JavaFX

CPSC 233: Introduction to Computer Science for Computer Science Majors II
Winter 2022

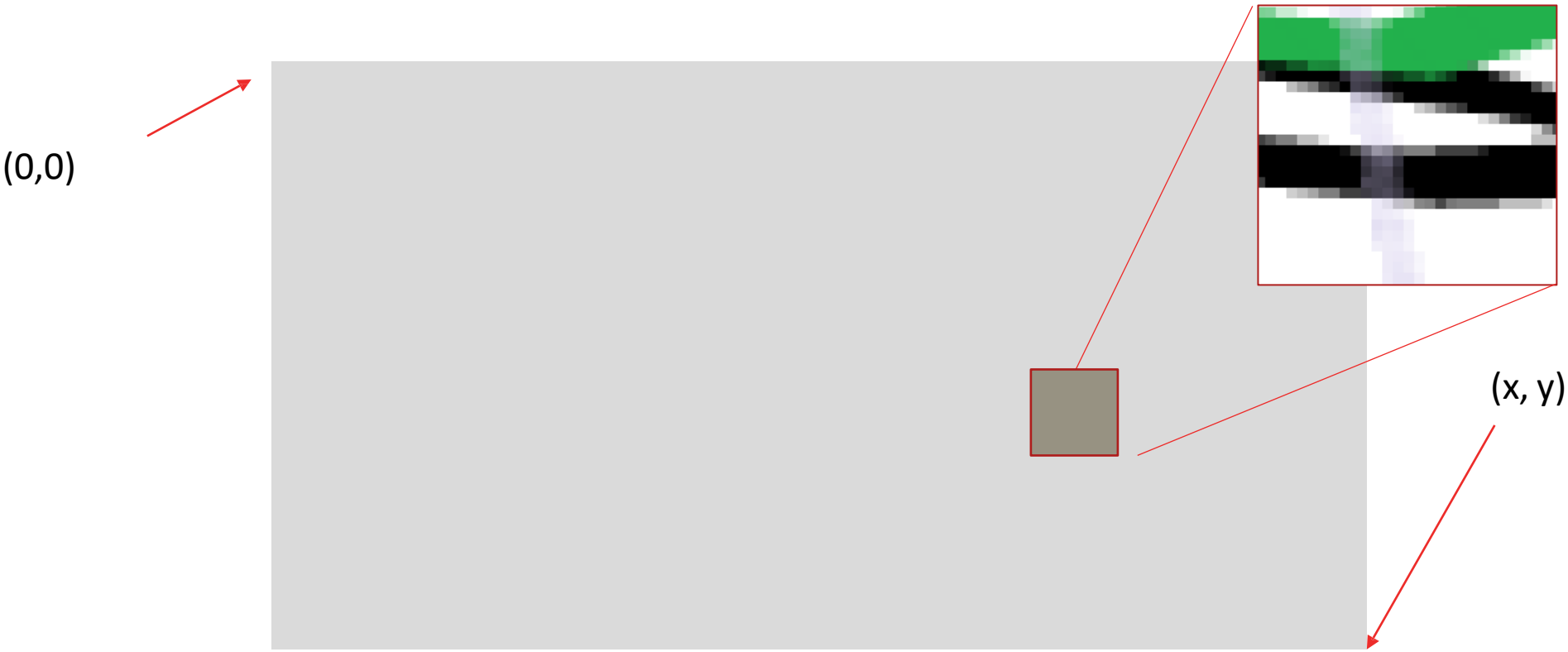
Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

Wednesday, 10 November 2021

UNIVERSITY OF CALGARY

So how do I make things graphical?

- Plot groups of pixels onto a screen



Graphics and Java

- AWT (Abstract Windows Toolkit) (since 1.0, use native OS look)
- Swing (1.2, replace AWT mostly, more options can look like system or Java variant)
- JavaFX (1.8, 1.11 removed as included and now own JDK)
  - Moved out of regular JDK as desktop apps became less of a default
  - Also capable of web style front ends
  - Gluon is a commercial port for Android/iOS use

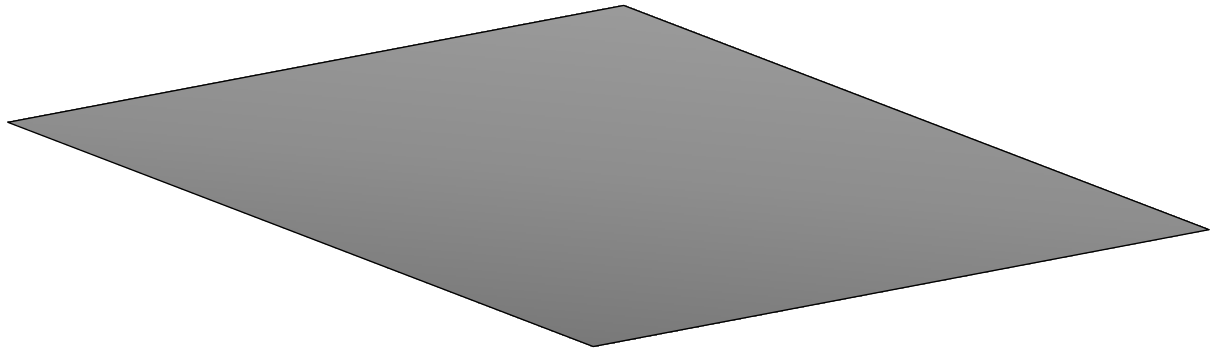
    https://openjfx.io/javadoc/20/
- Numerous other packages exist, but these are the most standard

What are the key classes of JavaFX, and how do they interact?

Anatomy of a JavaFX application (0)

Application (your app extends this class)

- think of it as the part that separates your program from your device
- calls start(javafx.stage.Stage)
- waits for the application to end



Anatomy of a JavaFX application (0)

Application (your app extends this class)

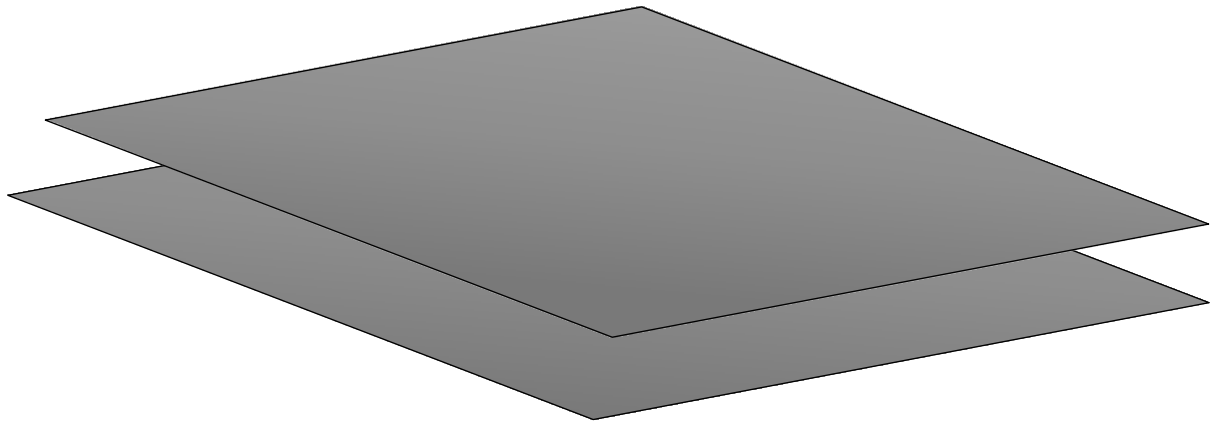
- think of it as the part that separates your program from your device
- calls start(javafx.stage.Stage)
- waits for the application to end

```
import javafx.application.Application;

public class HelloApplication extends
Application {

}
```

Anatomy of a JavaFX application (1)

Stage

- the main window of your app



Anatomy of a JavaFX application (1)

Stage

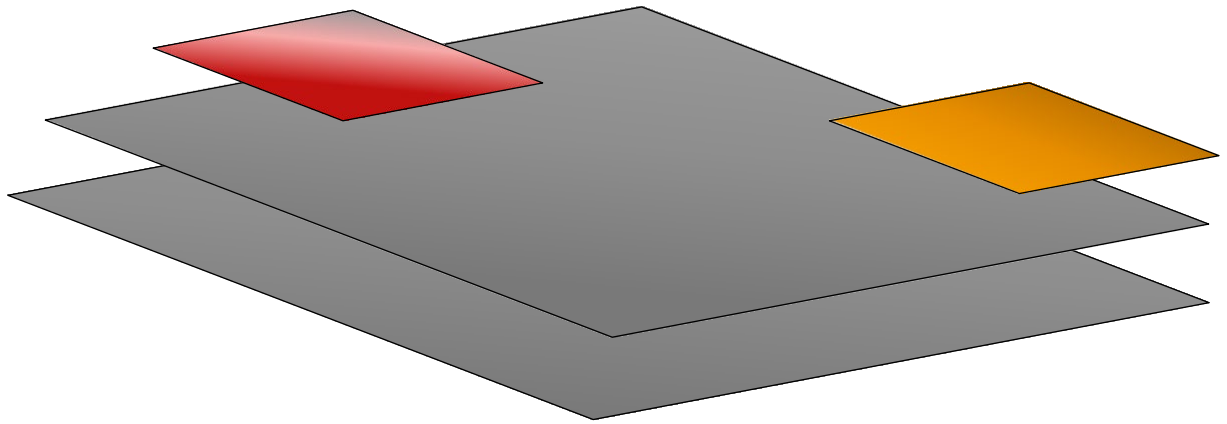
- the main window of your app

```
package com.example.javafxapp;

import javafx.application.Application;
import javafx.stage.Stage;
import java.io.IOException;

public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
    }
}
```

Anatomy of a JavaFX application (2)

Scene

- holds your content (optional to have more than one, but very useful when you have multiple screens or animation)



Anatomy of a JavaFX application (2)

Scene

- holds your content (optional to have more than one, but very useful when you have multiple screens or animation)

```
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("hello-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), WIDTH, HEIGHT);
    }
}
```

Anatomy of a JavaFX application (3)

```
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("hello-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 320, 240);
        stage.setTitle("Hello!");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```

.fxml?

```
<?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.javafxapp.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?

The basic set of GUI controls we can add

- https://openjfx.io/javadoc/20/javafx.controls/javafx/scene/control/package-summary.html
- Things like Buttons, Label from example

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

- or TextFields, MenuItems, ColorPickers, Alert windows

.fxml?

Can also add Panes (help us layout our controls)

- Vbox

Geometry that can help us control where things are

- Insets

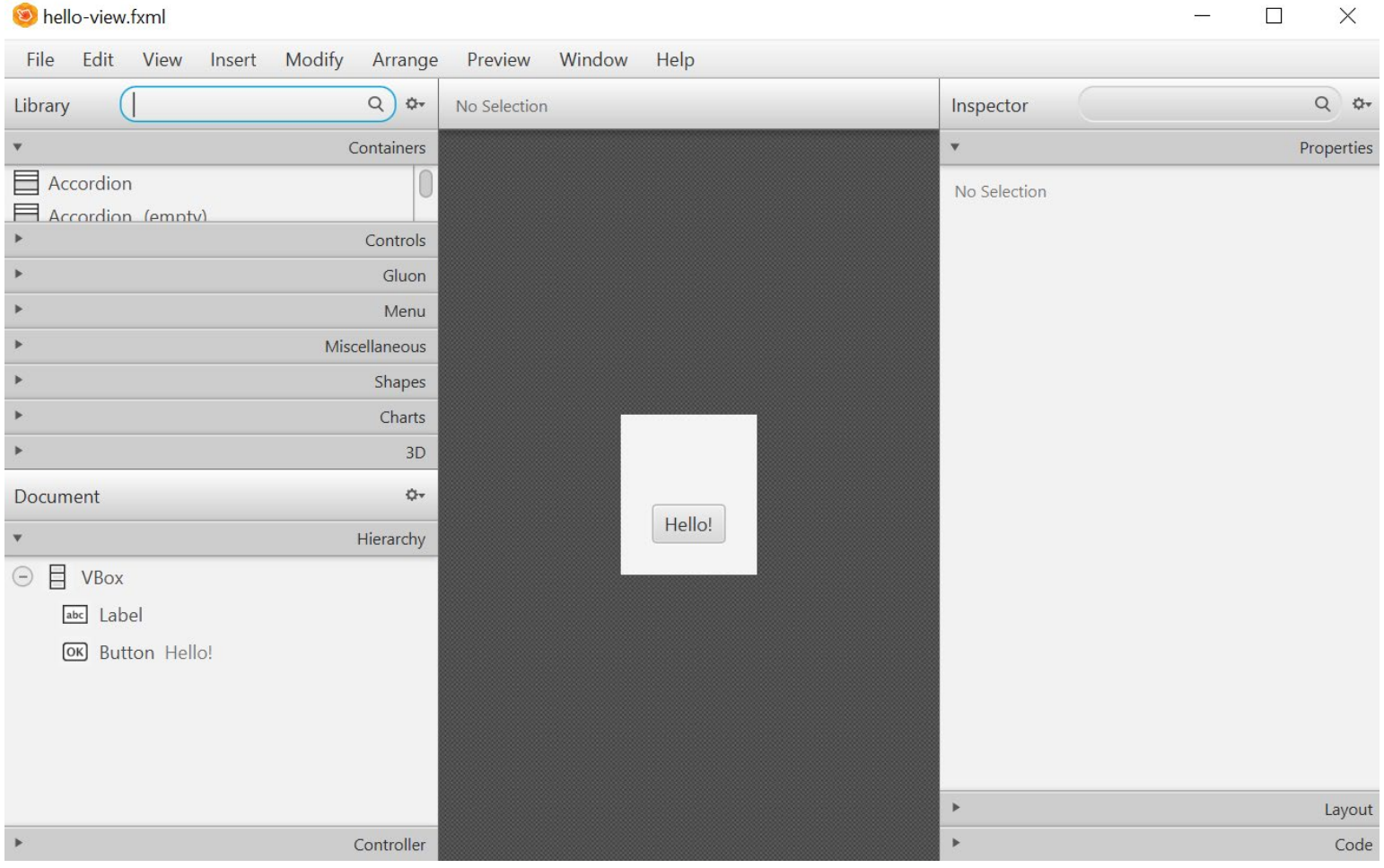
- javafx.scene.layout.**Pane**
  - javafx.scene.layout.**AnchorPane**
  - javafx.scene.layout.**BorderPane**
  - javafx.scene.layout.**FlowPane**
  - javafx.scene.layout.**GridPane**
  - javafx.scene.layout.**HBox**
  - javafx.scene.layout.**StackPane**
  - javafx.scene.layout.**TilePane**
  - javafx.scene.layout.**VBox**

- javafx.geometry.**Bounds**
  - javafx.geometry.**BoundingBox**
- javafx.geometry.**Dimension2D**
- javafx.geometry.**Insets**
- javafx.geometry.**Point2D** (implements javafx.animation.Interpolatable<T>)
- javafx.geometry.**Point3D** (implements javafx.animation.Interpolatable<T>)
- javafx.geometry.**Rectangle2D**

.fxml?

Often designed through SceneBuilder 2.0

- GUI drag and drop tool

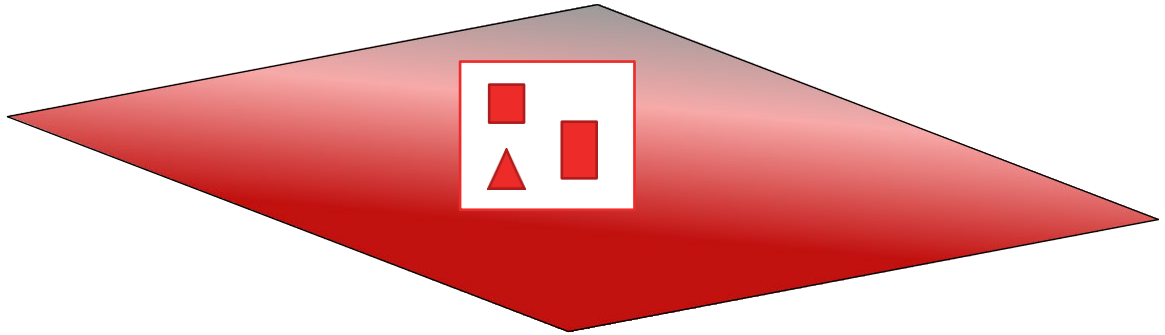


UNIVERSITY OF CALGARY

Anatomy of a JavaFX application (4)

Group

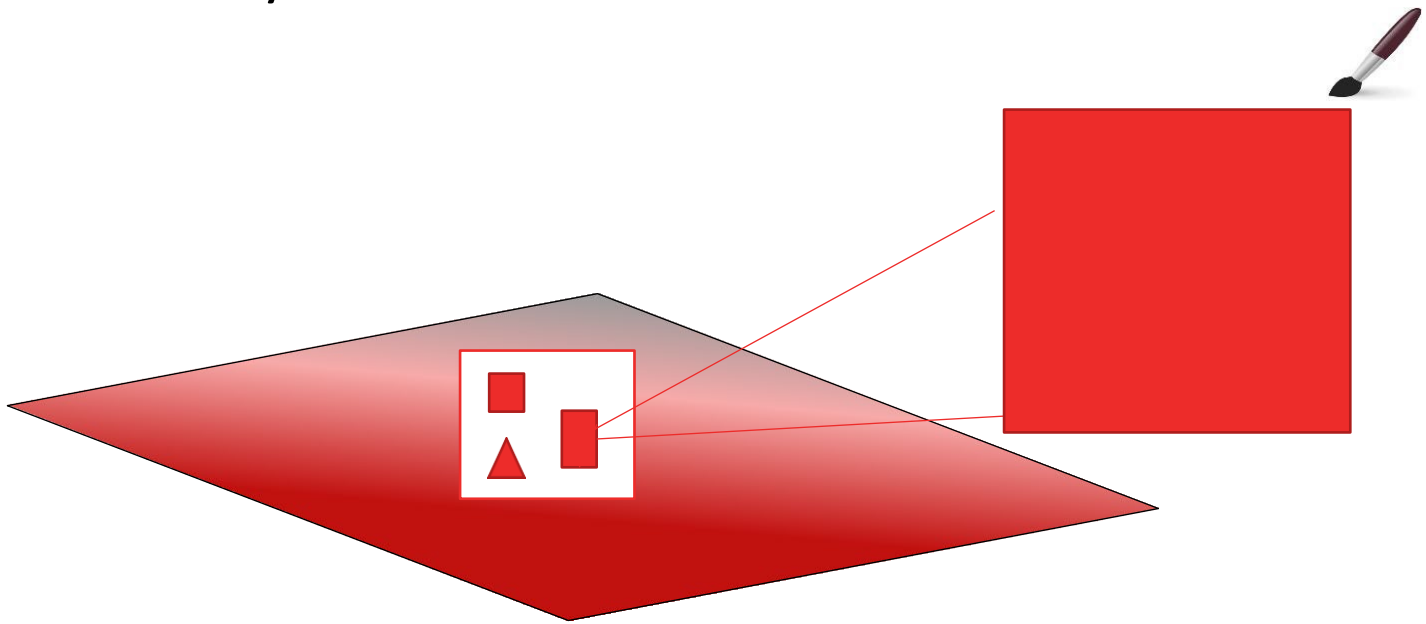
- Generally also declared within .fxml
- It is abstract idea around a bunch of parts that doesn't have a viewable version, but lets you interact with a bunch at once
- Like a layout pane (but without the visual consequence)
- helps to organize your content for simultaneous modification



Anatomy of a JavaFX application (5)

GraphicsContext

- Another component that can be added to application
- Provides a way to draw on a Canvas



Anatomy of a JavaFX application (6)

Node

- Another key class in JavaFX!
- Parent class for most objects that will be added to Scene
- Scene contains a Node called root() that will be very important

We'll create a drawing app in JavaFx?

Setting up your application

Start with design

- Keep it simple to start
  - Easiest GUI will not have to drastically change as actions occur.
- Sketch an image of your GUI
  - You know your first GUI is simple enough if in your sketch you can model each action and consequence without having to draw completely new images
  - More complicate GUI ideas will include different Scenes/popups
- Use SceneBuilder to reproduce your sketch

Setting up your application

Use SceneBuilder to reproduce your sketch

1. Name your important parts with ‘id’ that will involved in actions
2. Go to each control that has important actions that are associated with them and name the functions associated with the actions that occur
3. Save your .fxml
4. Generate your Controller starter code
5. Modify the controller code

We'll make a new empty App Starter (Drawing.java) and a controller for it (DrawingController.java)

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

public class Drawing extends Application {

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

    @Override
    public void start(Stage stage) {

    }
}
```

```java
package com.example.javafxapp;

public class DrawingController {
}
```

We'll make a new empty fxml file (drawing-view.fxml)

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

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane xmlns="http://javafx.com/javafx"
        xmlns:fx="http://javafx.com/fxml"
        fx:controller="com.example.javafxapp.DrawingController"
        prefHeight="600.0" prefWidth="800.0">

</AnchorPane>
```

Let’s make it so we can launch that view

```
@Override
public void start(Stage stage) throws IOException {
    FXMLLoader fxmlLoader = new FXMLLoader(Drawing.class.getResource("drawing-view.fxml"));
    Scene scene = new Scene(fxmlLoader.load(), 800, 600);
    stage.setTitle("Let's Draw!");
    stage.setScene(scene);
    stage.show();
}
```

Add Canvas to fxml and propagate name to Controller

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

<?import javafx.scene.canvas.Canvas?>
<?import javafx.scene.layout.AnchorPane?>

<AnchorPane prefHeight="600.0" prefWidth="800.0"
xmlns="http://javafx.com/javafx/17"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.example.javafxapp.DrawingController">
  <children>
    <Canvas fx:id="canvas" height="600.0" layoutX="-1.0" width="800.0" />
  </children>
</AnchorPane>
```

```
import javafx.fxml.FXML;
import javafx.scene.canvas.Canvas;

public class DrawingController {

   @FXML
   private Canvas canvas;

}
```

Setup controller to pull out the graphics context

```
public class DrawingController {

    GraphicsContext gc;

    @FXML
    private Canvas canvas;

    @FXML
    public void initialize() {
        gc = canvas.getGraphicsContext2D();
    }

}
```

Now that we've placed controls in our interface, how do I use them?

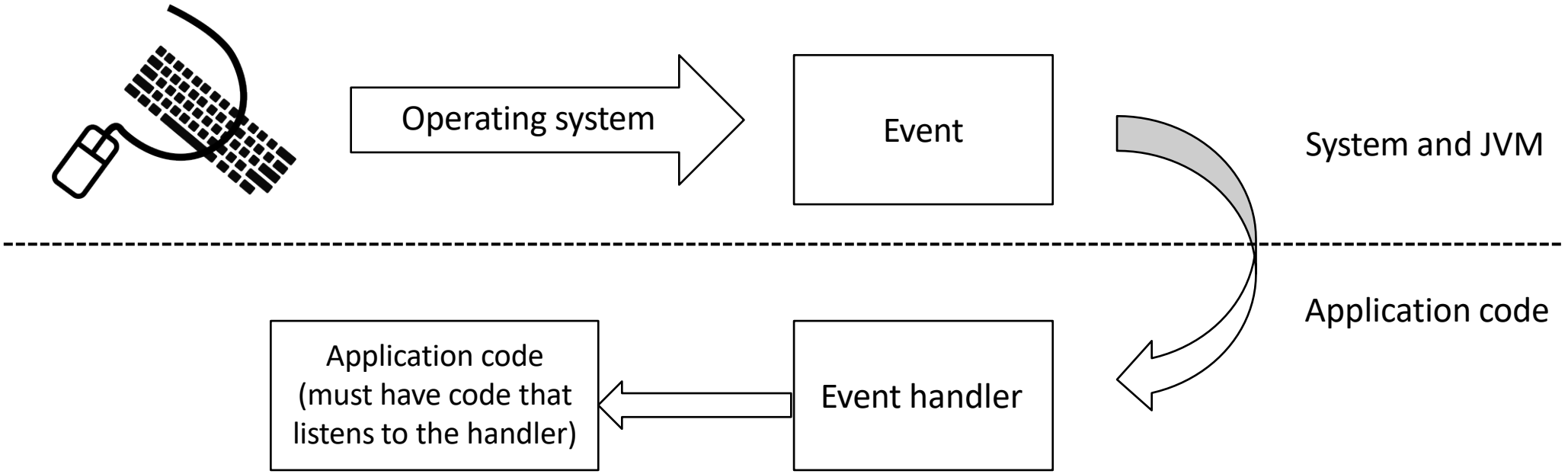
27

Things that trigger events

- Interactions with widgets such as buttons, text boxes, dropdown boxes, etc.
- Timers
- Mouse button presses (or touch screen)
- Typing on the keyboard
- Tilting, swiping on a device
- Receiving messages via the network
- Button press on game controller

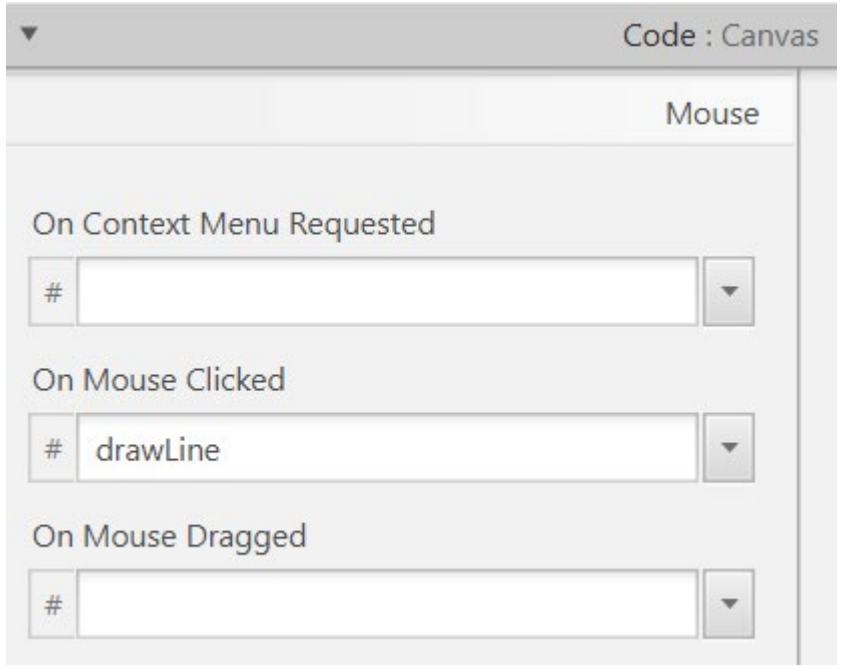
Event handling

- A set of classes known as *event handlers* let us make our interfaces interactive



You'll learn more about the operating system itself and how hardware interfaces with software in 2nd and 3rd year.

Let’s add an event handler from SceneBuilder




```
public class DrawingController {
    GraphicsContext gc;
    @FXML
    private Canvas canvas;
    @FXML
    public void initialize() {
        gc = canvas.getGraphicsContext2D();
    }

    @FXML
    void drawLine(MouseEvent event) {
    }

}
```

```
<Canvas fx:id="canvas" height="600.0" layoutX="-1.0" onMouseClicked="#drawLine" width="800.0" />
```

UNIVERSITY OF CALGARY

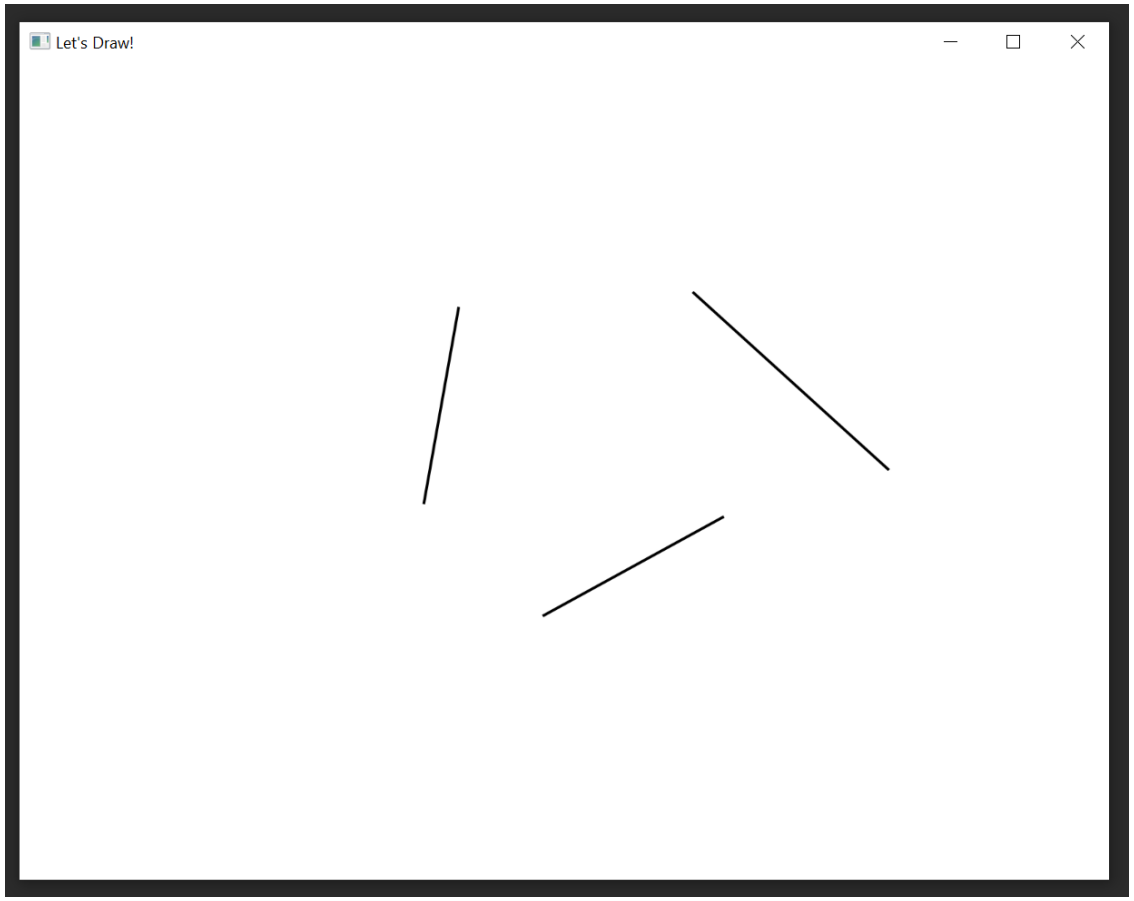
Let’s add an event handler from SceneBuilder

An updated handler for the mouse click
Here we will indicate if this is a first click or a second click
Then we will draw between them

```
boolean line_started = false;
@FXML
void drawLine(MouseEvent event) {
    if(!line_started){
        line_started = true;
        System.out.println("First");
    }else{
        line_started = false;
        System.out.println("Second");
    }
}
```

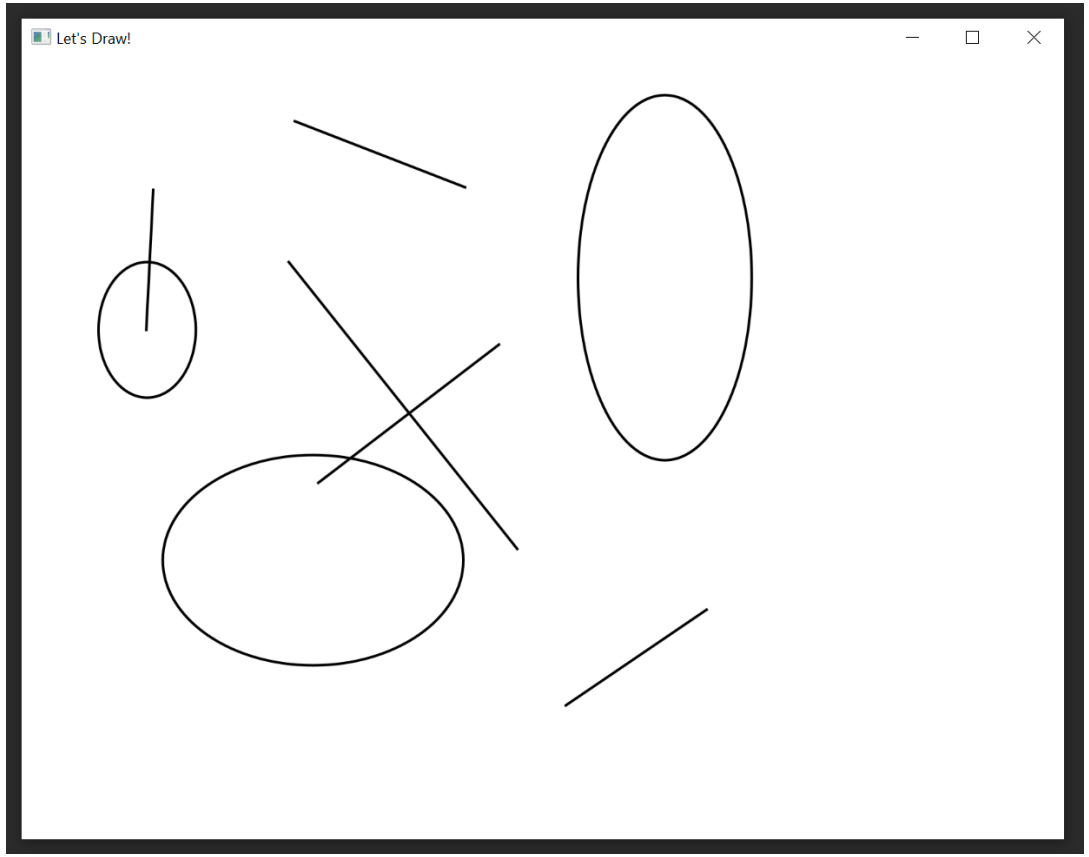
Ramp it up to draw a line

```
boolean line_started = false;
@FXML
void drawLine(MouseEvent event) {
    if(!line_started){
        gc.beginPath();
        gc.moveTo(event.getSceneX(), event.getSceneY());
        line_started = true;
    }else{
        gc.setLineWidth(2);
        gc.setStroke(Color.BLACK);
        gc.lineTo(event.getSceneX(), event.getSceneY());
        gc.stroke();
        line_started = false;
    }
}
```



What about drawing an ellipse

```
boolean line_started = false;
boolean ellipse_started = false;
double arc_x, arc_y;
@FXML
void drawLine(MouseEvent event) {
    if(event.getButton() == MouseButton.PRIMARY) {
        if (!line_started) {
            gc.beginPath();
            gc.moveTo(event.getSceneX(), event.getSceneY());
            line_started = true;
            ellipse_started = false;
        } else {
            gc.setLineWidth(2);
            gc.setStroke(Color.BLACK);
            gc.lineTo(event.getSceneX(), event.getSceneY());
            gc.stroke();
            line_started = false;
            ellipse_started = false;
        }
```



What about drawing an ellipse

```
boolean line_started = false;
boolean ellipse_started = false;
double arc_x, arc_y;
@FXML
void drawLine(Mou
    if(event.getButto
        if (!line_started
            gc.beginPath
            gc.moveTo(e
            line_started =
            ellipse_starte
        } else {
            gc.setLineWi
            gc.setStroke(
            gc.lineTo(eve
            gc.stroke();
            line_started =
            ellipse_starte
        }
```

```
}else if(event.getButton() == MouseButton.SECONDARY){
        if (!ellipse_started) {
            gc.beginPath();
            arc_x = event.getSceneX();
            arc_y = event.getSceneY();
            ellipse_started = true;
            line_started = false;
        } else {
            gc.setLineWidth(2);
            gc.setStroke(Color.BLACK);
            gc.arc(arc_x, arc_y,Math.abs(event.getSceneX()-arc_x), Math.abs(event.getSceneY()-arc_y),0,360);
            gc.stroke();
            ellipse_started = false;
            line_started = false;
        }
    }
}
```

Event Handlers

- You can write a custom event handler as its own class

```
@FXML
public void initialize() {
    gc = canvas.getGraphicsContext2D();
    gc.setFill(Color.WHITE);
    canvas.setOnMouseClicked(new MyMouseClickedHandler(gc));
}
```

Event Handlers (in MyMouseClickHandler.java)

- You can write a custom event

```
@FXML
public void initialize() {
    gc = canvas.getGraphicsContext2D();
    gc.setFill(Color.WHITE);
    canvas.setOnMouseClicked(new MyM
}
```

```
package com.example.javafxapp;
import …;
public class MyMouseClickedHandler implements EventHandler<MouseEvent> {

    GraphicsContext gc;

    public MyMouseClickedHandler(GraphicsContext gc) {
        this.gc = gc;
    }

    boolean line_started = false;
    boolean ellipse_started = false;
    double arc_x, arc_y;

    public void handle(MouseEvent event){
        if(event.getButton() == MouseButton.PRIMARY) {
            if (!line_started) {
                gc.beginPath();
                gc.moveTo(event.getSceneX(), event.getSceneY());
                //…
```

Event Handlers

- As internal class

```
@FXML
public void initialize() {
    gc = canvas.getGraphicsContext2D();
    gc.setFill(Color.WHITE);
    canvas.setOnMouseClicked(new EventHandler<MouseEvent>() {
        boolean line_started = false;
        boolean ellipse_started = false;
        double arc_x, arc_y;
        @Override
        public void handle(MouseEvent event) {
            if(event.getButton() == MouseButton.PRIMARY) {
                if (!line_started) {
                    gc.beginPath();
                    gc.moveTo(event.getSceneX(), event.getSceneY());
                    //...
```

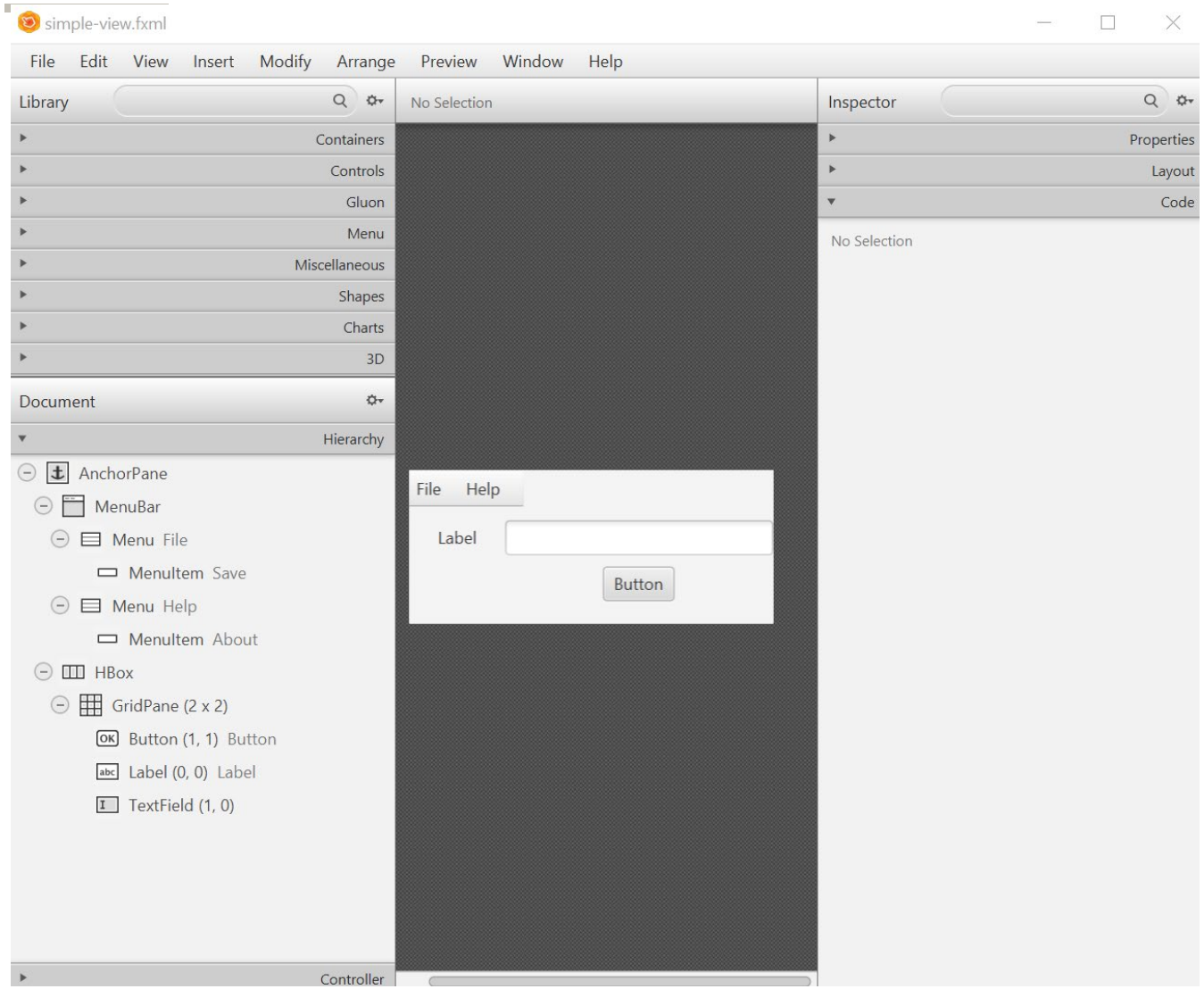
Don’t forget to listen!

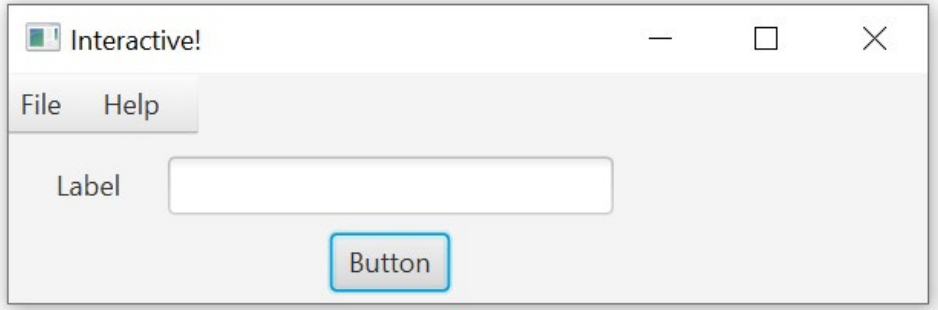
- Your application must have something which listens for events
  - method call is commonly on the form “setOn<noun><verb>”
    - nouns: Action, Mouse, Keyboard
    - verbs: Clicked, Pressed, etc
  - There are many different kinds of actions – you may have to experiment to get the one you want

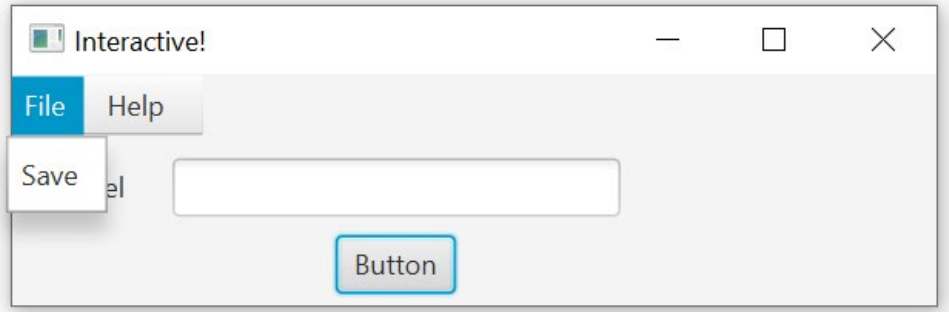
Let’s work with a button, label, menu, and alert!

39

Let’s make a simple action app







UNIVERSITY OF CALGARY

Adjust Label when Button is clicked to the text in TextField

```
public class SimpleViewController {

    @FXML
    private Label mylabel;

    @FXML
    private TextField mytextfield;

    @FXML
    void buttonClicked(ActionEvent event) {
        mylabel.setText(mytextfield.getText());
    }

}
```

Alert Menu! Save File Menu!

```
@FXML
    void menuAboutAction(ActionEvent event) {
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION, "Did you see this coming?");
        alert.show();
    }
    @FXML
    void saveAction(ActionEvent event) {
        final FileChooser fileChooser = new FileChooser();
        fileChooser.setInitialDirectory(new File("."));
        fileChooser.setInitialFileName("world.txt");
        File fileSave = fileChooser.showSaveDialog(new Stage());
        System.out.println(fileSave);
    }
```

Onward to … OO for Data Structures

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY