elements in the View / FXML file.



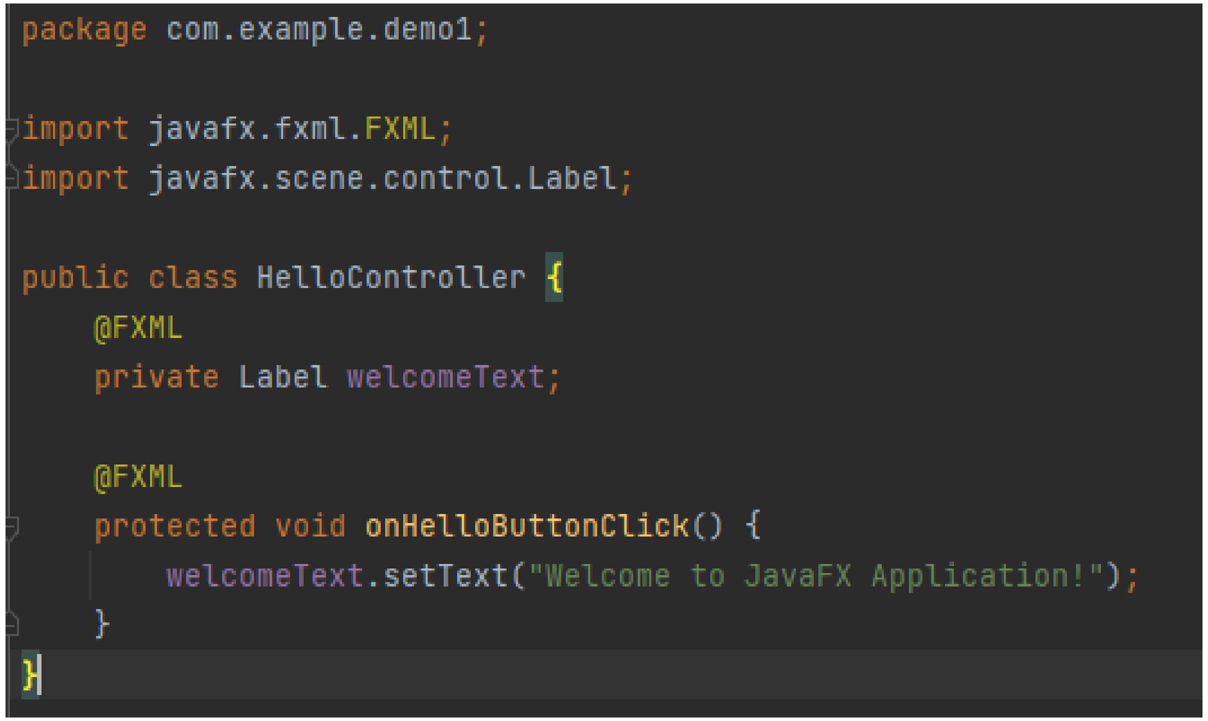
```java
package com.example.demo1;

import javafx.fxml.FXML;
import javafx.scene.control.Label;

public class HelloController {
    @FXML
    private Label welcomeText;

    @FXML
    protected void onHelloButtonClick() {
        welcomeText.setText("Welcome to JavaFX Application!");
    }
}
```

Assign the Controller to FXML File

- This makes the connection between the FXML file and it's specific Controller.
- The fx:controller attribute is added to the root node of the FXML file.
- It's recommended to have a separate Controller for each FXML file.



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

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

<?import javafx.scene.control.Button?>
<VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"
      fx:controller="com.example.demo1.HelloController">
    <padding>
        <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
    </padding>

    <Label fx:id="welcomeText"/>
    <Button text="Hello!" onAction="#onHelloButtonClick"/>
</VBox>
```



```java
package com.example.demo1;

import javafx.fxml.FXML;
import javafx.scene.control.Label;

public class HelloController {
    @FXML
    private Label welcomeText;

    @FXML
    protected void onHelloButtonClick() {
        welcomeText.setText("Welcome to JavaFX Application!");
    }
}
```

Working with Controller.java

- Create references for any JavaFX element which will be accessed from code.

Ex: private Label welcomeText;

- Create methods which will be called on button clicks or other actions.

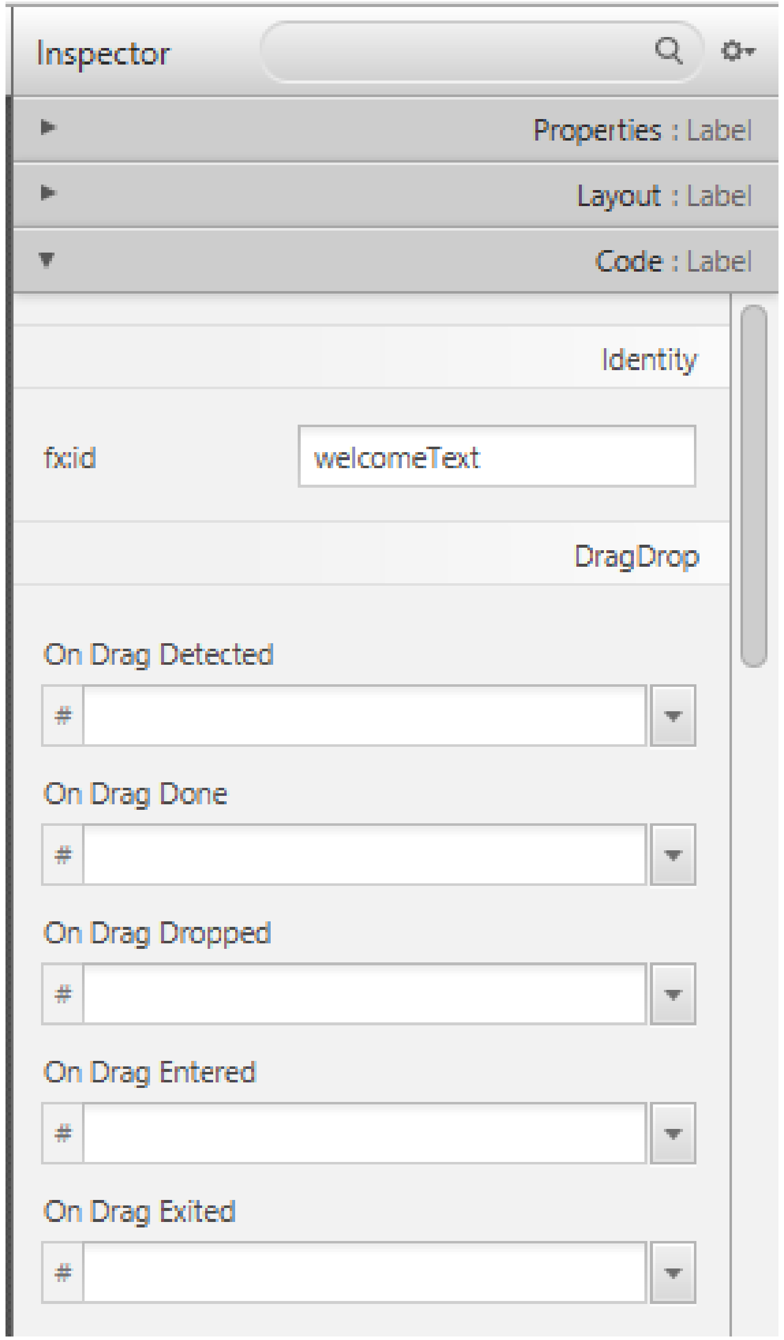
Ex: public void onHelloButtonClick()

- Pay attention to the imports and the @FXML annotation.

(Only import from javafx library)

Assigning the ids - SceneBuilder

- From SceneBuilder, select the component to add the reference to.
- If the fx:controller is set in the FXML file, and the content of the Controller class is saved, the references created for FX elements will be given in a dropdown next to id field in the ***Code view***.
- Assign/ type the reference as the id of the appropriate element.



fx:id welcomeText

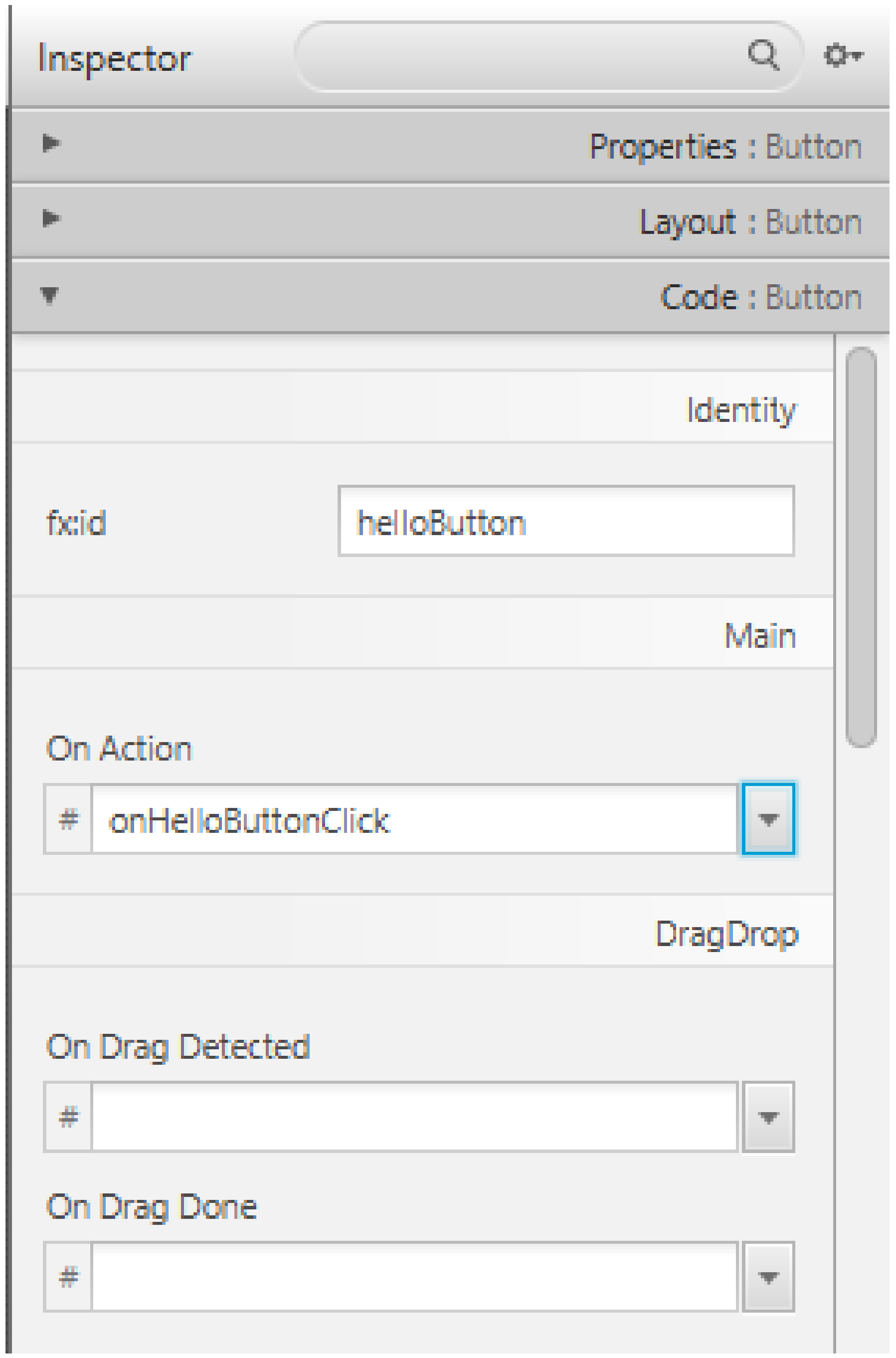
Assigning the ids - FXML

- The assigned id will be reflected in the FXML as follows.
- Alternatively, you can add this manually to the FXML file.

```
<Label fx:id="welcomeText"/>
```

Assigning the event handling methods - SceneBuilder

- Similarly, add the reference to the method to be called on button click.
- This should be added to the On Action field in the Code View after selecting the button.



fx:id helloButton

On Action # onHelloButtonClick

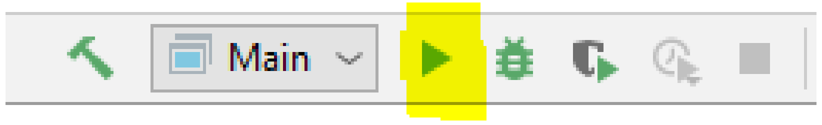
Assigning the event handling methods - FXML

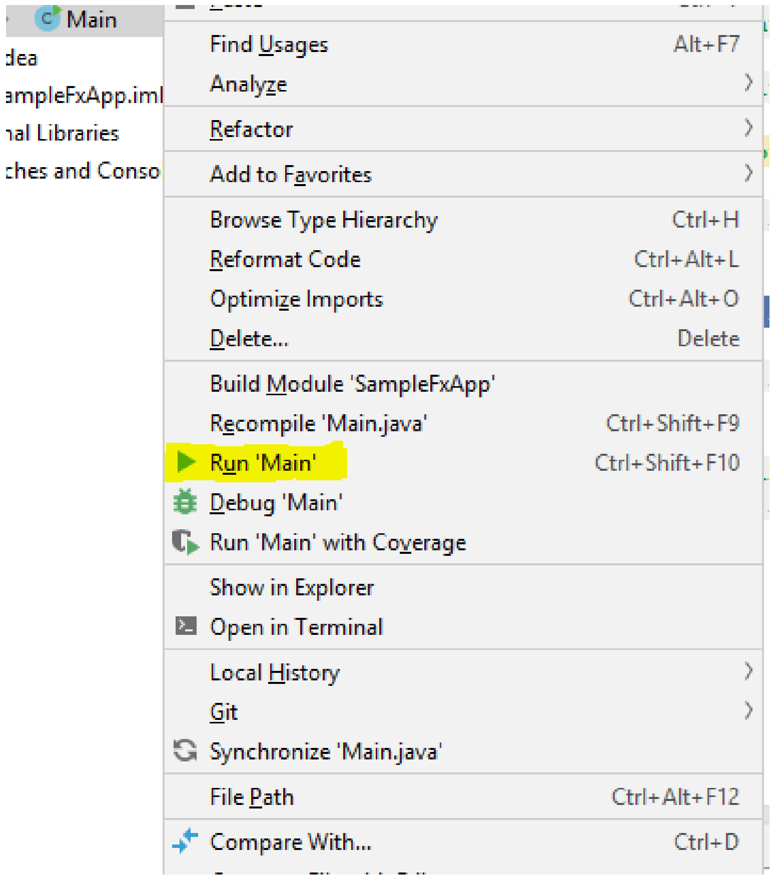
- The assigned method will be reflected in the FXML as follows.
- Alternatively, you can add this manually to the FXML file.

```
<Button fx:id="helloButton" text="Hello!" onAction="#onHelloButtonClick"/>
```

Run the program!

- Once you’re done with all the steps, click on the Run icon or right click the Main class and select Run.
- If you get any errors, please go through the steps again to make sure you have not missed any important steps.




Main
dea
ampleFxApp.iml
hal Libraries
ches and Conso
Find Usages Alt+F7
Analyze
Refactor
Add to Favorites
Browse Type Hierarchy Ctrl+H
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Delete... Delete
Build Module 'SampleFxApp'
Recompile 'Main.java' Ctrl+Shift+F9
Run 'Main' Ctrl+Shift+F10
Debug 'Main'
Run 'Main' with Coverage
Show in Explorer
Open in Terminal
Local History
Git
Synchronize 'Main.java'
File Path Ctrl+Alt+F12
Compare With... Ctrl+D