



JavaFX I

Shin-Jie Lee (李信杰)

Associate Professor

Computer and Network Center

Department of Computer Science and Information Engineering

National Cheng Kung University



GUI

Graphical User Interface



```
User@RPI: ~$ ssh -p 22097 P768950348140.116.246.189
Enter passphrase for '/etc/Users/User/.ssh/id_rsa':
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-70-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

System information as of Fri Apr 9 18:24:55 CST 2021

System load: 0.0      Processes: 168
Usage of /: 41.8% of 7.76GB   Users logged in: 0
Memory usage: 24%
Swap usage: 0%
* Introducing self-healing high availability clusters in MicroK8s.
  Simple, hardened, Kubernetes for production, from RaspberryPi to DC.
  https://microk8s.io/high-availability
14 updates can be installed immediately.
0 of them are security updates.
```



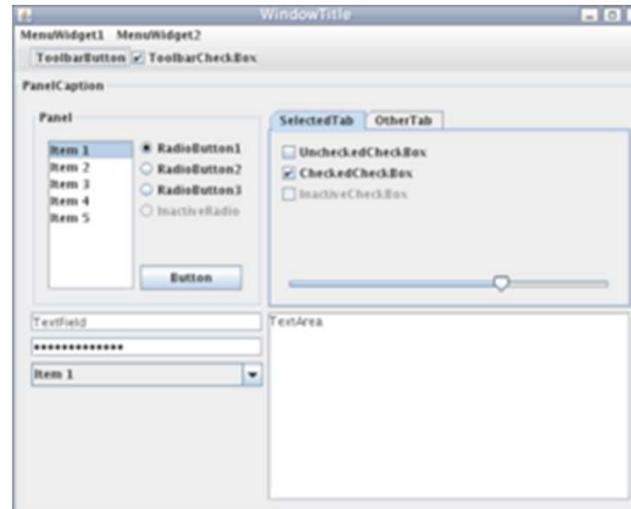


Java GUI Tools

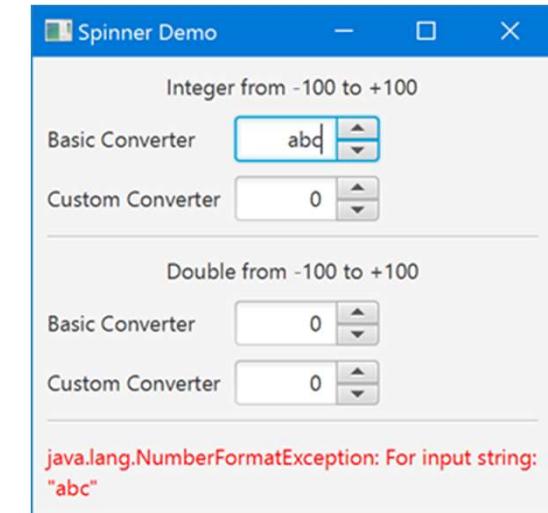
AWT



Swing



FX



Removed from JDK 11



JavaFX Feature 1

Cross-Platform



Windows, macOS, Linux

WebView

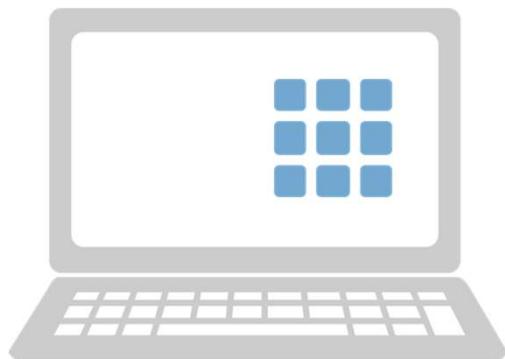
iOS, Android, Windows Mobile

“write once, run
anywhere”



JavaFX Feature 2

Native Packaging



Distribute as a native app

No dependencies on Java version

Minimized app size



Why JavaFX was removed from JDK11?



Modular System of JDK

more flexibility for contributors to engage in the open source community



HTML



Cannot Compete with HTML5

HTML5 is a larger web standard which cannot be easily replaced



JavaFX removed from JDK

=

Manual Installation



1. Installation



Download JavaFX SDK

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

& Long Term Support (LTS) ☰

& Latest Release

Product	Public version	LTS version	Platform	Download
JavaFX Windows SDK	11.0.2	11.0.11 More info	Windows	Download [SHA256]
JavaFX Windows jmods	11.0.2	11.0.11 More info	Windows	Download [SHA256]
JavaFX Mac OS X SDK	11.0.2	11.0.11 More info	Mac	Download [SHA256]
JavaFX Mac OS X jmods	11.0.2	11.0.11 More info	Mac	Download [SHA256]
JavaFX Linux SDK	11.0.2	11.0.11 More info	Linux	Download [SHA256]
JavaFX Linux jmods	11.0.2	11.0.11 More info	Linux	Download [SHA256]
JavaFX armv6hf SDK	11.0.2	11.0.11 More info	Embedded armv6hf	Download [SHA256]
JavaFX Documentation	11.0.2	11.0.11 More info	Javadoc	Download [SHA256]



Download JavaFX SDK

Extract the downloaded file, you will find
three folders :

- & bin/
- & legal/
- & lib/



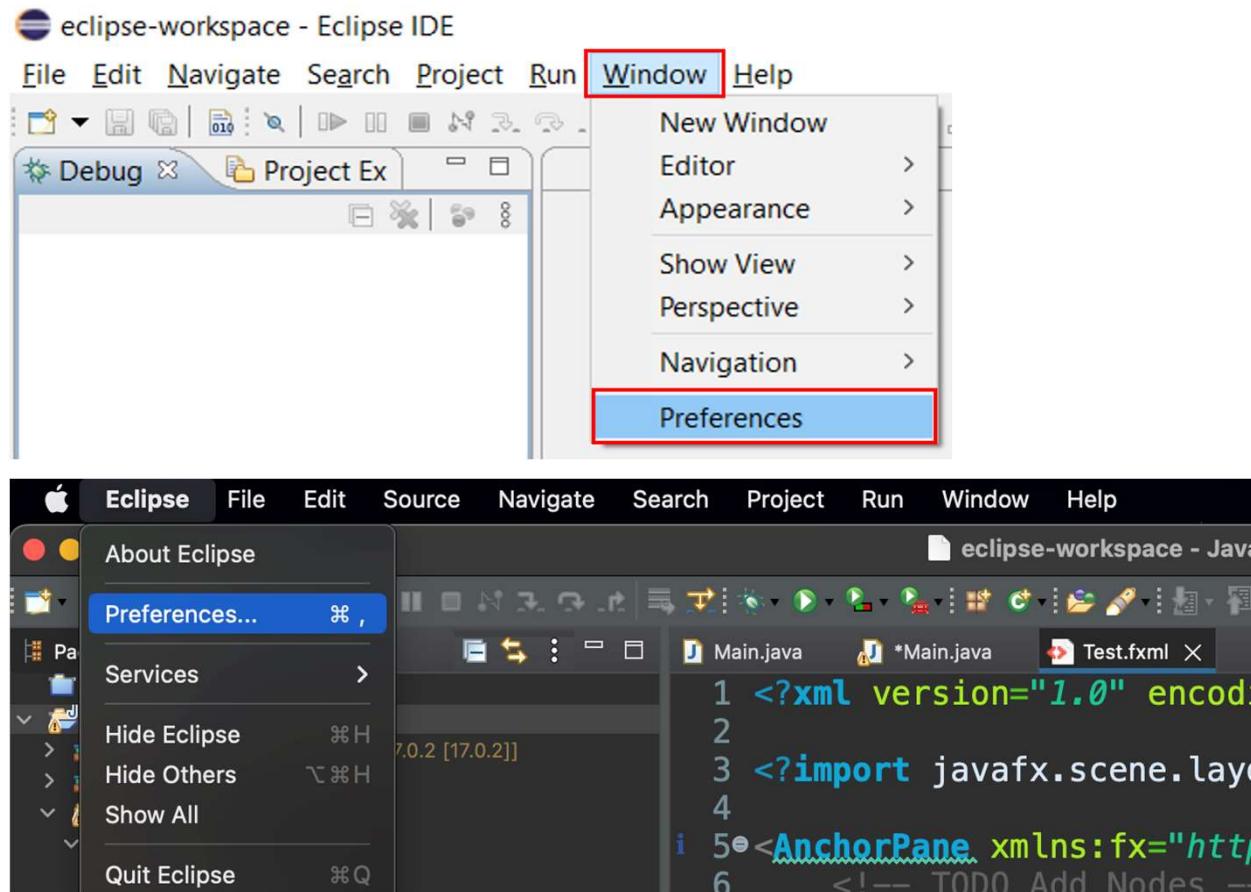
Install JavaFX

& <https://openjfx.io/openjfx-docs/#install-javafx>



Install JavaFX in Eclipse IDE

& Window → Preferences

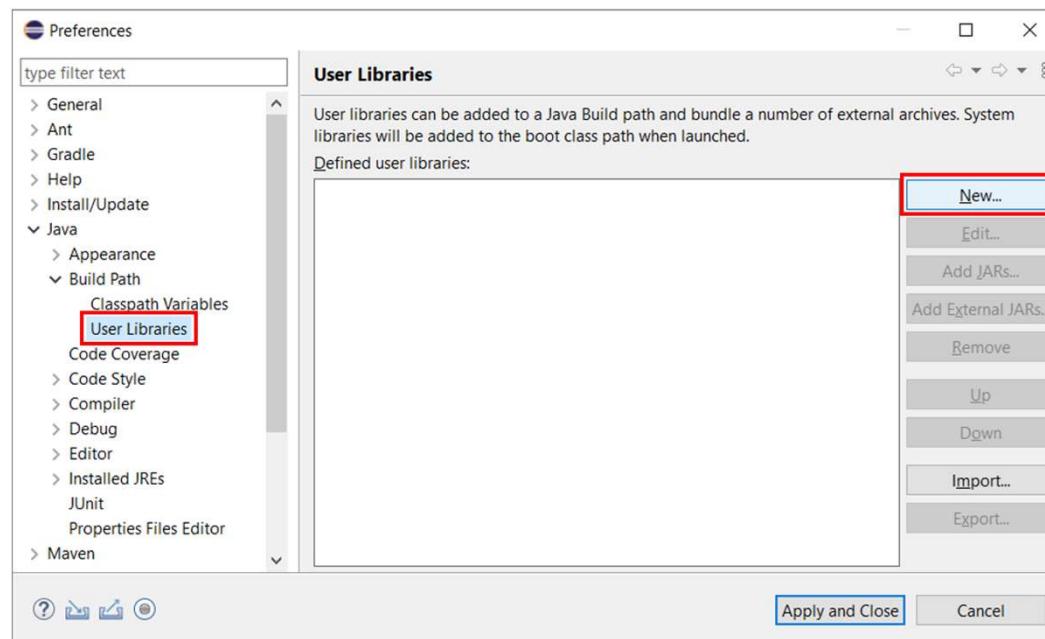




Install JavaFX in Eclipse IDE

& Create New User Library

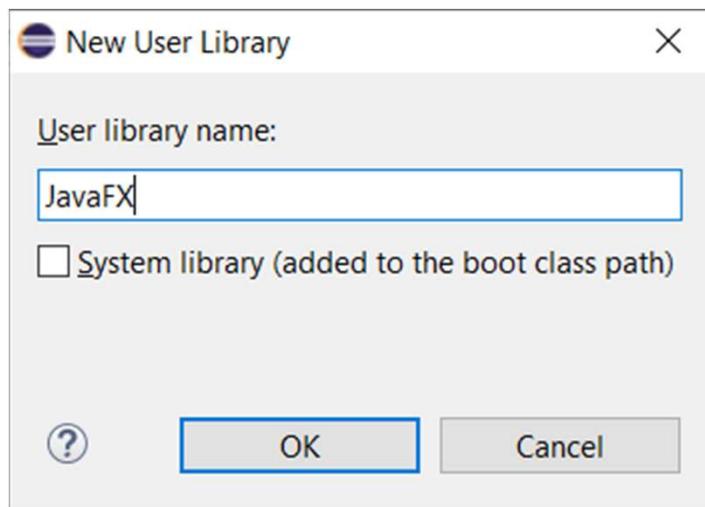
⌘ Java → Build Path → User Libraries → New...





Install JavaFX in Eclipse IDE

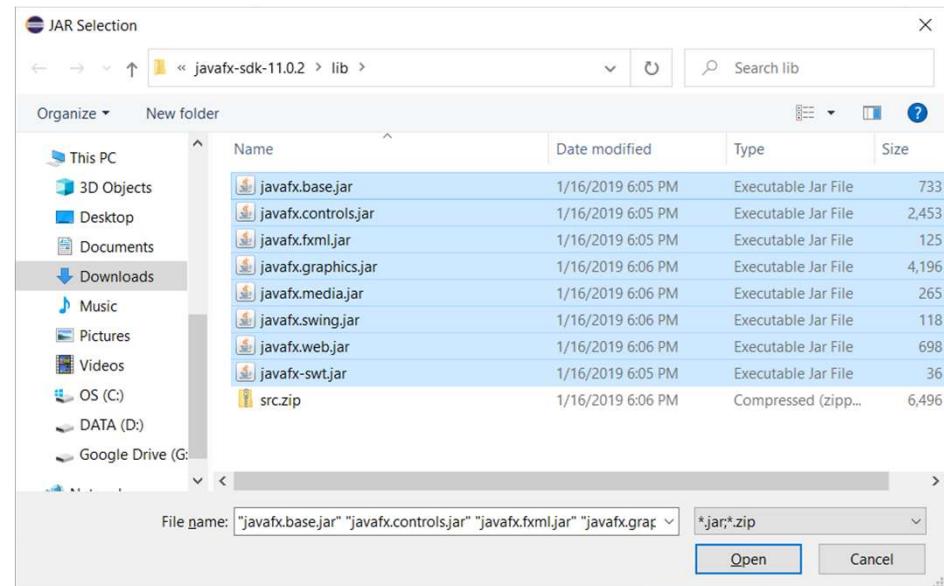
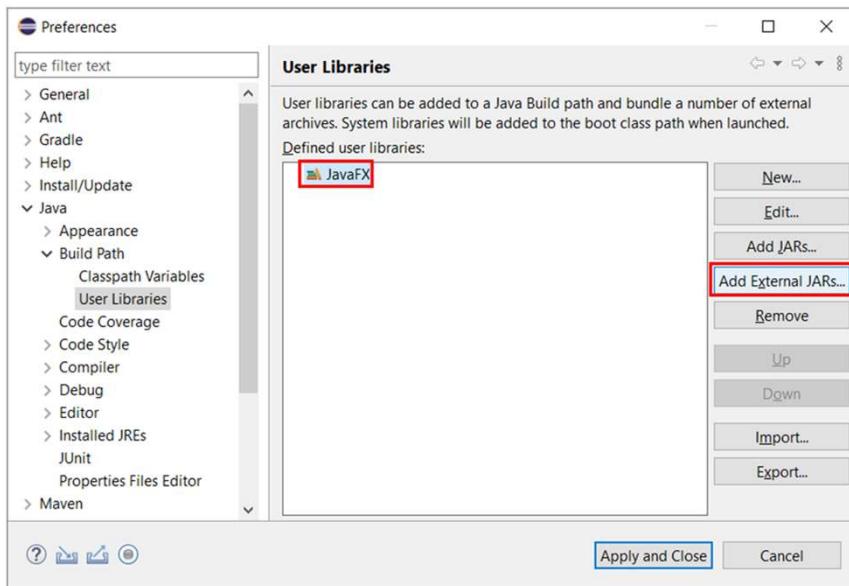
& Enter User Library Name (“JavaFX” , "javafx-11.0.2" , ...)





Install JavaFX in Eclipse IDE

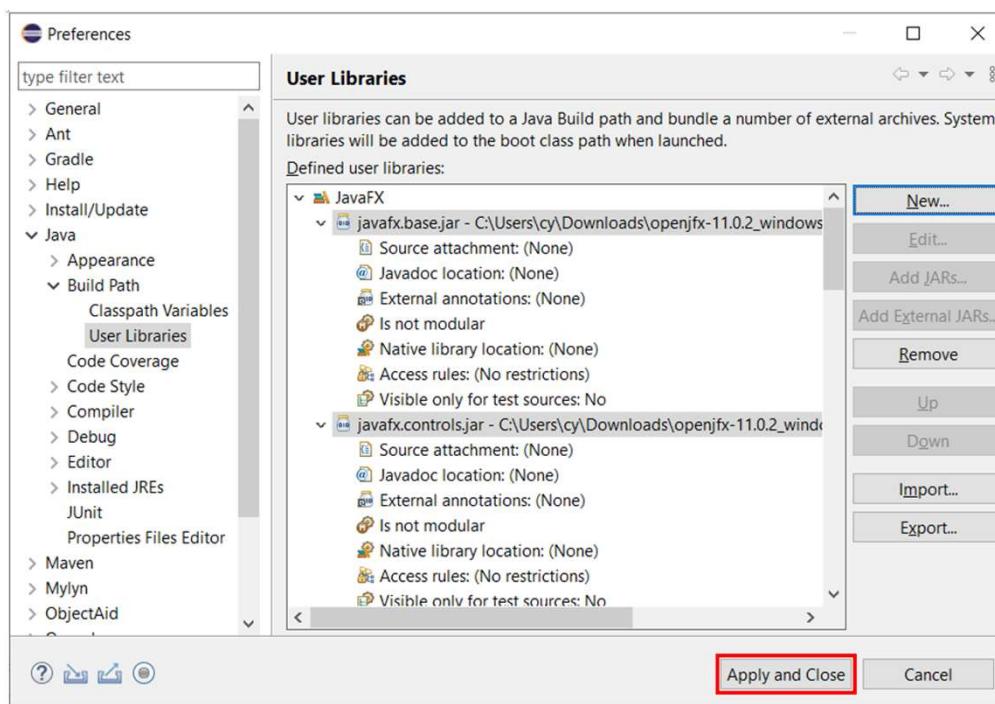
1. Add External JARs...
2. Select All *.jar in lib/





Install JavaFX in Eclipse IDE

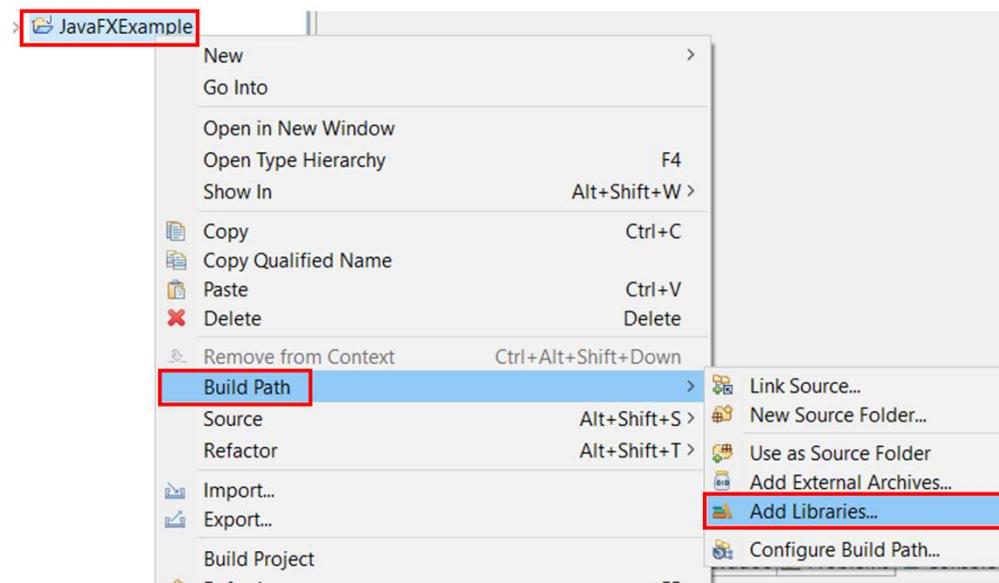
& Apply and Close





Install JavaFX in Eclipse IDE

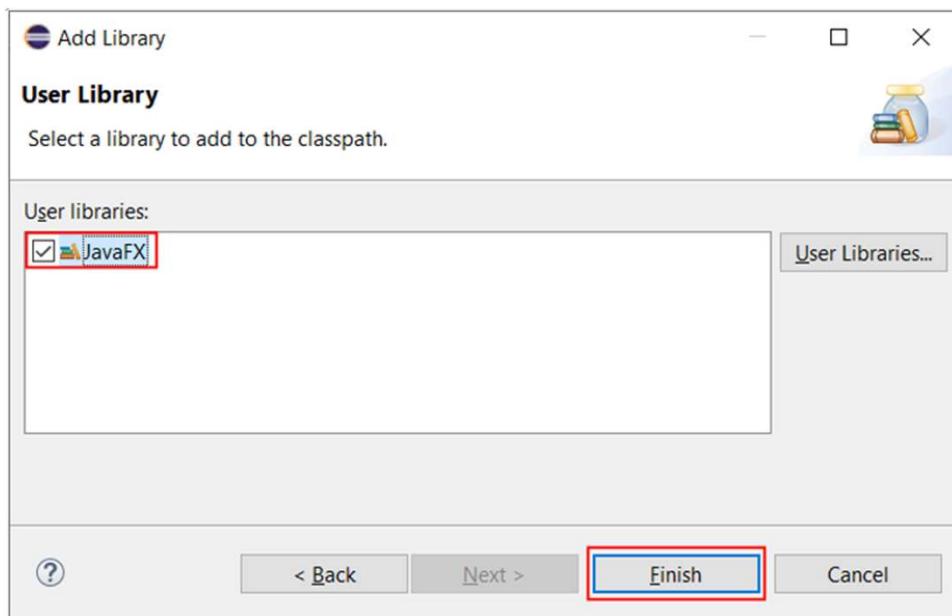
1. Create a New Java Project
2. Right Click the Project
3. Bulid Path → Add Libraries...





Install JavaFX in Eclipse IDE

1. Select “User Library” → Next
2. Select JavaFX User Library → Finish





Try it !

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloJavaFX extends Application {

    @Override
    public void start(Stage stage) {
        Label label = new Label("Hello, JavaFX");
        Scene scene = new Scene(new StackPane(label), 300, 200);
        stage.setScene(scene);
        stage.show();
    }

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



Oops...

Warning: If you now run the project it will compile but you will get this error:

Error: JavaFX runtime components are missing, and are required to run this application

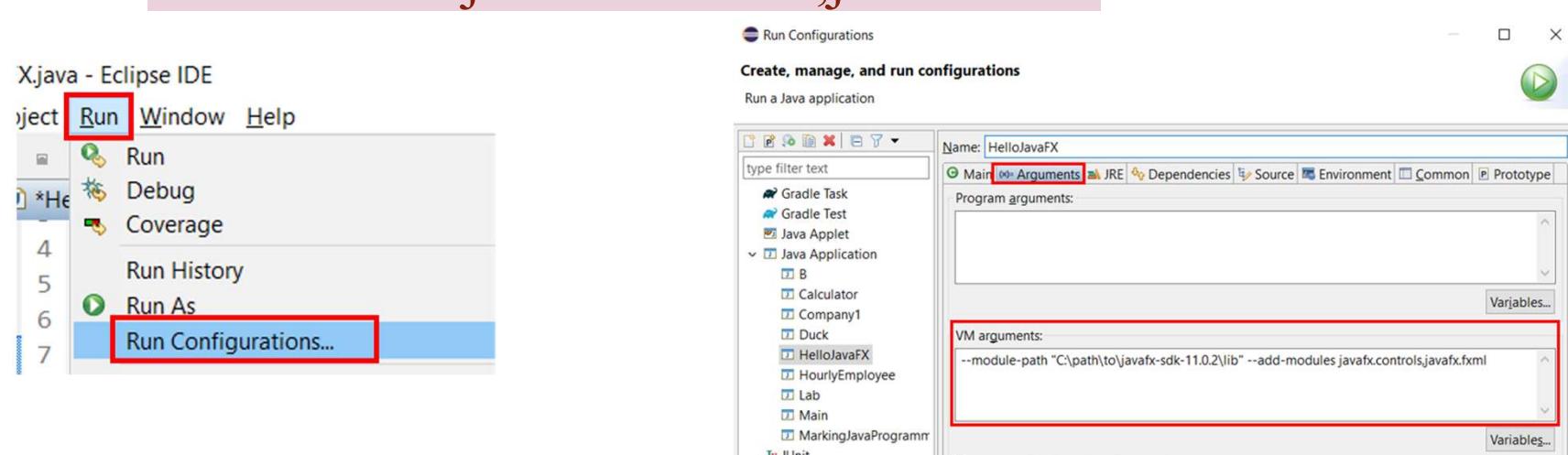
This error is shown since the Java 15 launcher checks if the main class extends `javafx.application.Application`. If that is the case, it is required to have the `javafx.graphics` module on the module-path.

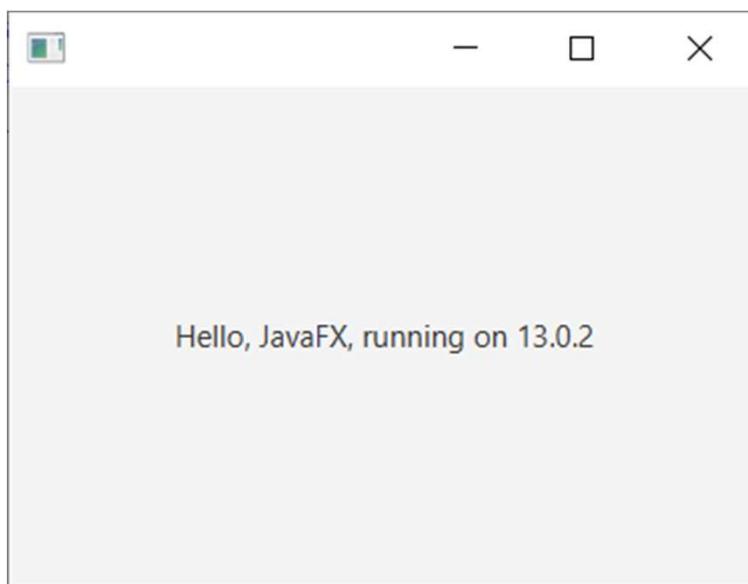


Add VM arguments

1. Run → Run Configurations...
2. Arguments → VM arguments

- ⌘ Linux/Mac: `--module-path /path/to/javafx-sdk-15.0.1/lib --add-modules javafx.controls,javafx.fxml`
- Windows: `--module-path "C:\path\to\javafx-sdk-11.0.2\lib" --add-modules javafx.controls,javafx.fxml`



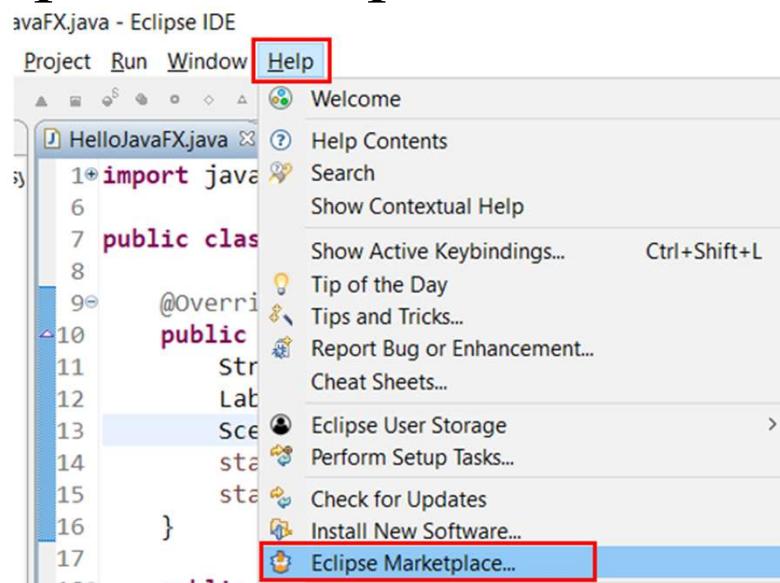




Install e(fx)clipse in Eclipse IDE

[e\(fx\)clipse](#) is an extension that provides JavaFX tooling for the Eclipse IDE.

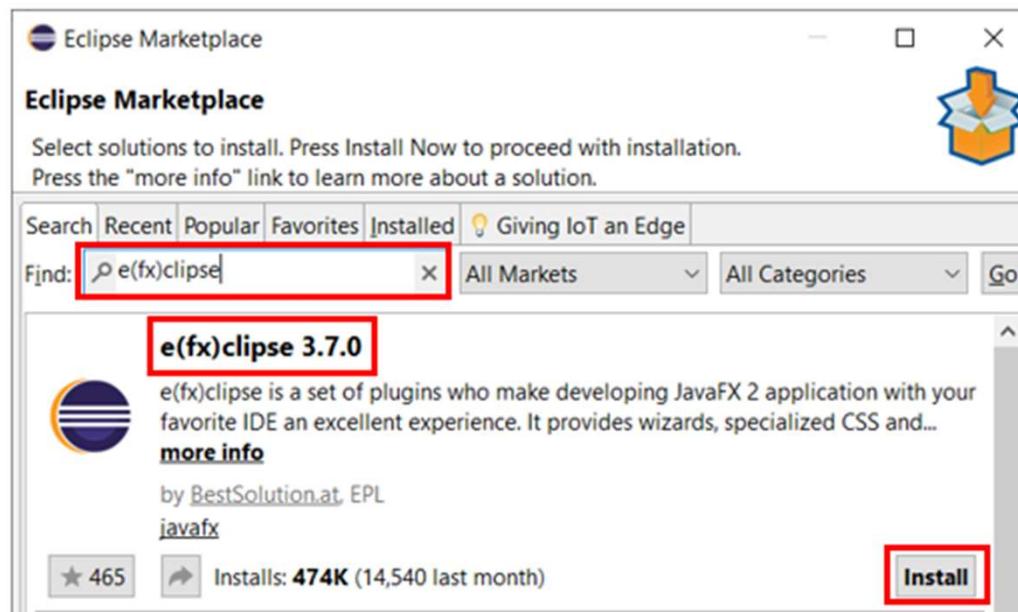
& Help → Eclipse Marketplace...





Install e(fx)clipse in Eclipse IDE

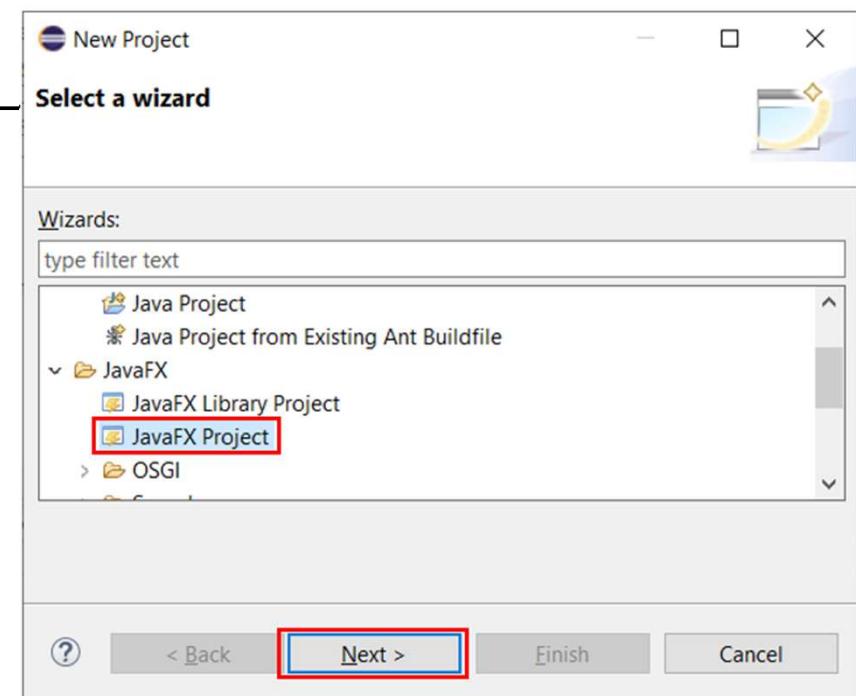
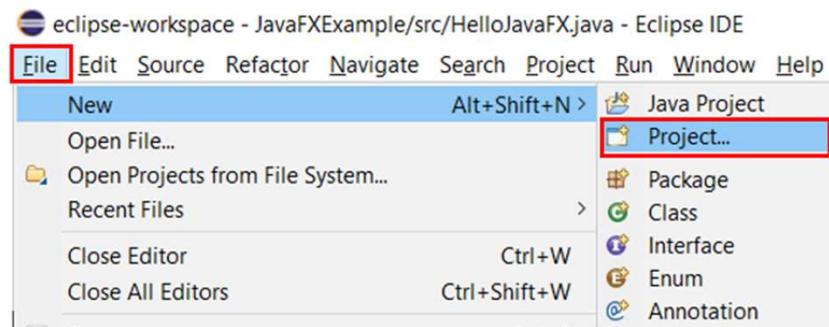
& Find "e(fx)clipse" → Install





Install e(fx)clipse in Eclipse IDE

- & File → Project
- & JavaFX Project → Next —





Oops...

- & ┌ Error occurred during initialization of boot layer
java.lang.module.FindException: Module
HelloJavaFX not found ┘

eclipse-workspace - HelloJavaFX/src/application/Main.java - Eclipse IDE

File Edit Source Refactor Navigate Project Run Window Help

Package Explorer HelloJavaFX.java Lab.java Main.java

```
1 package application;
2
3@import javafx.application.Application;□
4
5 public class Main extends Application {
6     @Override
7     public void start(Stage primaryStage) {
8         try {
9             BorderPane root = new BorderPane();
10            Scene scene = new Scene(root,400,400);
11            scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
12            primaryStage.setScene(scene);
13            primaryStage.show();
14        } catch(Exception e) {
15            e.printStackTrace();
16        }
17    }
18 }
```

Search Declaration Coverage Javadoc Problems Console

<terminated> Main (1) [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (May 3, 2021, 9:21:53 PM – 9:21:56 PM)

Error occurred during initialization of boot layer
java.lang.module.FindException: Module HelloJavaFX not found



Solution 1: Quick Fixes in Eclipse

& Open module-info.java

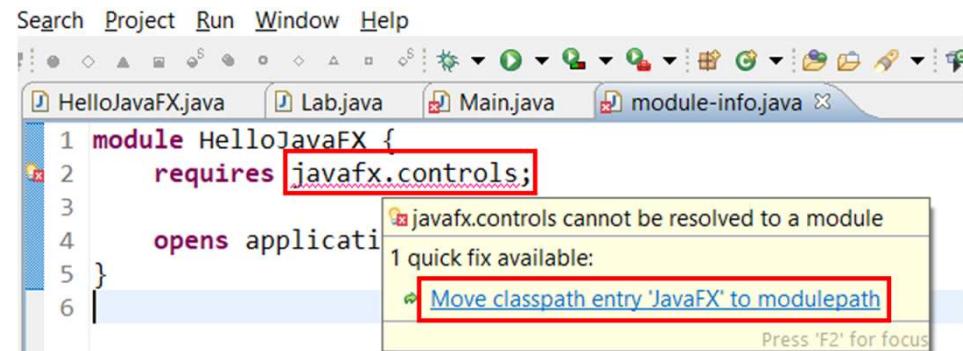
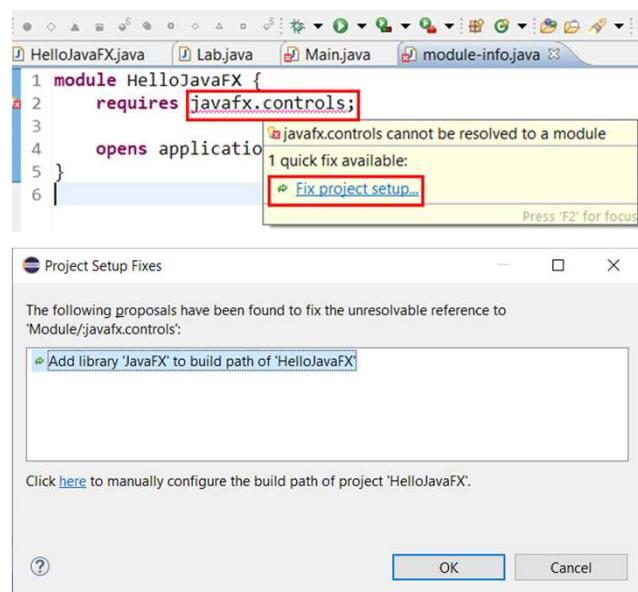
The screenshot shows the Eclipse IDE interface. The title bar reads "eclipse-workspace - HelloJavaFX/src/module-info.java - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations. The left side features the "Package Explorer" view, which displays the project structure: "HelloJavaFX" (expanded) containing "src" (expanded), "application" (expanded), "Main.java", "application.css", and "module-info.java" (highlighted with a red border). The right side shows the code editor with the following content:

```
1 module HelloJavaFX {  
2     requires javafx.controls;  
3  
4     opens application to javafx.fxml;  
5 }  
6
```



Solution 1: Quick Fixes in Eclipse

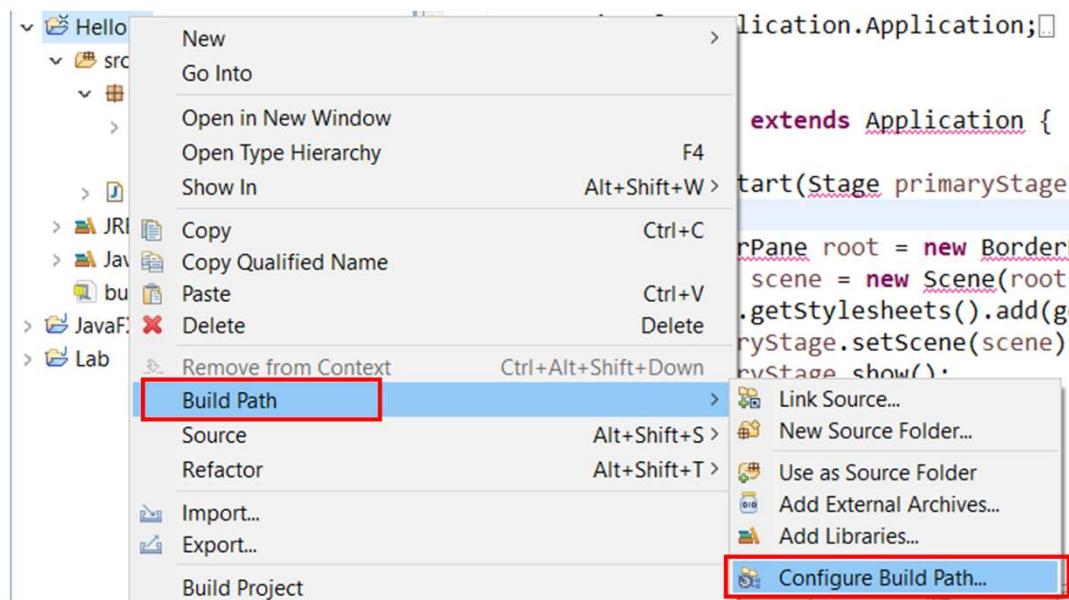
- a. Fix project setup
- b. Move classpath entry 'JavaFX' to modulepath :





Solution 2: Manually Fix Ref Problems

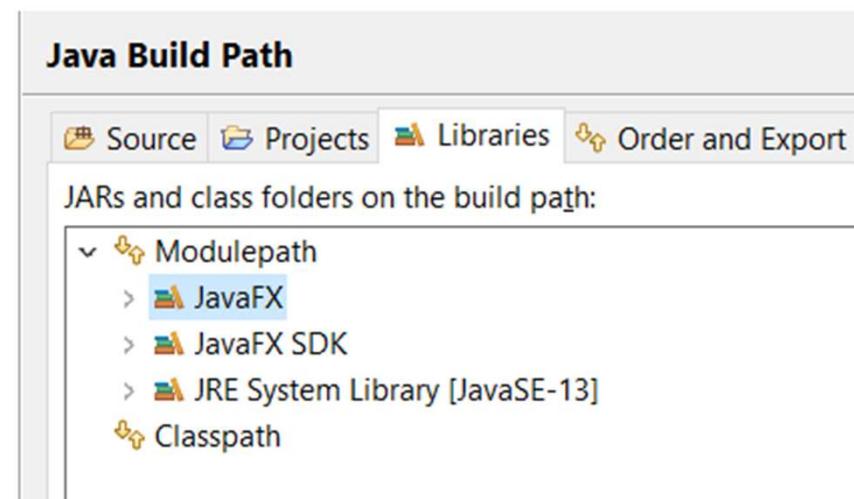
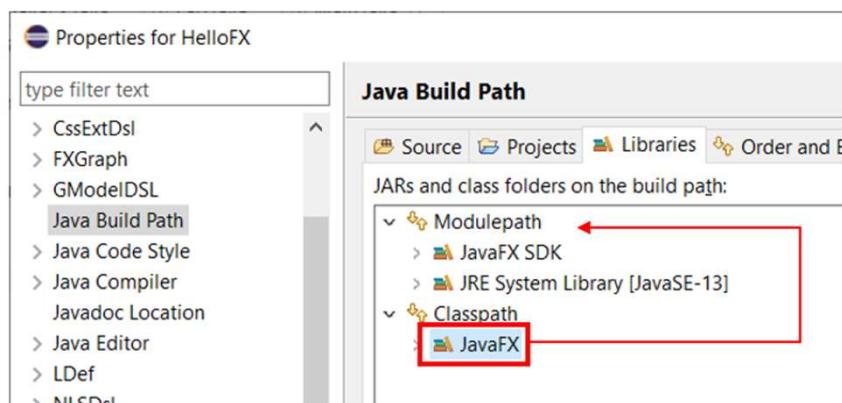
1. Add JavaFX User Library
2. Right Click Project → Build Path → Configure Build Path :





Solution 2: Manually Fix Ref Problems

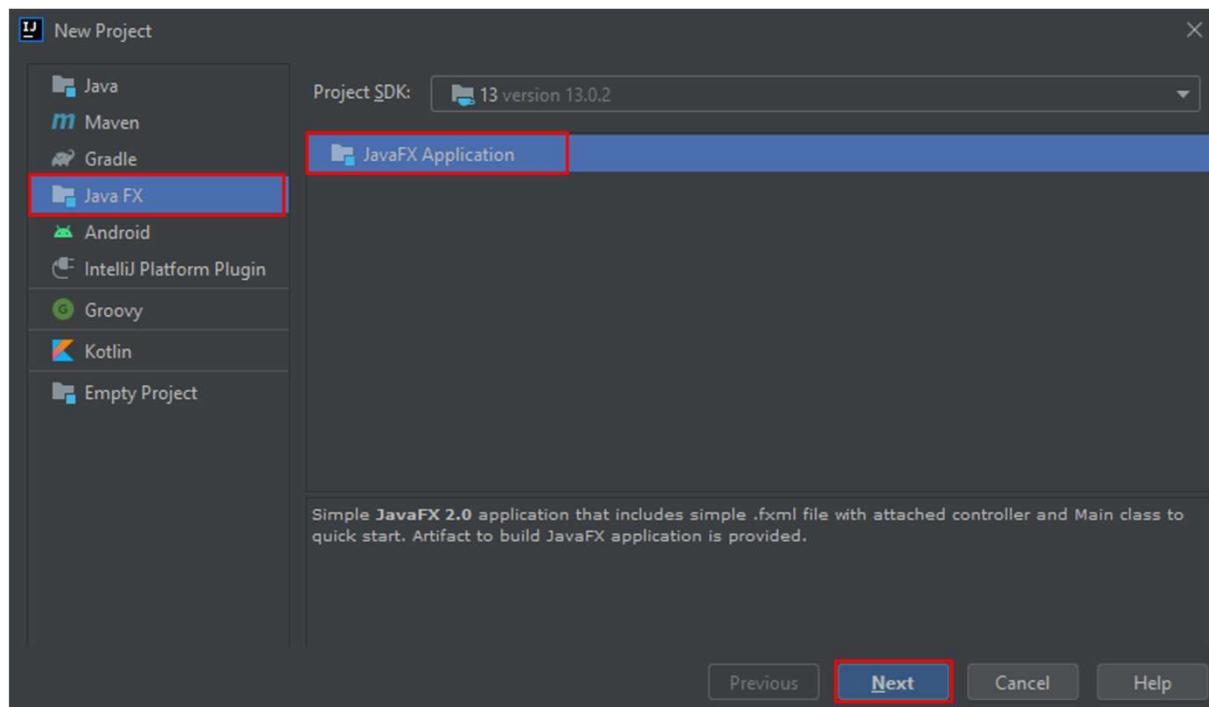
3. Move JavaFX User From classpath to module path :





Install JavaFX in IntelliJ IDEA

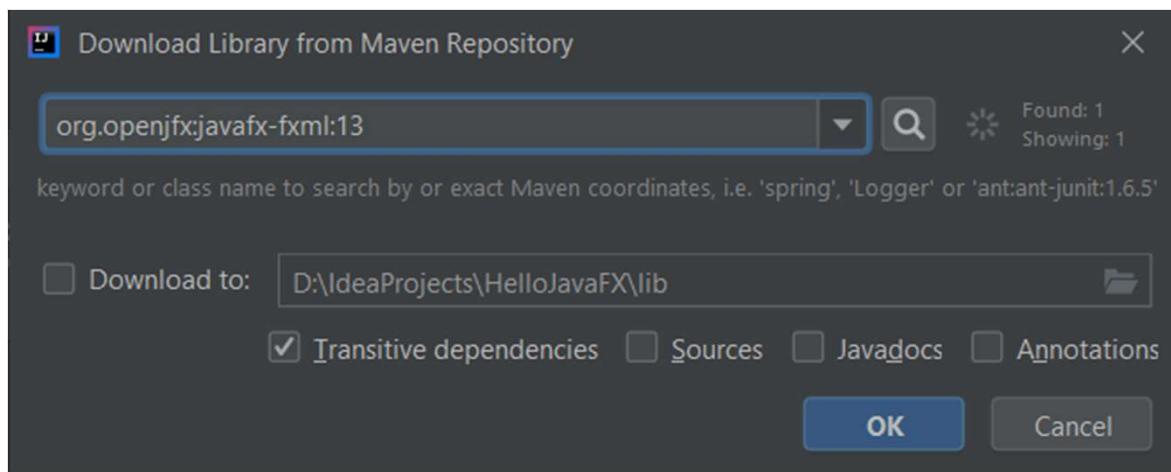
1. New Project → Java FX → JavaFX Application
→ Next :





Install JavaFX in IntelliJ IDEA

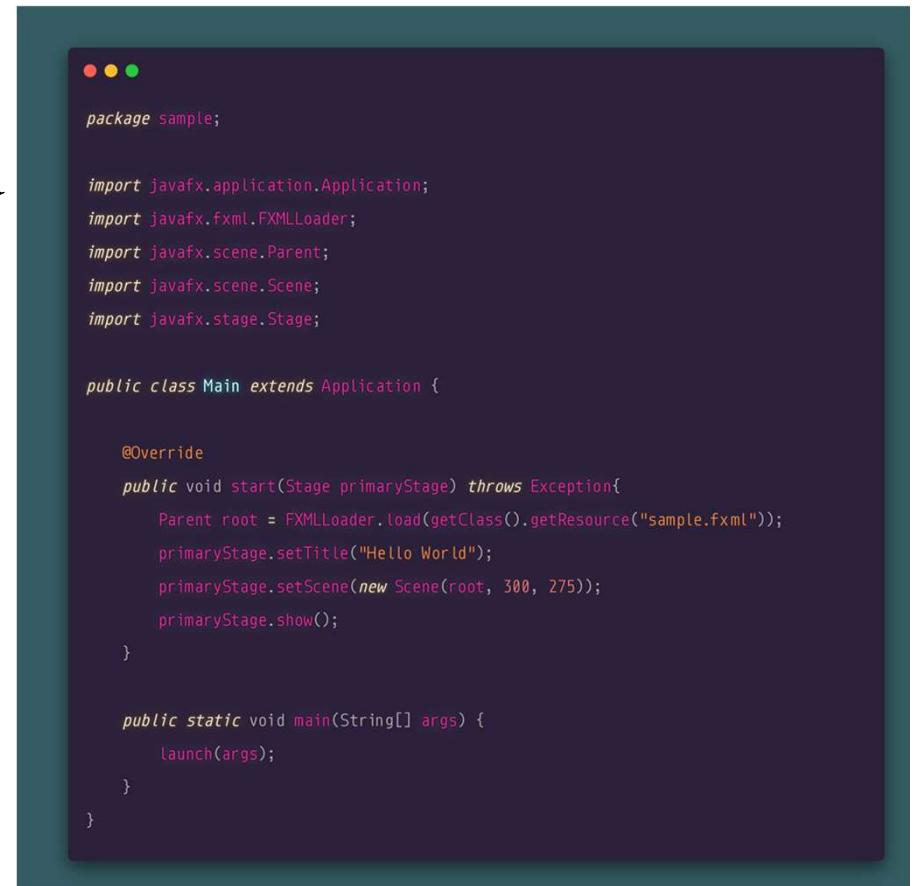
- ¶ If this is your first time creating a JavaFX Application, IntelliJ IDEA should “Download Library from Maven Repository” automatically





Install JavaFX in IntelliJ IDEA

& You will see IntelliJ IDEA
create sample code for you
in Main.java



```
package sample;

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

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root = FXMLLoader.load(getClass().getResource("sample.fxml"));
        primaryStage.setTitle("Hello World");
        primaryStage.setScene(new Scene(root, 300, 275));
        primaryStage.show();
    }

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



Oops...

Warning: If you now run the project it will compile but you will get this error:

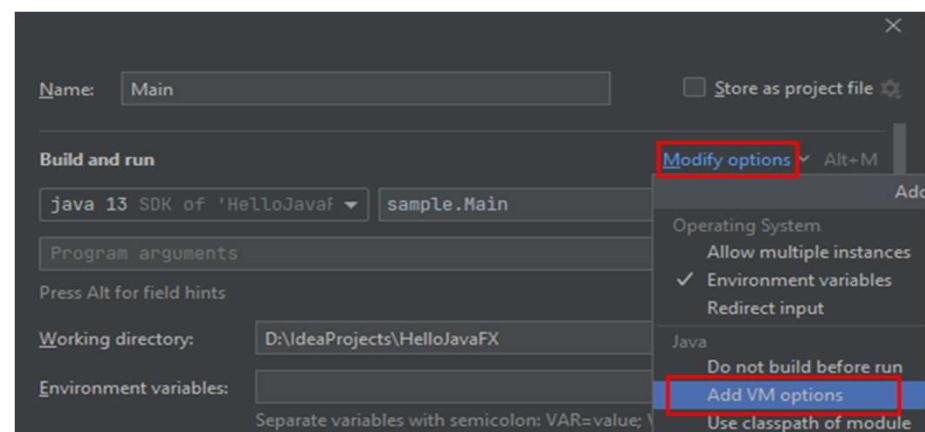
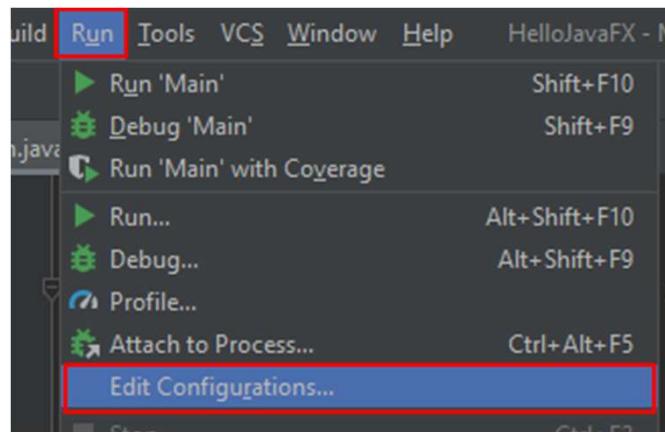
Error: JavaFX runtime components are missing, and are required to run this application

This error is shown since the Java 15 launcher checks if the main class extends `javafx.application.Application`. If that is the case, it is required to have the `javafx.graphics` module on the module-path.



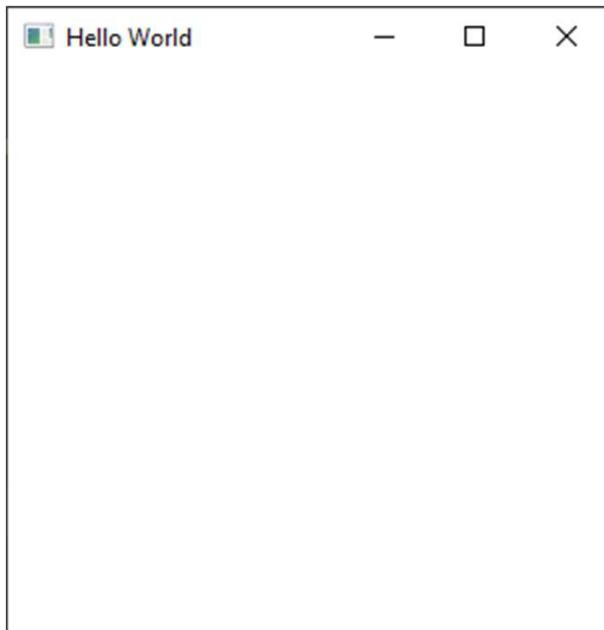
Add VM arguments

1. Run → Run Configurations...
2. Modify Options → VM arguments
 - ⌘ Linux/Mac: `--module-path /path/to/javafx-sdk-15.0.1/lib --add-modules javafx.controls,javafx.fxml`
 - Windows: `--module-path "\path\to\javafx-sdk-15.0.1\lib" --add-modules javafx.controls,javafx.fxml`





Install JavaFX in IntelliJ IDEA





2. Stage

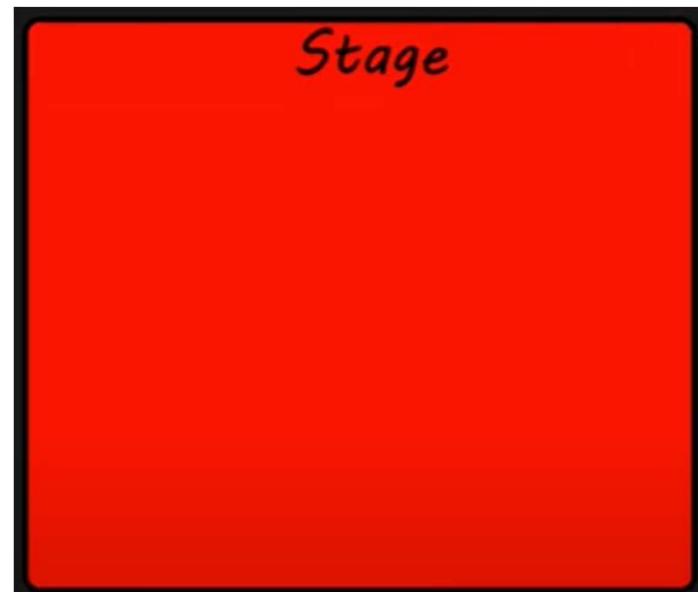


Stage

Top-level container to hold
the FX GUI applications

The Window of your
application

Similar to JFrame



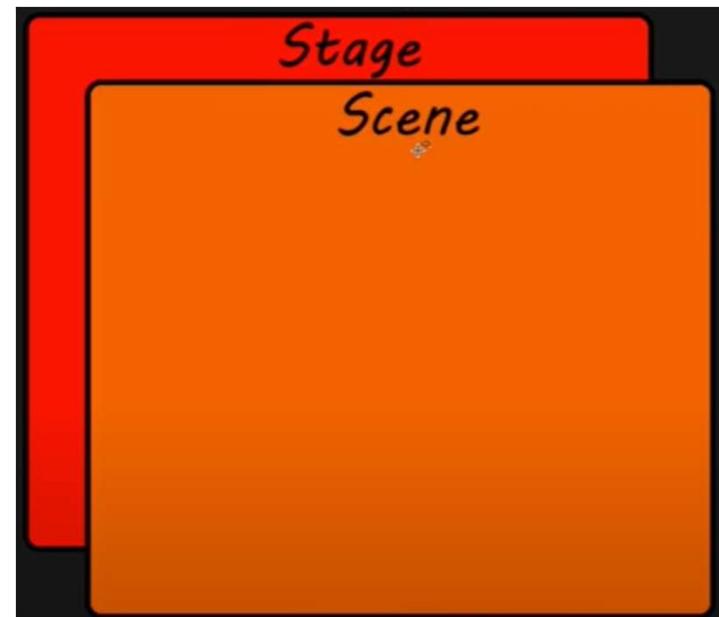


Scene

Add a scene to the “stage”

A drawing surface for
graphical content

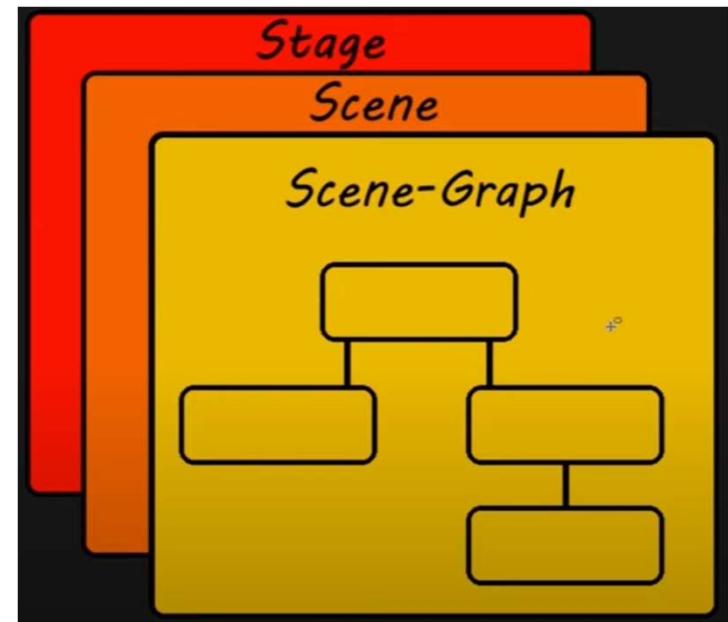
Similar to JPanel





Scene-Graph

A hierarchical tree that can hold and organize nodes

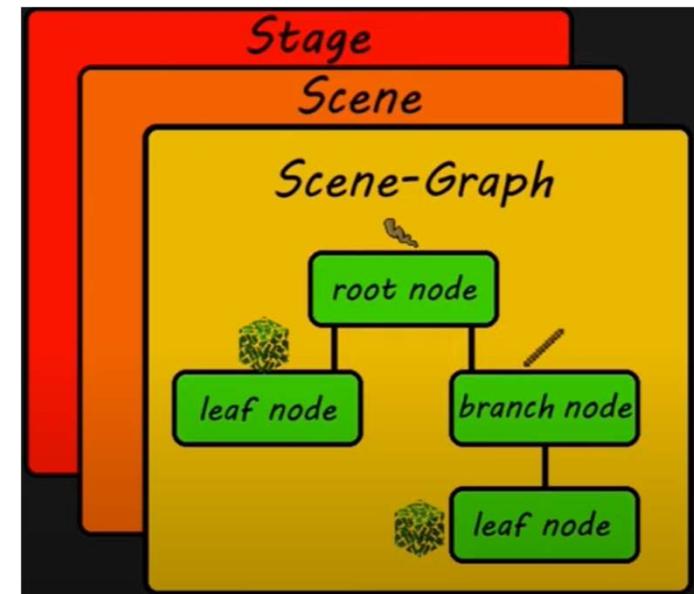




Nodes

Different components that can be added to the scene

Buttons, textbox, image...







```
*Main.java X application.css
1 package application;
2
3 import javafx.application.Application;□
4
5
6 public class Main extends Application {□
7
8
9     public static void main(String[] args) {
10         Launch(args);
11     }
12
13     @Override
14     public void start(Stage primaryStage) throws Exception {
15
16     }
17
18
19
20 }
21
22
```

Inherits the
“Application” class,
must implement the
abstract method “start”



```
*Main.java x application.css
1 package application;
2
3 import javafx.application.Application;□
4
5
6 public class Main extends Application {
7
8
9     public static void main(String[] args) {
10
11         Launch(args);
12
13     }
14
15
16     @Override
17     public void start(Stage primaryStage) throws Exception {
18
19
20     }
21 }
22
```

The code shows a Java file named Main.java. It imports the Application class from javafx.application. The main method is defined, and it calls the launch() method with args as parameters. The launch() method is highlighted with a red box.

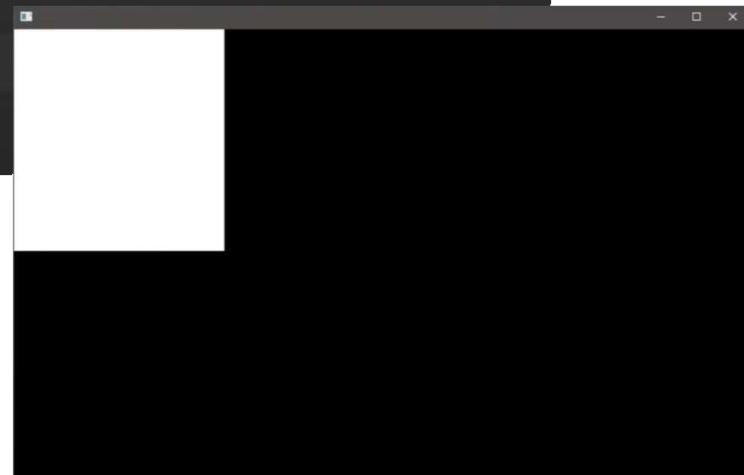
launch() is a static method inherited from the Application class, which will allow the “start()” method be called



Stage

```
@Override  
public void start(Stage stage) throws Exception {  
  
    //Stage stage = new Stage();  
  
    } }
```

If we run this, a blank window will appear, but when you resize it, the background will be weird

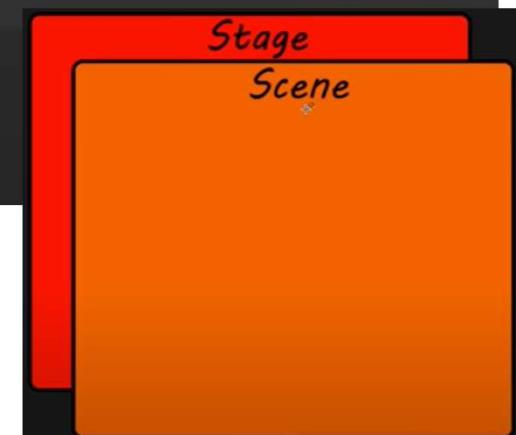




Scene

```
16•     @Override  
17     public void start(Stage stage) throws Exception {  
18  
19         //Stage stage = new Stage();  
20         Scene scene = new Scene();|  
21         stage.show();  
22     }  
23 }
```

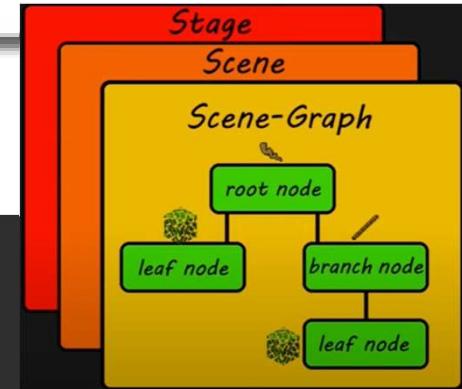
But we need at least a node for it to show





Group

```
@Override  
public void start(Stage stage) throws Exception {  
  
    //Stage stage = new Stage();  
    Group root = new Group();      "Group" can be think  
    Scene scene = new Scene(root);  as a group of nodes  
  
    stage.show();  
}
```



Put your group into the Scene

```
import javafx.application.Application;  
import javafx.stage.Stage;  
import javafx.scene.Group;  
import javafx.scene.Scene;
```

Remember to import the corresponding libraries



Group

```
@Override  
public void start(Stage stage) throws Exception {  
  
    //Stage stage = new Stage();  
    Group root = new Group();  
    Scene scene = new Scene(root);  
  
    stage.setScene(scene);  
    stage.show();  
}
```

Set the scene into the stage

Now if you run your app,
the background will be
resizable



Other Customization

```
@Override  
public void start(Stage stage) throws Exception {  
  
    //Stage stage = new Stage();  
    Group root = new Group();  
    Scene scene = new Scene(root,Color.BLACK);  
    Set the color of the scene,  
    remember to import the color  
    package  
  
    stage.setTitle("Stage Demo Program w00t w00t");  
    Set the title of the stage  
  
    stage.setWidth(420);  
    stage.setHeight(420);  
    stage.setResizable(false);  
    Make the window not resizable  
    stage.setX(50);  
    stage.setY(50);  
    Set where the window will appear on  
    screen
```



Changing App Icon

The screenshot shows a JavaFX project named "HelloFX". The file structure under "src" includes "application", "Main.java", "application.css", and "icon.png". A red box highlights the "icon.png" file. The code editor shows the Main.java file with the following code:

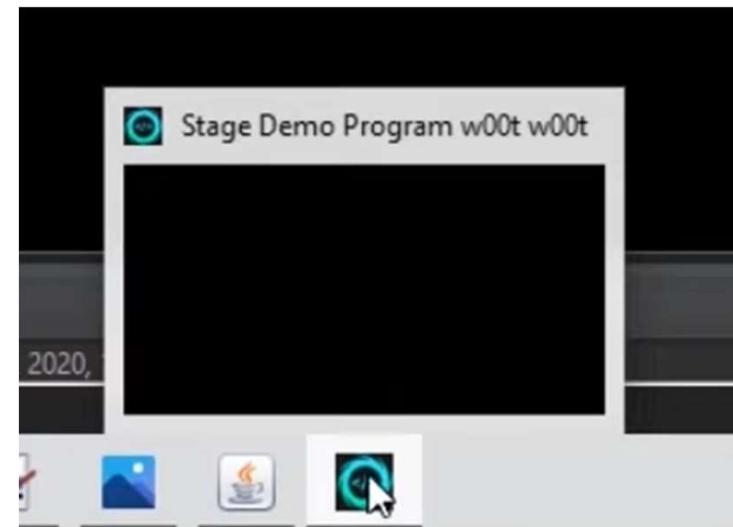
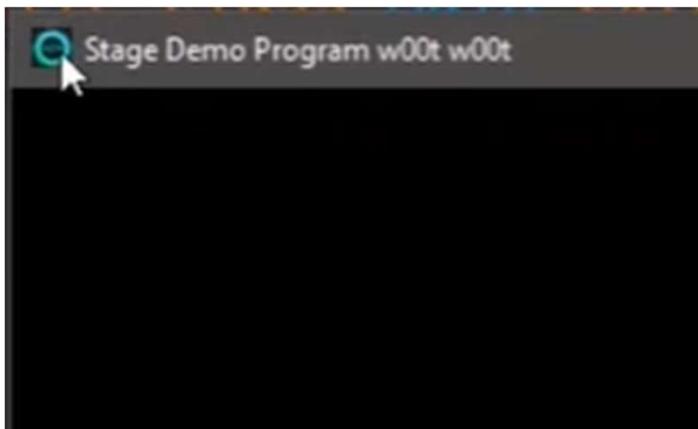
```
@Override  
public void start(Stage stage) throws Exception {  
  
    //Stage stage = new Stage();  
    Group root = new Group();  
    Scene scene = new Scene(root, Color.BLACK);  
  
    Image icon = new Image("icon.png");  
    stage.getIcons().add(icon);  
    stage.setTitle("Stage Demo Program w00t w00t");  
  
    stage.setScene(scene);  
    stage.show();  
}
```

A callout box points to the "icon.png" file in the file tree with the text "Put your file under the src folder". Another callout box points to the highlighted code in the editor with the text "Make an Image object with the path to your icon file and add it to the stage".

Make an Image object with the path to your icon file and add it to the stage



Changing App Icon

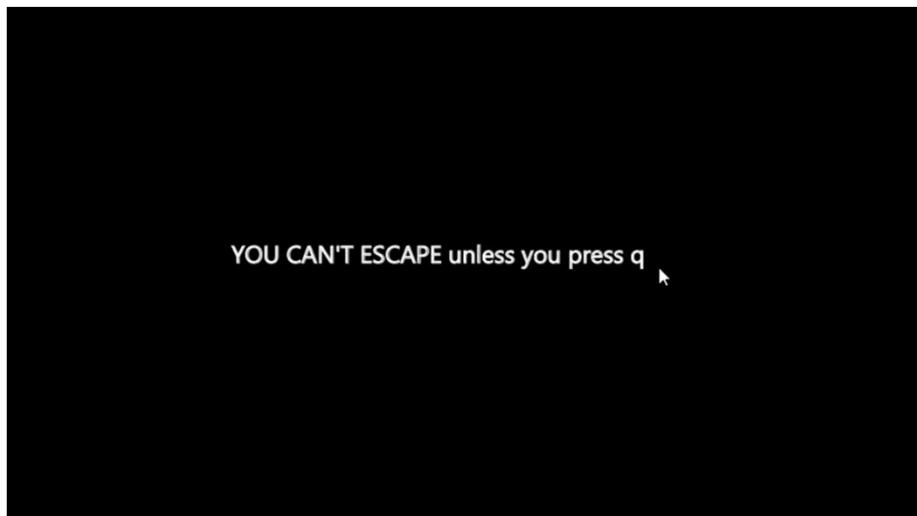




Full Screen Mode

```
stage.setFullScreen(true);    The hint message shown to users  
stage.setFullScreenExitHint("YOU CAN'T ESCAPE unless you press q");  
stage.setFullScreenExitKeyCombination(KeyCombination.valueOf("q"));
```

change the key combination to exit full screen





3. Scenes



Text

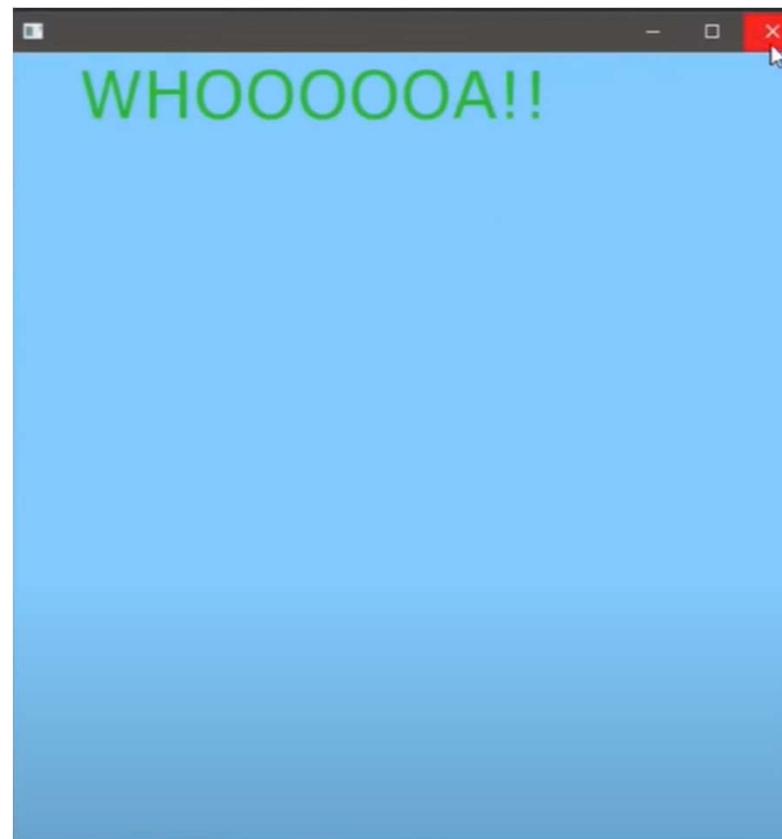
```
@Override  
public void start(Stage primaryStage) throws Exception {  
  
    Group root = new Group();  
    Scene scene = new Scene(root,600,600,Color.LIGHTSKYBLUE);  
    Stage stage = new Stage();  
  
    Text text = new Text();  
    text.setText("WHOOOOA!!");  
    text.setX(50);  
    text.setY(50);  
    text.setFont(Font.font("Verdana",50));  
    text.setFill(Color.LIMEGREEN);  
  
    root.getChildren().add(text);  
    stage.setScene(scene);  
    stage.show();
```

Make a Text object

Add the text object to the root group



Text





Line

```
Line line = new Line();
line.setStartX(200);
line.setStartY(200);
line.setEndX(500);
line.setEndY(200);
line.setStrokeWidth(5);
line.setStroke(Color.RED);
line.setOpacity(0.5);
line.setRotate(45);|
```

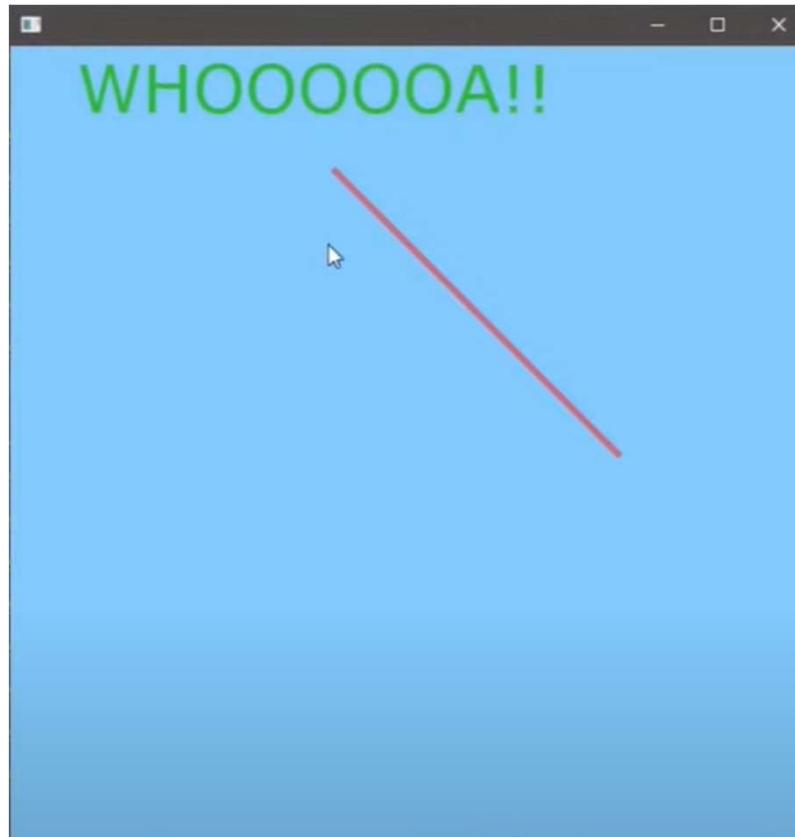
Make a Line object

```
root.getChildren().add(text);
root.getChildren().add(line);
```

Add the line object to the root group



Line





Rectangle

```
Rectangle rectangle = new Rectangle();
rectangle.setX(100);
rectangle.setY(100);
rectangle.setWidth(100);
rectangle.setHeight(100);
rectangle.setFill(Color.BLUE);
rectangle.setStrokeWidth(5);
rectangle.setStroke(Color.BLACK);
```

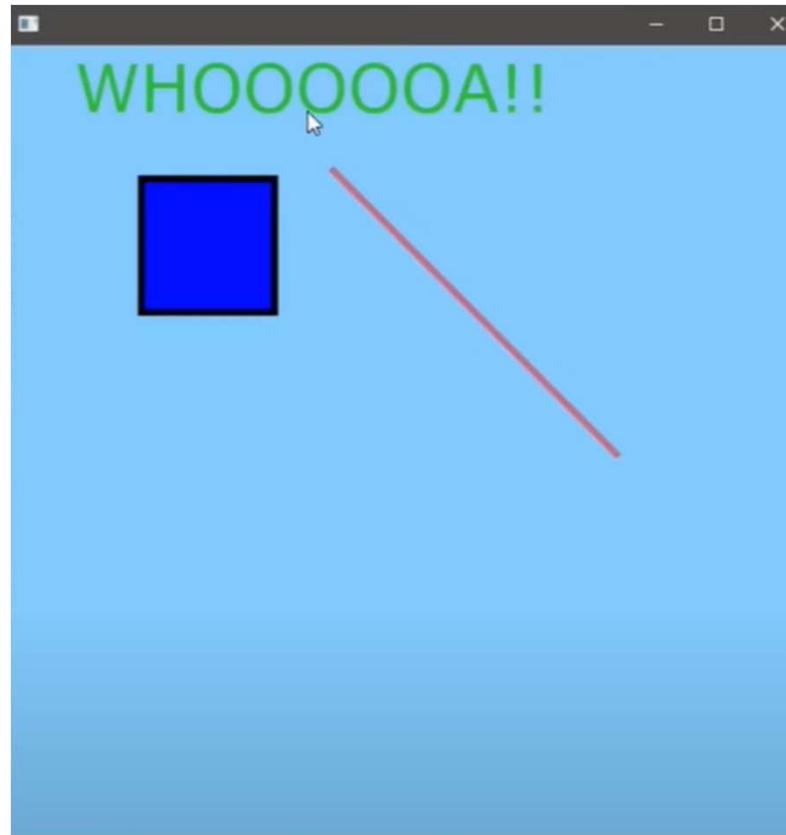
Make a Rectangle object

```
root.getChildren().add(text);
root.getChildren().add(line);
root.getChildren().add(rectangle);
stage.setScene(scene);
stage.show();
```

Add the rectangle object to
the root group



Rectangle





Triangle

```
Polygon triangle = new Polygon();
triangle.getPoints().setAll(
    200.0,200.0,
    300.0,300.0,
    200.0,300.0
);
triangle.setFill(Color.YELLOW);
```

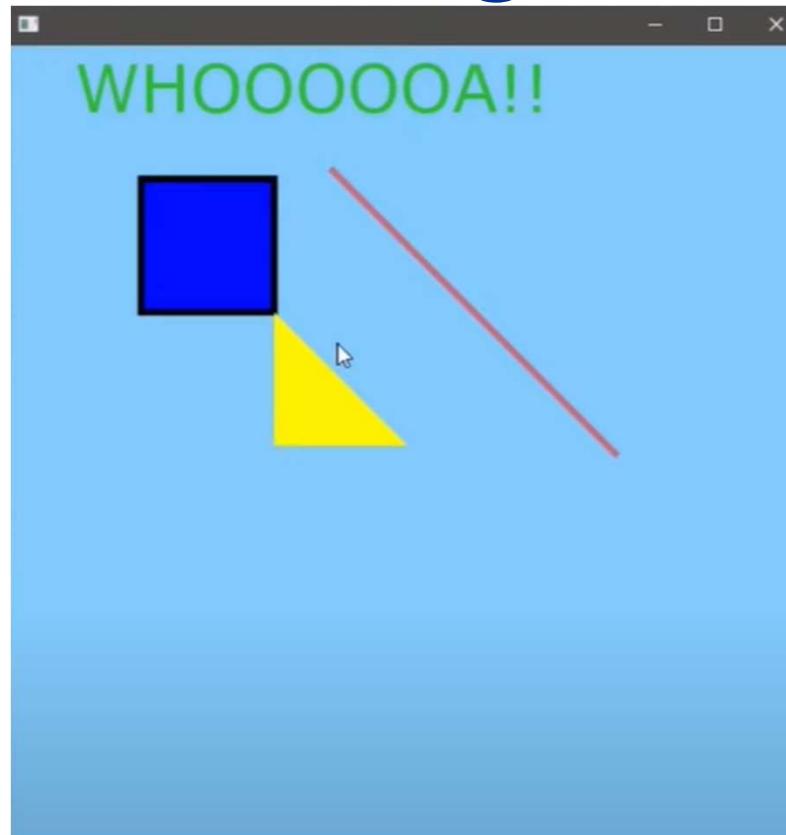
Make a Polygon object and define all of its points

```
root.getChildren().add(text);
root.getChildren().add(line);
root.getChildren().add(rectangle);
root.getChildren().add(triangle);
```

Add the polygon object to the root group



Triangle





Circle

```
Circle circle = new Circle();
circle.setCenterX(350);
circle.setCenterY(350);
circle.setRadius(50);
circle.setFill(Color.ORANGE);
```

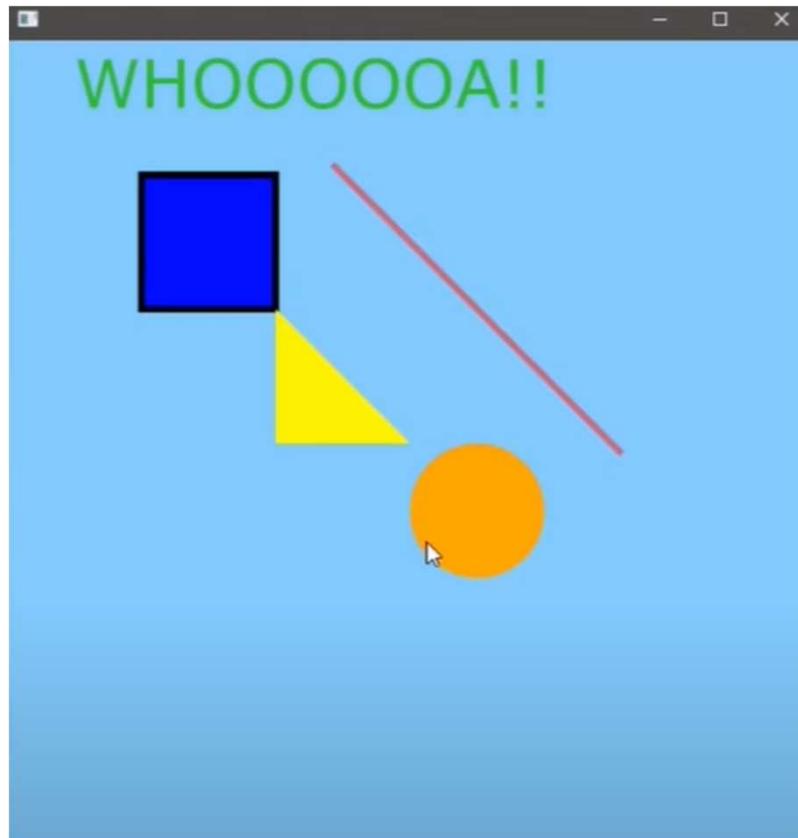
Make a Circle object

```
root.getChildren().add(text);
root.getChildren().add(line);
root.getChildren().add(rectangle);
root.getChildren().add(triangle);
root.getChildren().add(circle);
```

Add the circle object to the root group

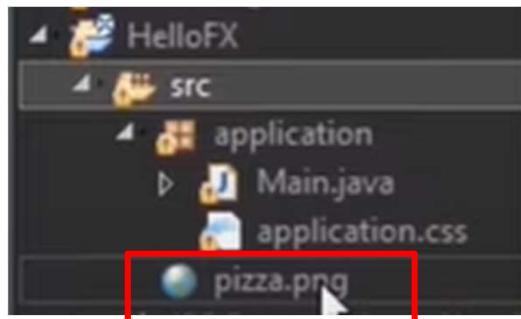


Circle





Insert Image



Place file under src folder

```
Image image = new Image("pizza.png");
```

```
ImageView imageView = new ImageView(image);
```

```
imageView.setX(400);
```

```
imageView.setY(400);
```

Create an Image object with
your file

```
root.getChildren().add(text);
```

```
root.getChildren().add(line);
```

```
root.getChildren().add(rectangle);
```

```
root.getChildren().add(triangle);
```

```
root.getChildren().add(circle);
```

```
root.getChildren().add(imageView);
```

```
stage.setScene(scene);
```

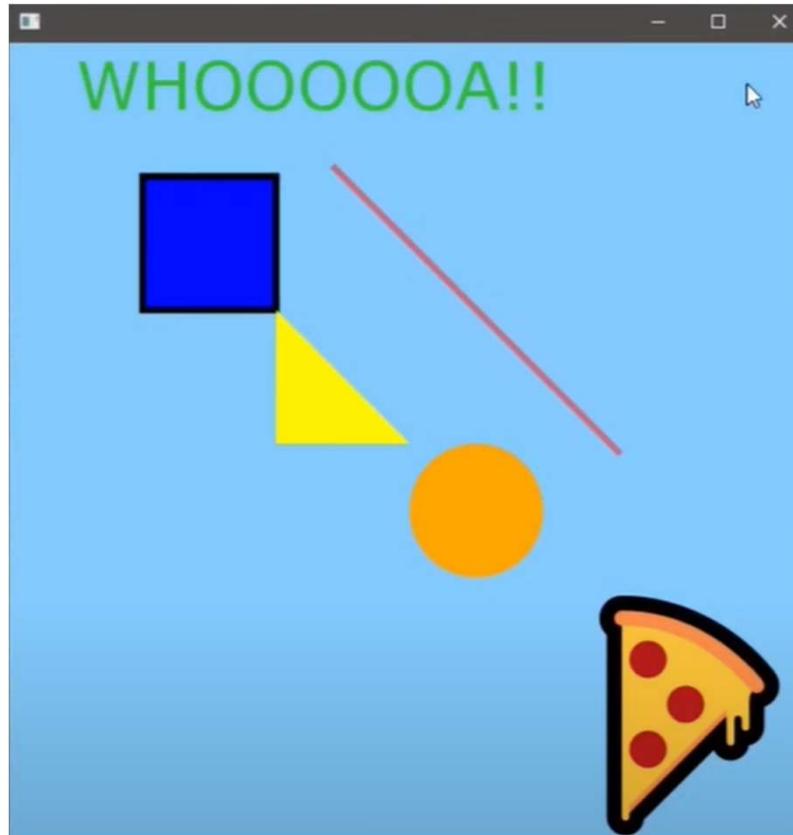
```
stage.show();
```

Create an imageView object
with your image object

Add the imageView object to
the root group



Image





4.

Scene Builders



Download and Install Scene Builder

Google search results for "javafx scene builder":

Search bar: javafx scene builder

Filter: All

About 385,000 results (0.37 seconds)

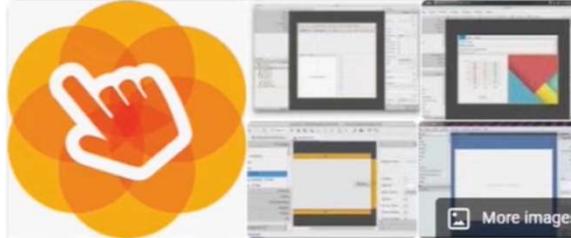
1. [Scene Builder - Gluon](http://gluonhq.com/products/scene-builder)
Scene Builder works with the JavaFX ecosystem – official controls, community projects, and ...
Scene Builder is free and open source, but is backed by Gluon.

2. [JavaFX Scene Builder Information - Oracle](http://www.oracle.com/java/technologies/javase/javafx/)
JavaFX Scene Builder is a visual layout tool that lets users quickly design JavaFX application user interfaces, without coding.

3. [JavaFX Scene Builder 1.x Archive - Oracle](http://www.oracle.com/java/technologies/javafxscenebui...)
WARNING: These versions of JavaFX Scene Builder may include components that do not contain the latest security patches and are not recommended for use in ...

People also ask:

How do I use JavaFX Scene Builder?



Scene Builder

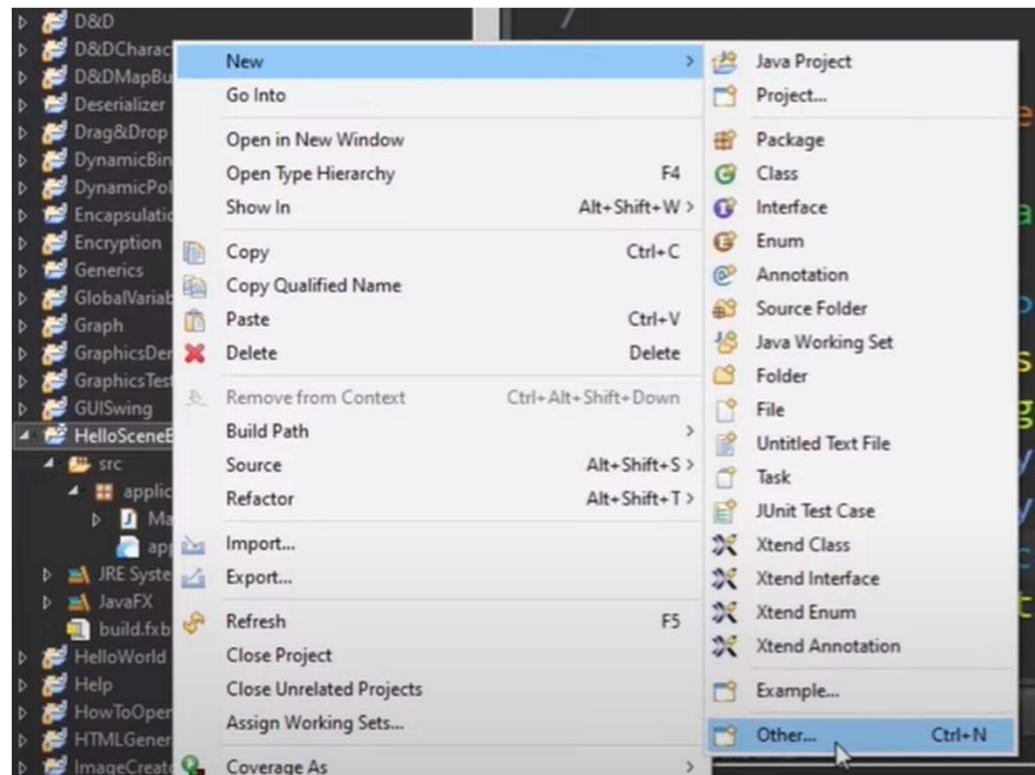
JavaFX Scene Builder is a visual layout tool that lets users quickly design JavaFX application user interfaces, without coding.

Initial release date: 2012
Developer: Oracle Corporation
Programming language: Java



Eclipse Installation

1. Create a FX project
2. Right click on the project > New > Other

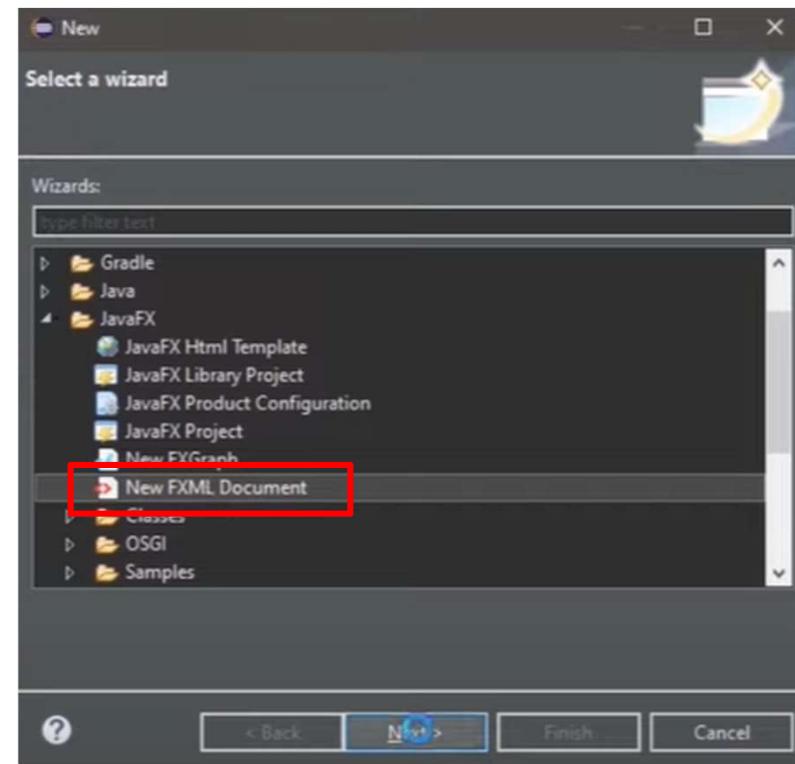




Eclipse Installation

3. Select **New FXML Document**

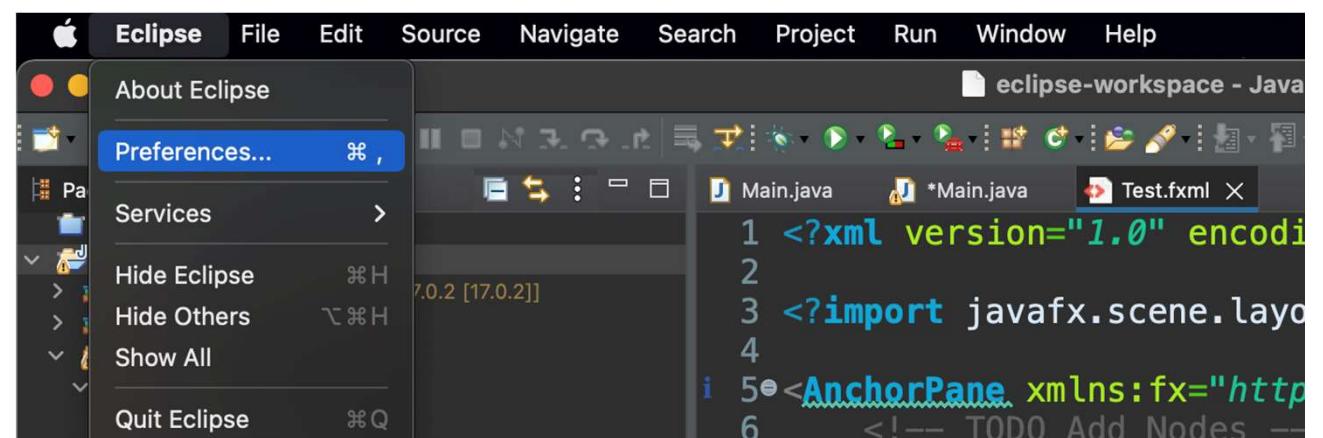
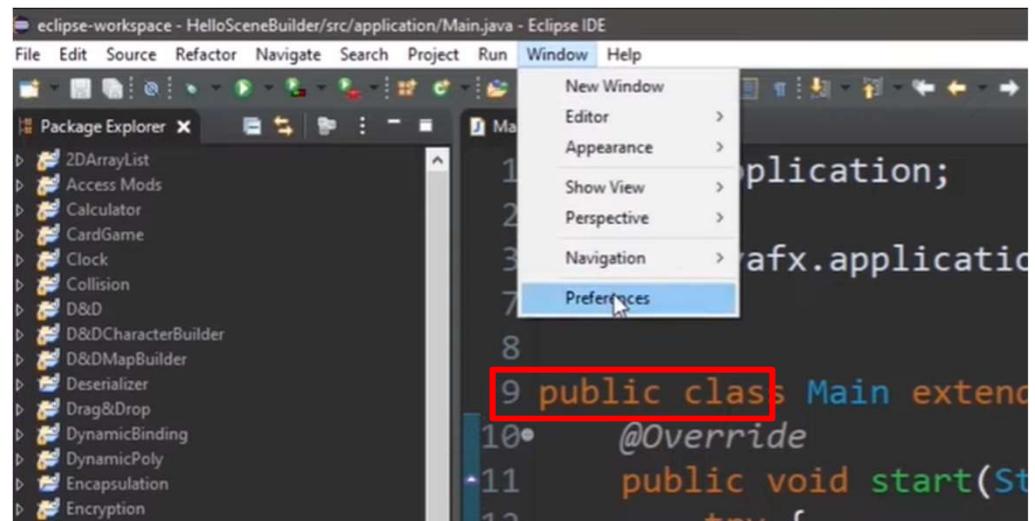
4. Type in the Name
for your FXML File
and Finish





Eclipse Installation

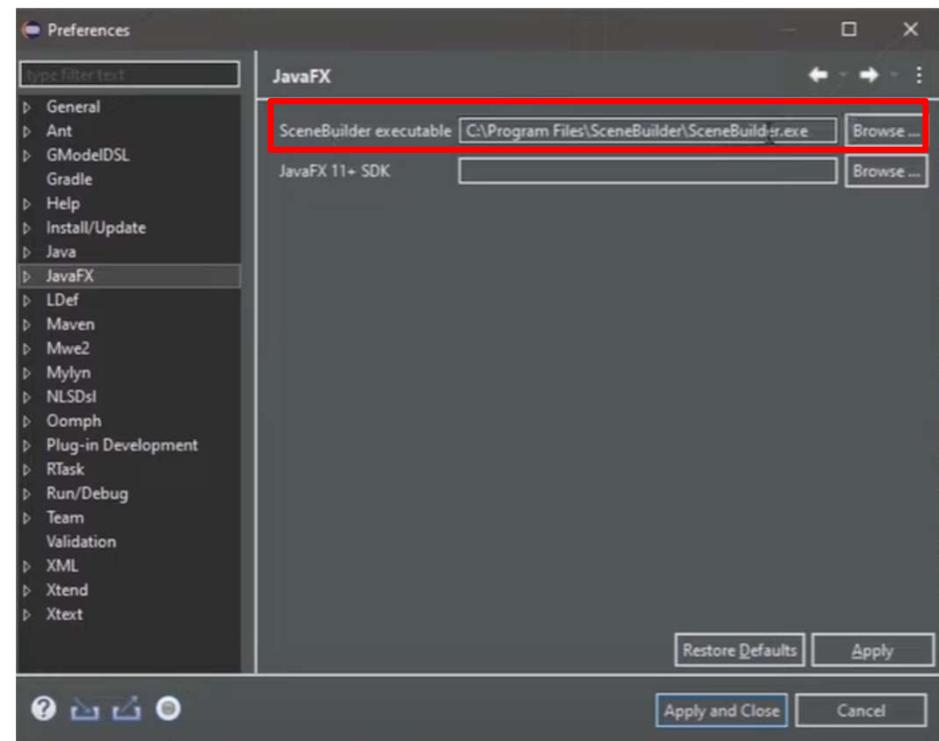
5. Window > Preferences





Eclipse Installation

6. Browse for the path of the SceneBuilder executable file you downloaded

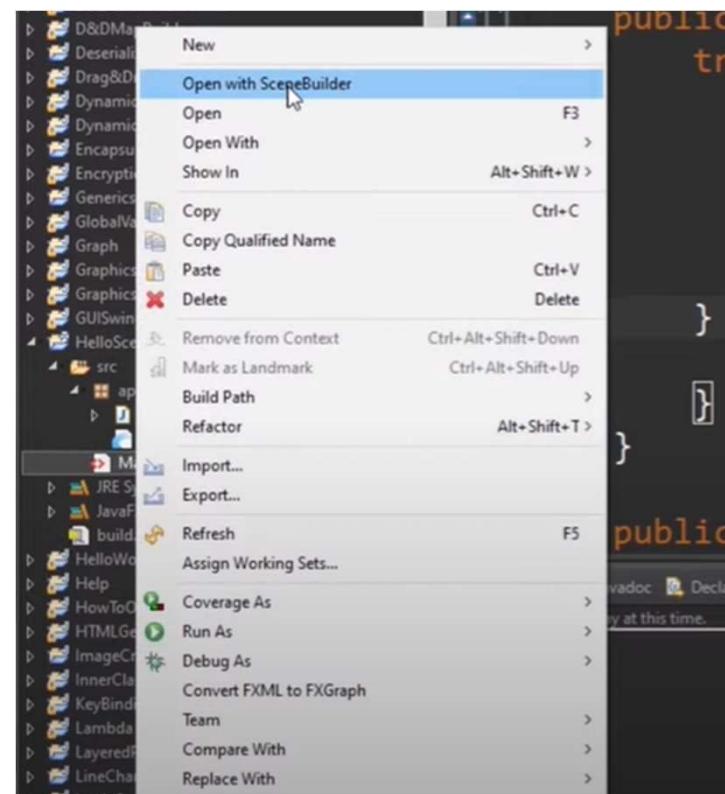


Mac: /Applications/SceneBuilder.app



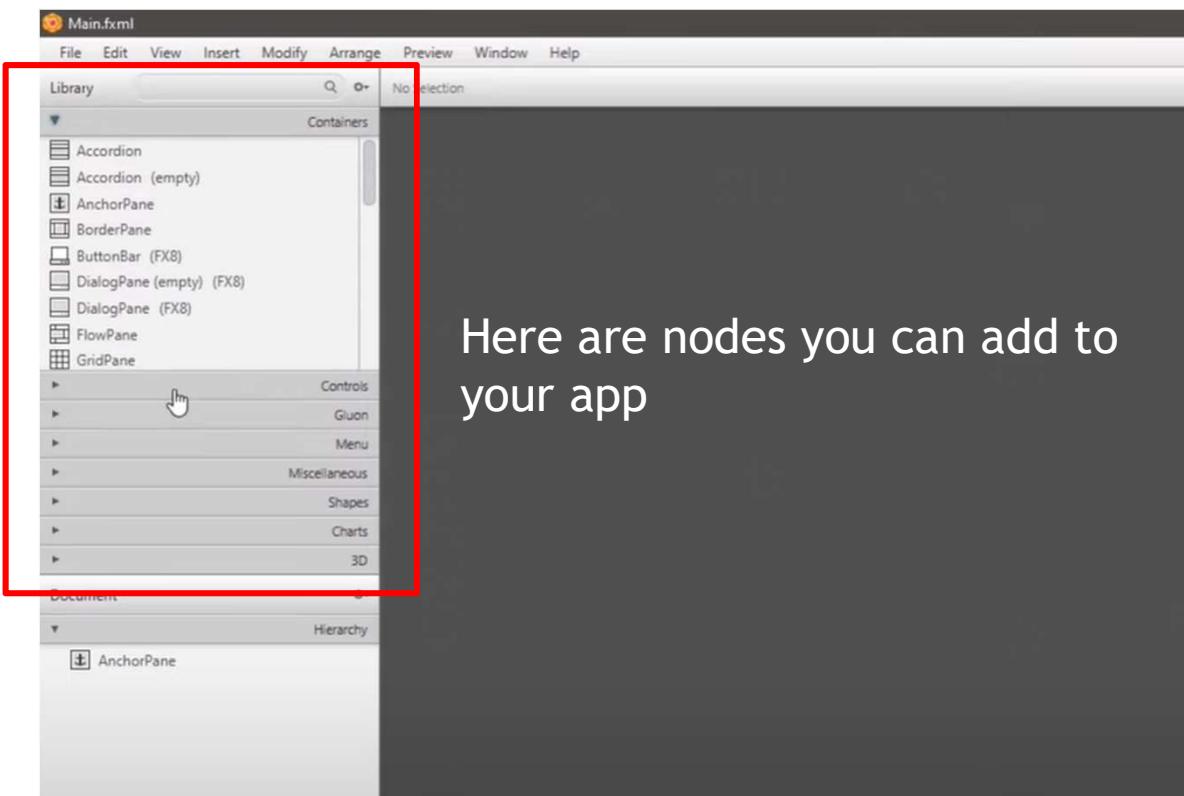
Eclipse Installation

7. Right click on the FXML file > Open with SceneBuilder

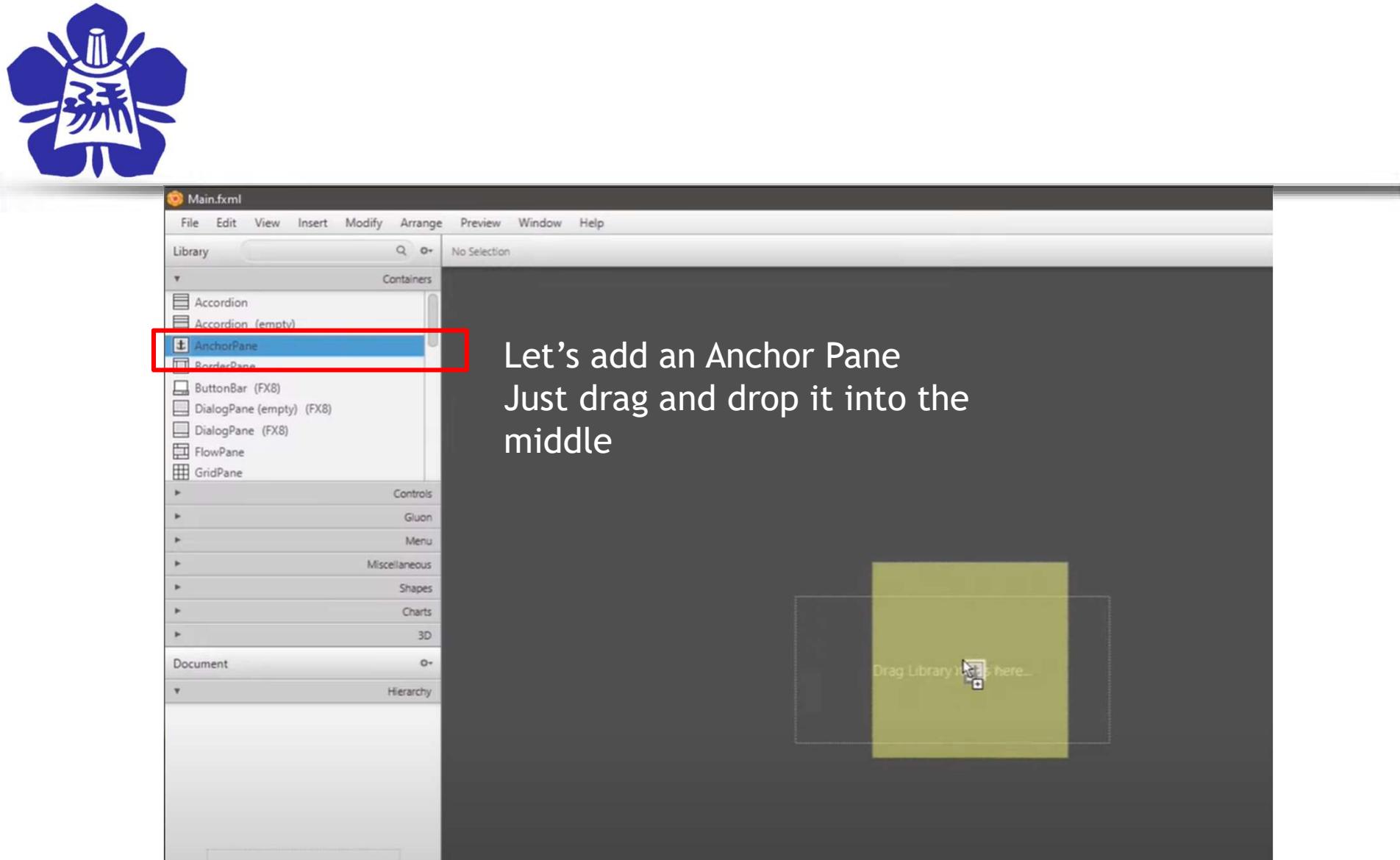




SceneBuilder



Here are nodes you can add to
your app



Let's add an Anchor Pane
Just drag and drop it into the
middle



Main.fxml

File Edit View Insert Modify Arrange Preview Window Help

New Ctrl+N

New from Template

Open... Ctrl+O

Open Recent

Save Ctrl+S

Save As... Ctrl+Shift+S

Revert to Saved

Reveal in Explorer

Import

Include

Close Window Ctrl+W

Preferences... Ctrl+,

Quit

Ellipse Charts

3D

Document Hierarchy

AnchorPane

- Button Button
- Button Button
- Button Button
- Button Button

Arc

No Selection

Containers Controls GUI Menu Miscellaneous Shapes

Let's add some buttons and shapes
Remember to save your file

Button

Button

Button

Button



This is the default start()
when you create a FX project

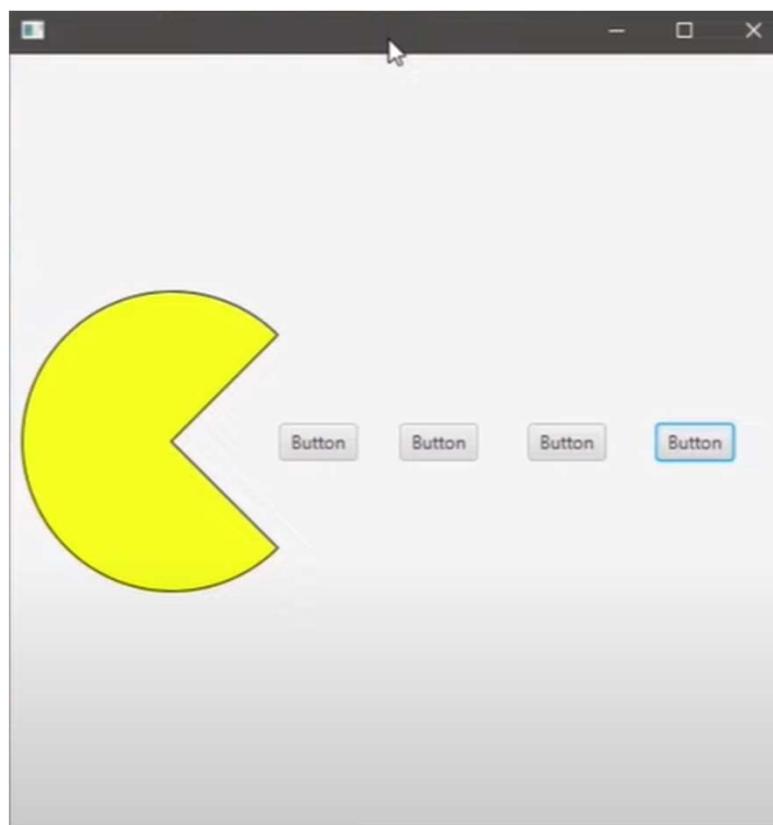
```
public void start(Stage primaryStage) {  
    try {  
        BorderPane root = new BorderPane();  
        Scene scene = new Scene(root, 400, 400);  
        scene.getStylesheets().add(getClass().getResource("application.css"))  
    }  
}
```

```
public void start(Stage primaryStage) {  
    try {  
        Parent root = FXMLLoader.load(getClass().getResource("/Main.fxml"));  
        Scene scene = new Scene(root);  
        scene.getStylesheets().add(getClass().getResource("application.css"));  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    } catch(Exception e) {  
        e.printStackTrace();  
    }  
}
```

Change the BorderPane root into a Parent object that will load your FXML file as the resource
And remove the default scene size



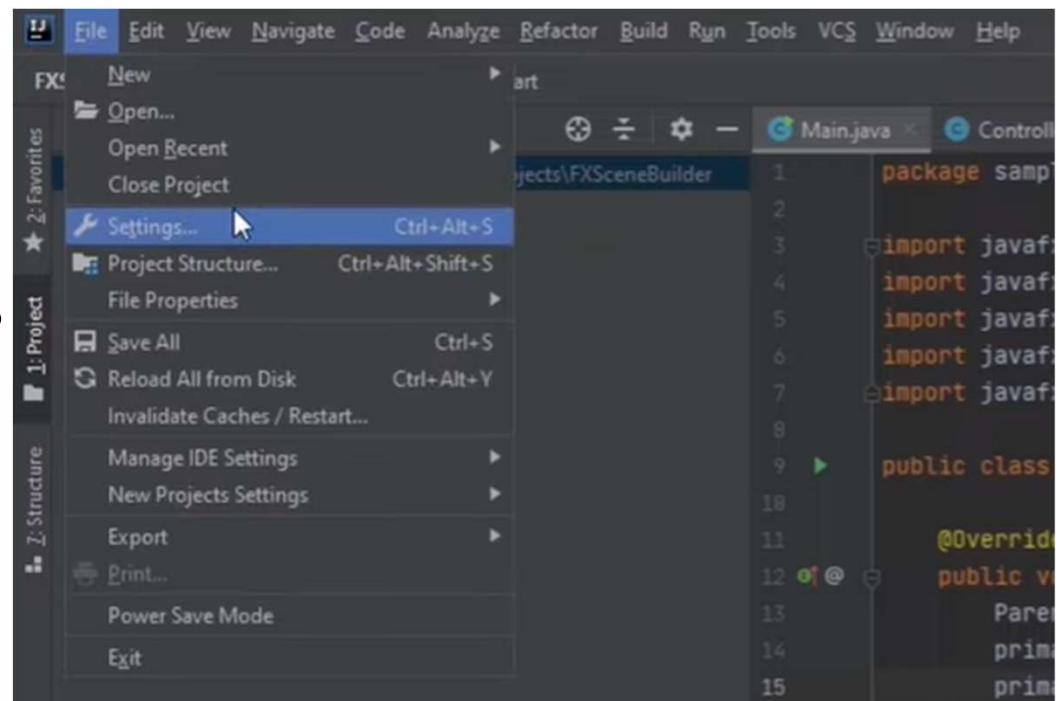
And your app should be running!





IntelliJ Installation

1. Create a FX project
2. Go to File > Settings

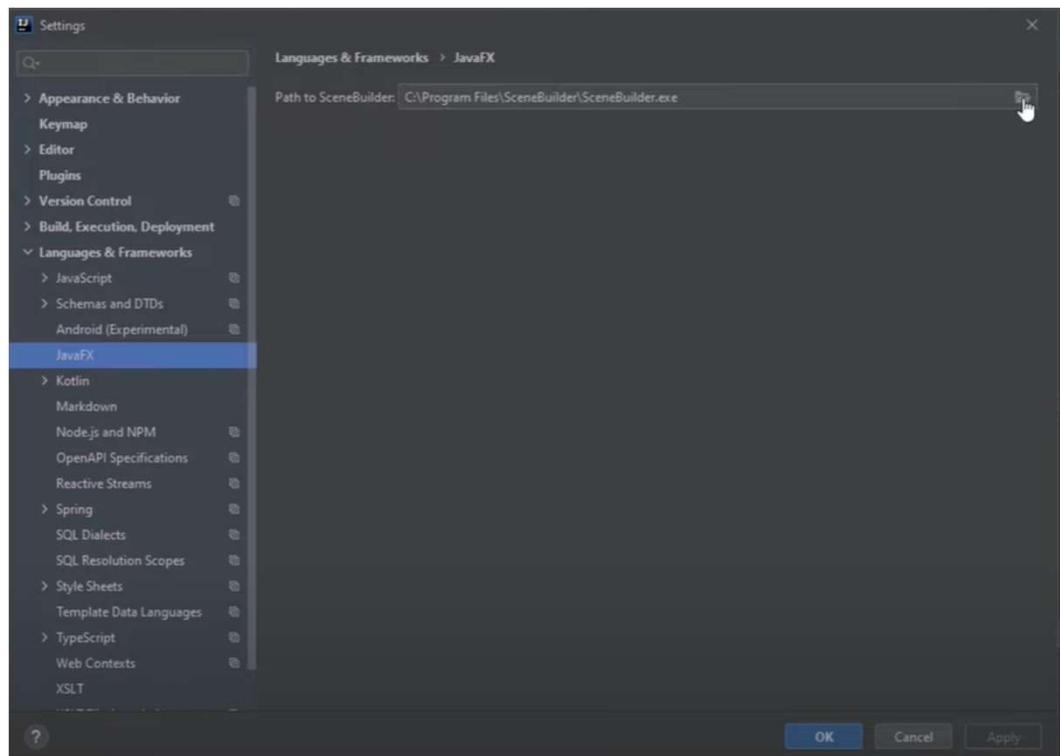




IntelliJ Installation

3. Under **Languages & Frameworks > JavaFX**,
browse your path to the
SceneBuilder executable

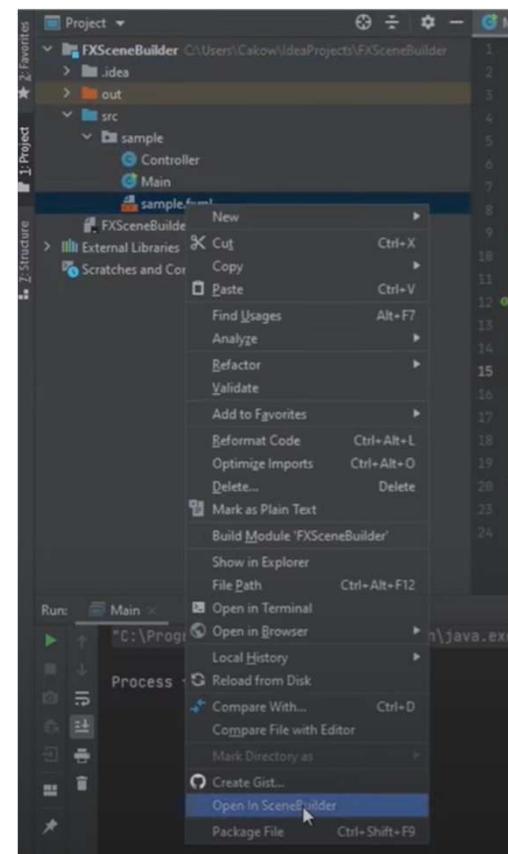
4. Click **Apply** and **OK**





IntelliJ Installation

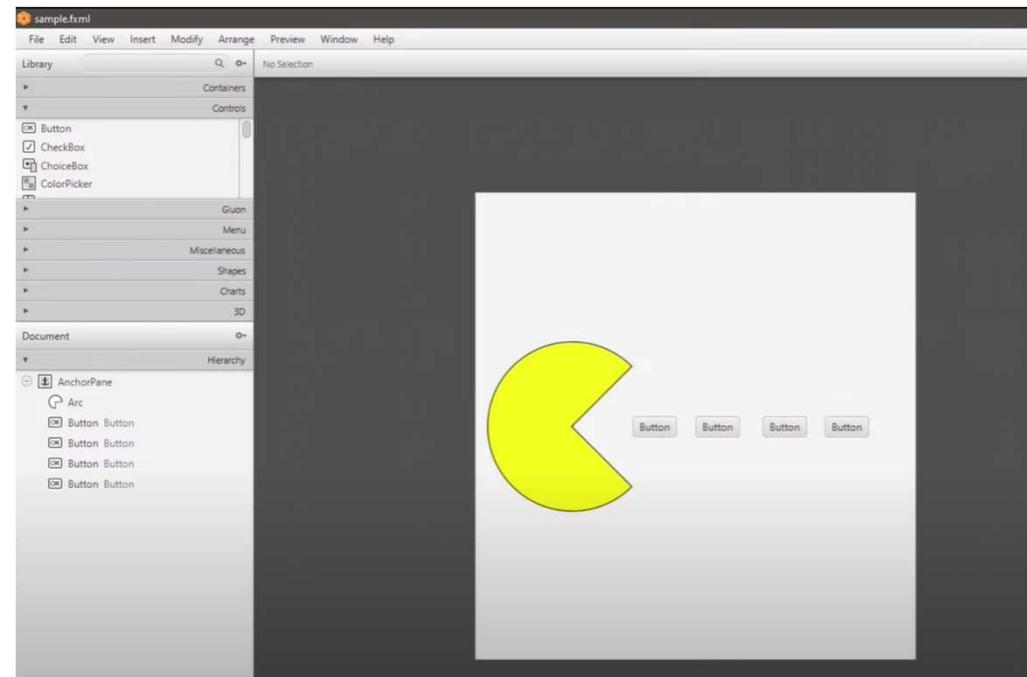
5. Right click on the FXML file > Open in SceneBuilder





IntelliJ Installation

6. Do the same thing
in SceneBuilder





IntelliJ Installation

```
@Override  
public void start(Stage primaryStage) throws Exception{  
    Parent root = FXMLLoader.load(getClass().getResource( name: "sample.fxml"));  
    primaryStage.setTitle("Hello World");  
    primaryStage.setScene(new Scene(root));  
    primaryStage.show();  
}
```

link the fxml file to the root node

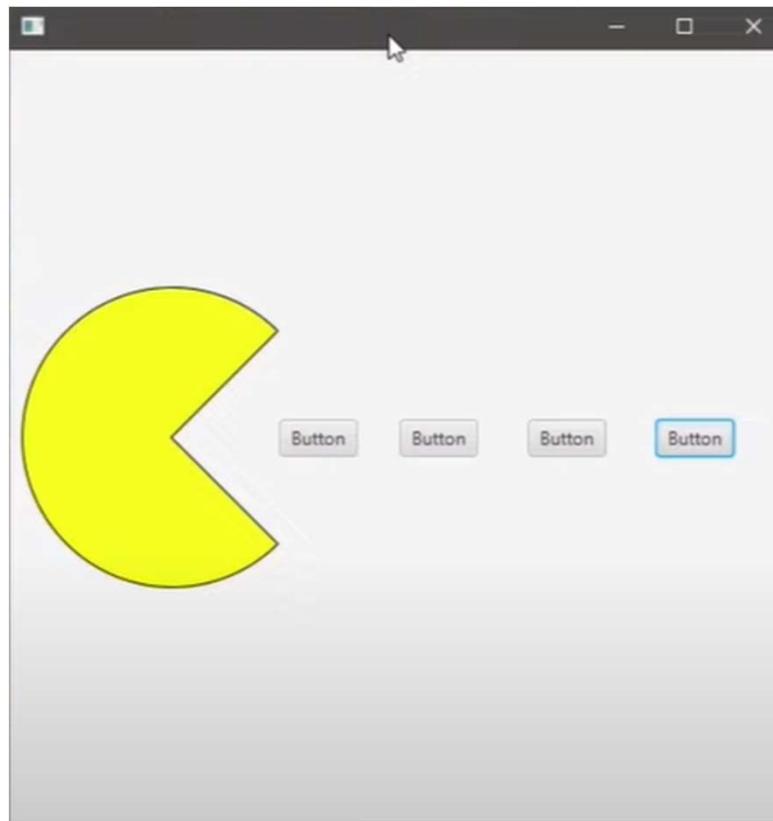
And delete the default size of the scene



5. Event Handling



The buttons here have no reaction, why?





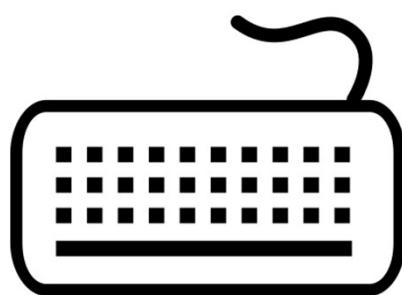
Because no events are handled



What are events?



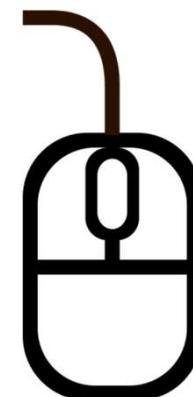
Any events that happens



A button has been clicked

The mouse has moved

An arrow key has been
pressed



A file has been uploaded



What happens with the events?



Programmers handle them

A button has been clicked → go to another page

The mouse has moved → change color

An arrow key has been pressed → change walking
direction

A file has been uploaded → show a popup window



Programs use Listeners to listen for events

Listener must be
24H always
listening if an event
has been fired

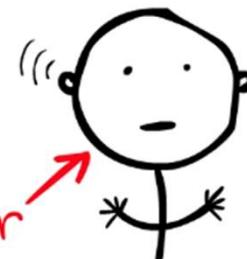
a button
has been
clicked!

someone pressed
a keyboard key!

the animation
has started!

a file has
finished
loading!

event listener

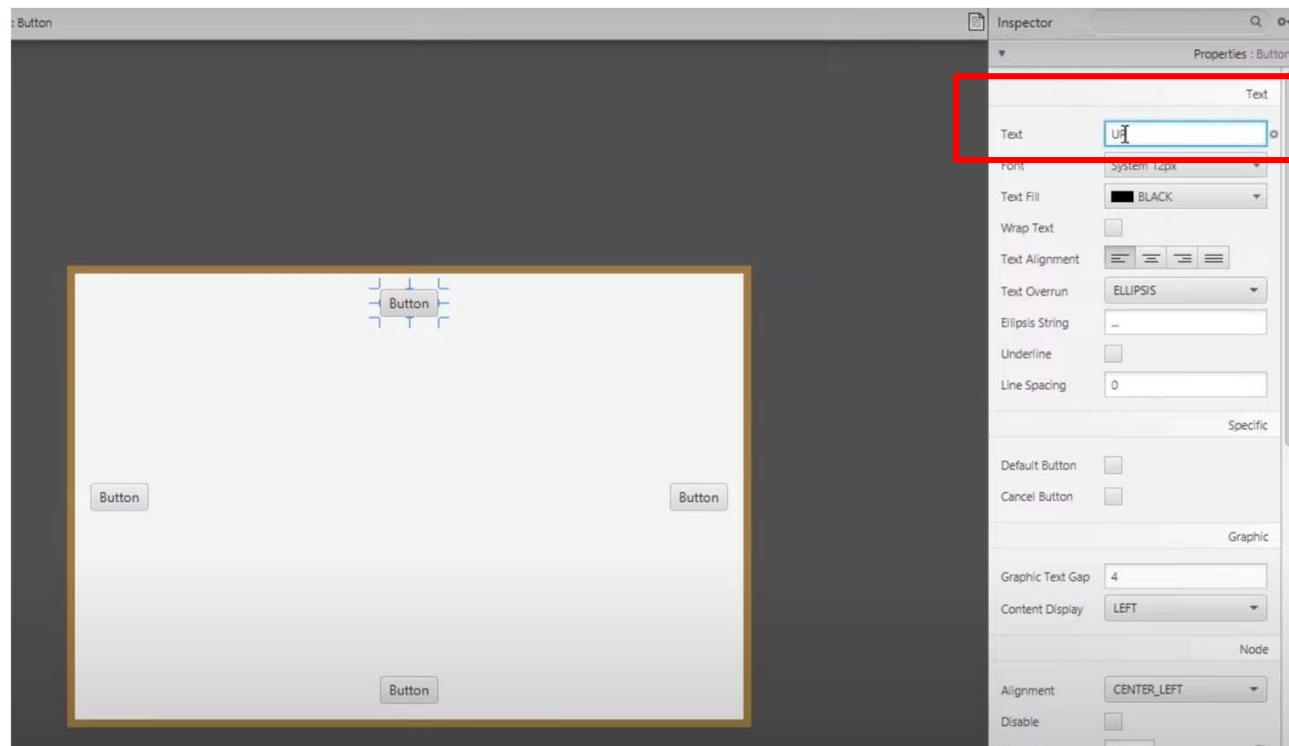




Let's make some event handlers



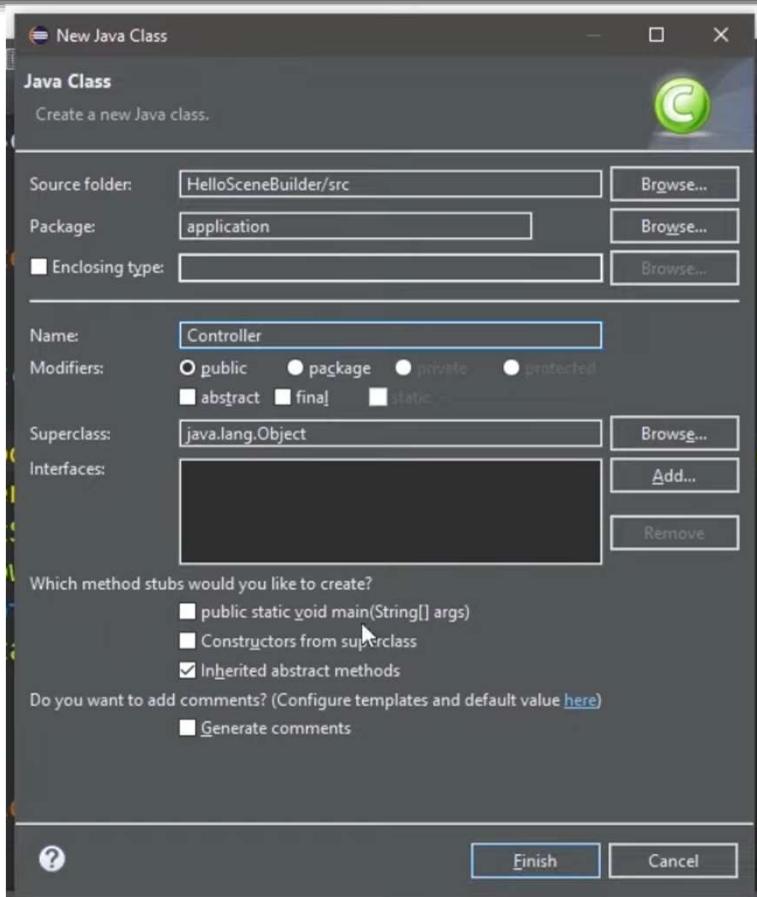
Make a FXML file with 4 buttons



change the text of
the button under
Properties > Text

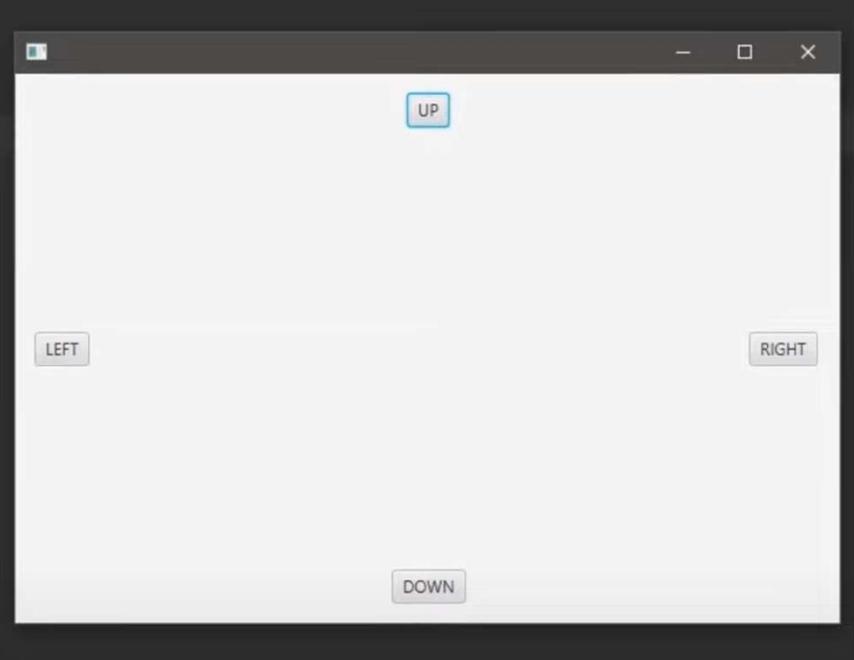


Add a new Class





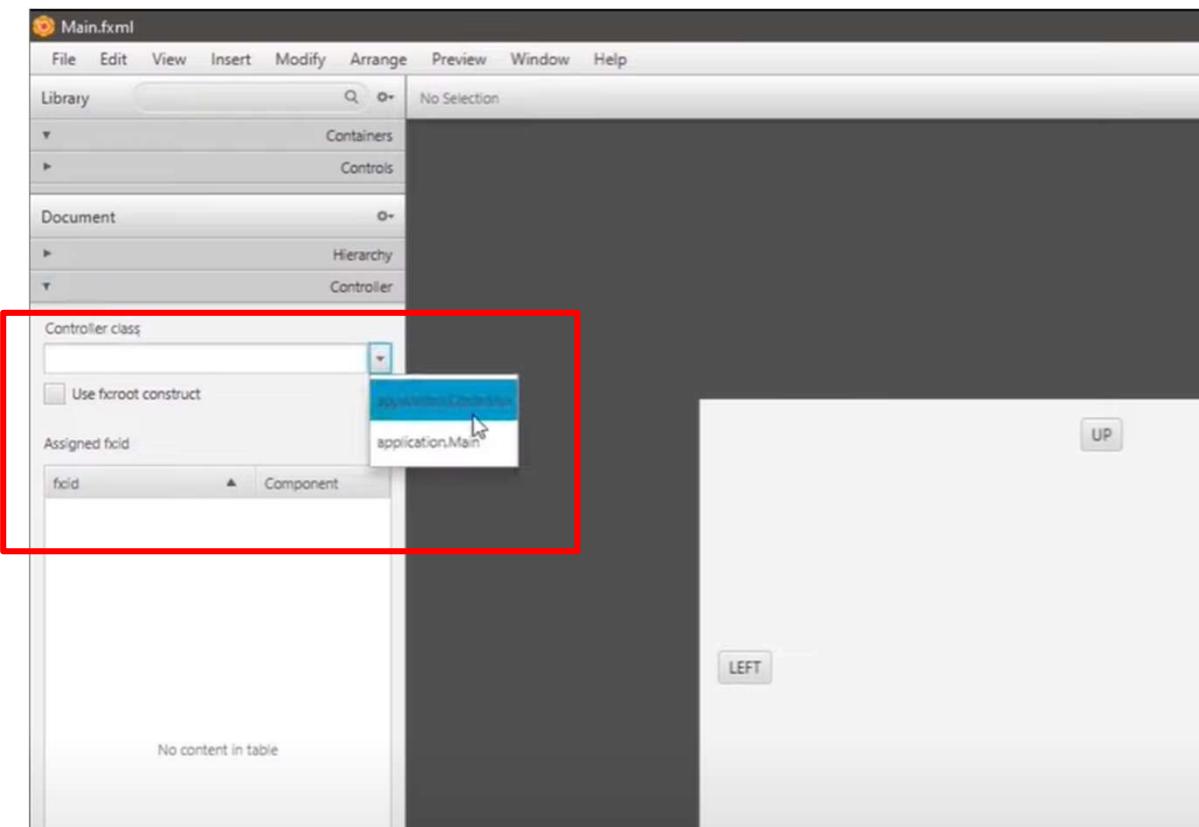
```
1 package application;
2
3 import javafx.event.ActionEvent;
4
5 public class Controller {
6
7     public void up(ActionEvent e) {
8         System.out.println("UP");
9     }
10    public void down(ActionEvent e) {
11        System.out.println("DOWN");
12    }
13    public void left(ActionEvent e) {
14        System.out.println("LEFT");
15    }
16    public void right(ActionEvent e) {
17        System.out.println("RIGHT");
18    }
19}
```

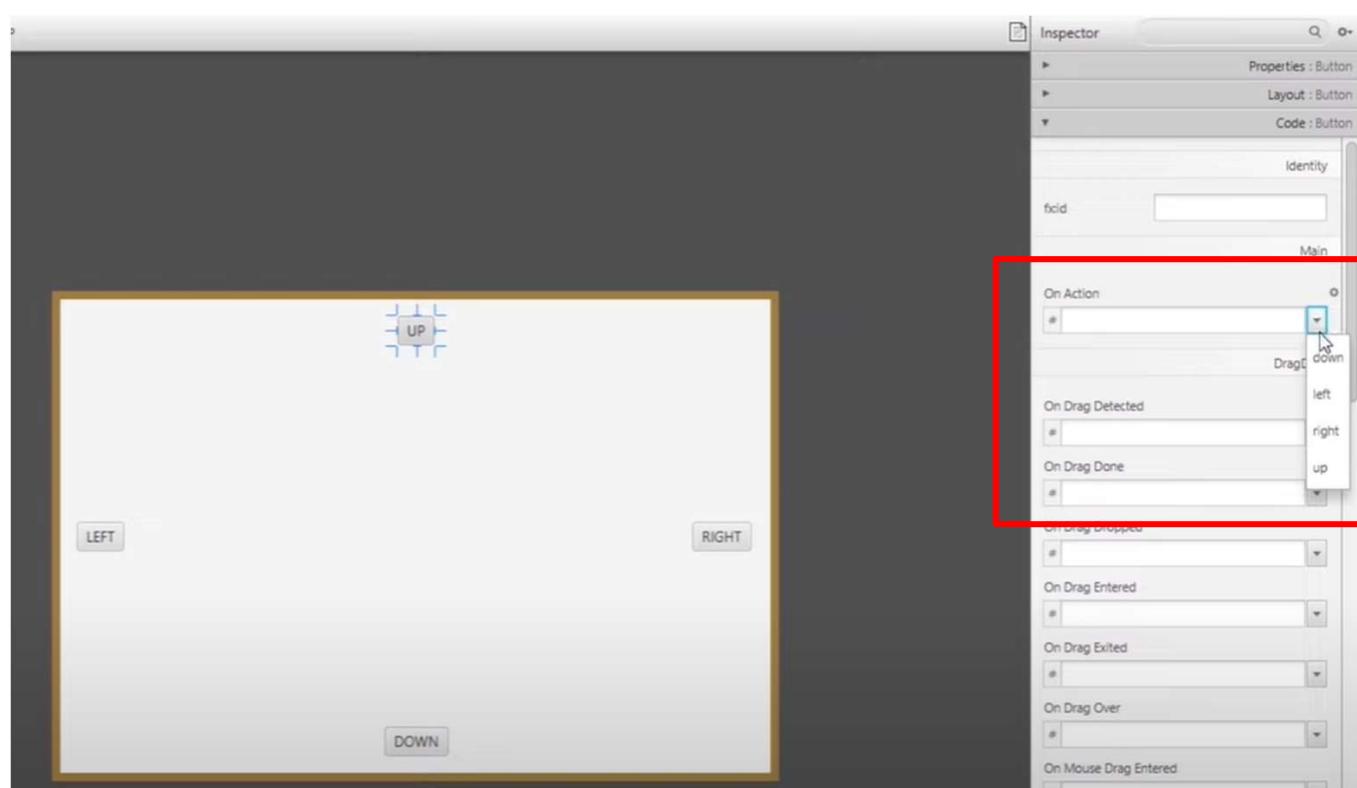


Write functions for each button that will have ActionEvent as the parameter
Make sure to import ActionEvent from javafx



Go back to the
SceneBuilder and
**select Controller
Class** > Select your
controller class you
just created





And select the corresponding function for On Action



Now text should appear when you click on the buttons

The screenshot shows an IDE interface with three tabs: Main.java, Main.fxml, and Controller.java. The Main.java tab contains the following Java code:

```
7 import javafx.scene.Scene;
8
9
10 public class Main {
11
12     @Override
13     public void start(Stage stage) throws Exception {
14         try {
15             Parent root = FXMLLoader.load(getClass().getResource("Main.fxml"));
16             Scene scene = new Scene(root);
17             stage.setScene(scene);
18             stage.show();
19         } catch(Exception e) {
20             e.printStackTrace();
21         }
22     }
23
24     public static void main(String[] args) {
25         launch(args);
26     }
}
```

A separate window titled "Main" is displayed, showing four buttons labeled "UP", "DOWN", "LEFT", and "RIGHT". The "UP" button is highlighted with a gray border.

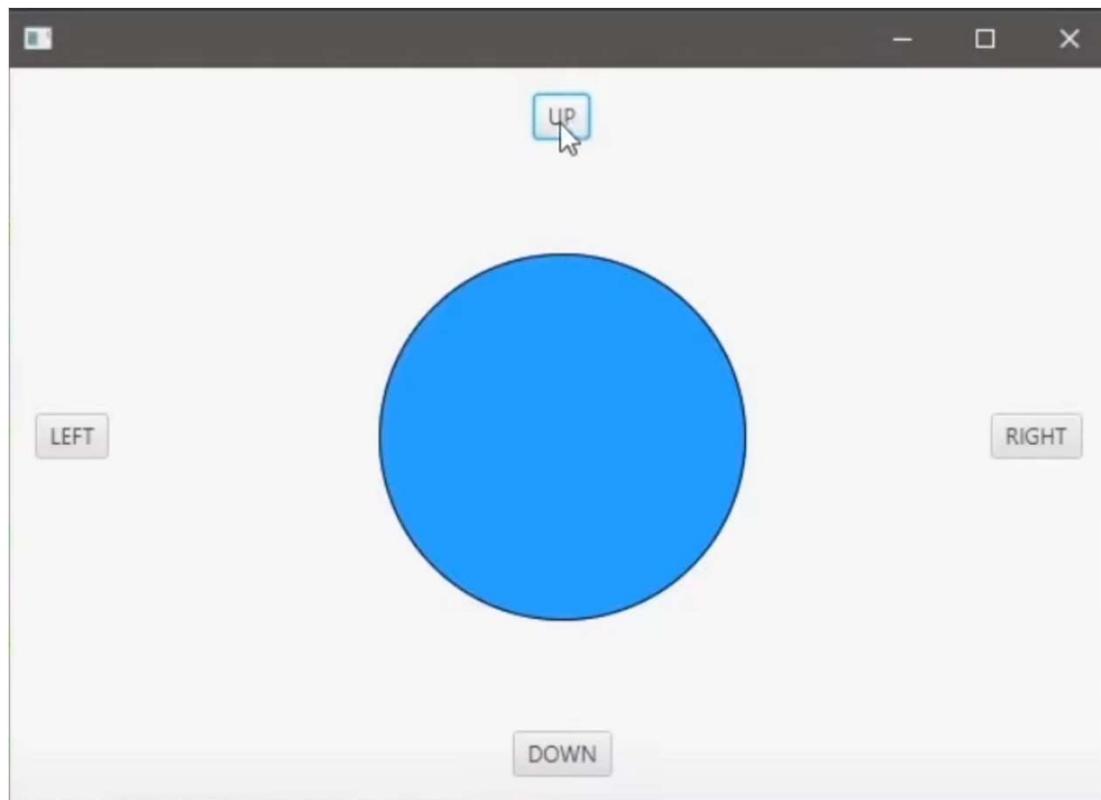
In the bottom-left corner of the IDE, there is a "Console" tab with the following output:

```
Main (55) [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (Dec 28, 2020, 2:01:17 PM)
UP
UP
DOWN
DOWN
DOWN
```

The first three lines ("UP", "UP", "DOWN") are highlighted with a red box.



Lab





Make a circle in SceneBuilder and name its fxid to "myCircle"

```
public class Controller {  
    @FXML  
    private Circle myCircle;  
    private double x;  
    private double y;  
  
    public void up(ActionEvent e) {  
        //System.out.println("UP");  
        myCircle.setCenterY(y-=10);  
    }  
    public void down(ActionEvent e) {  
        //System.out.println("DOWN");  
        myCircle.setCenterY(y+=10);  
    }  
    public void left(ActionEvent e) {  
        //System.out.println("LEFT");  
        myCircle.setCenterX(x-=10);  
    }  
}
```



Your FXML file should look like this

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.scene.control.Button?>
4 <?import javafx.scene.layout.AnchorPane?>
5 <?import javafx.scene.shape.Circle?>
6
7 <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" pref-
8     <children>
9         <Button layoutX="285.0" layoutY="14.0" mnemonicParsing="false" onAction="#up" text="UP" />
10        <Button layoutX="534.0" layoutY="188.0" mnemonicParsing="false" onAction="#right" text="RIGHT" />
11        <Button layoutX="274.0" layoutY="361.0" mnemonicParsing="false" onAction="#down" text="DOWN" />
12        <Button layoutX="14.0" layoutY="188.0" mnemonicParsing="false" onAction="#left" text="LEFT" />
13        <Circle fx:id="myCircle" fill="DODGERBLUE" layoutX="301.0" layoutY="201.0" radius="100.0" stroke-
14     </children>
15 </AnchorPane>    the fxid of the shape
16
```

onAction = the function being fired when the button is on action

the fxid of the shape