



Prifysgol
Abertawe
Swansea
University

CS-230 Software Engineering

L12: JavaFX

Dr. Liam O'Reilly

Semester 1 – 2020

History of GUI in Java

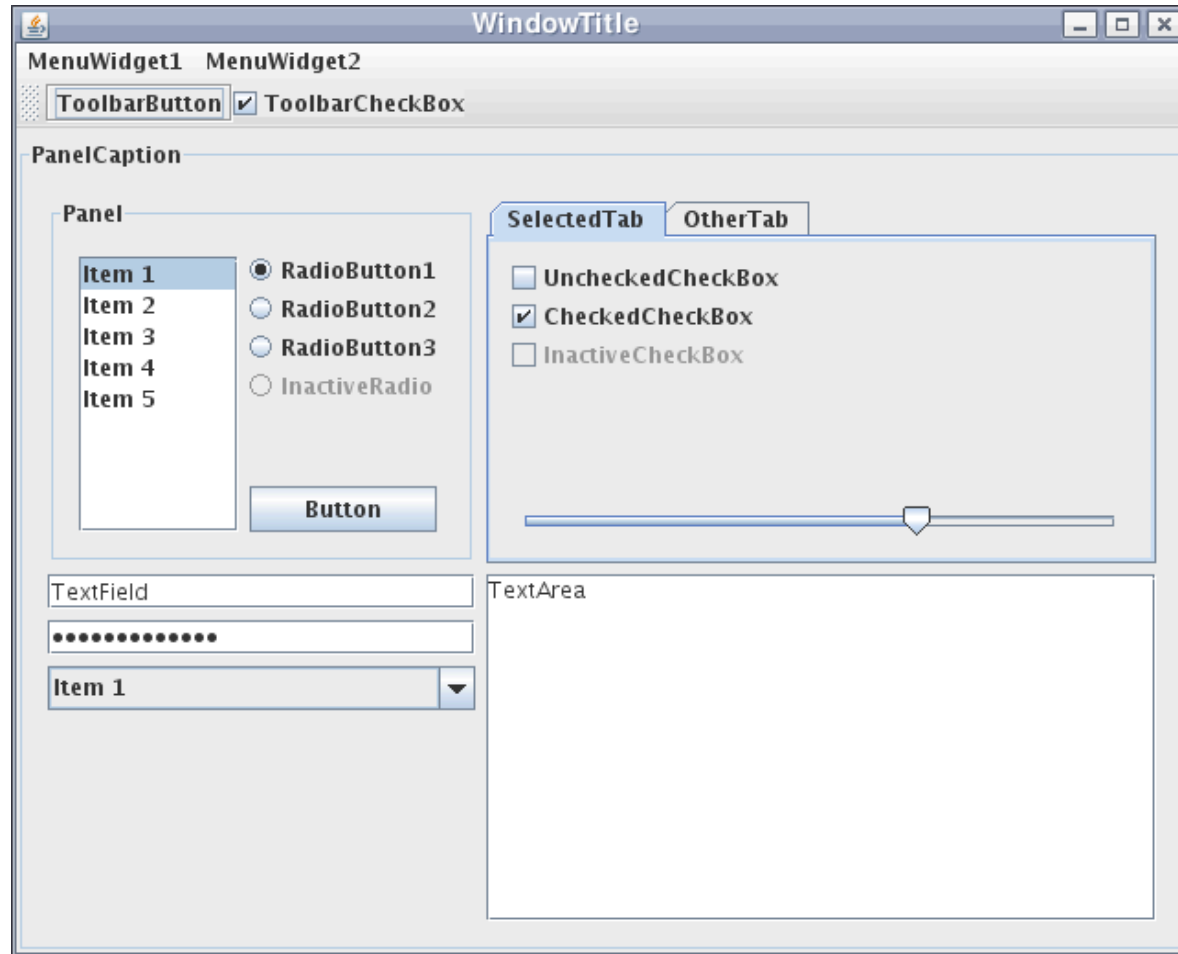
- Java's original GUI library was the Abstract Window Toolkit (AWT).
- Swing was added in Java SE 1.2.
- Swing has been primary Java GUI technology till now.
- Swing is now in maintenance mode—Oracle has stopped development and will provide only bug fixes.
- JavaFX is the new kid on the block.
- There are other GUI frameworks for Java but the above are the standard “built-in” ones.

AWT Example



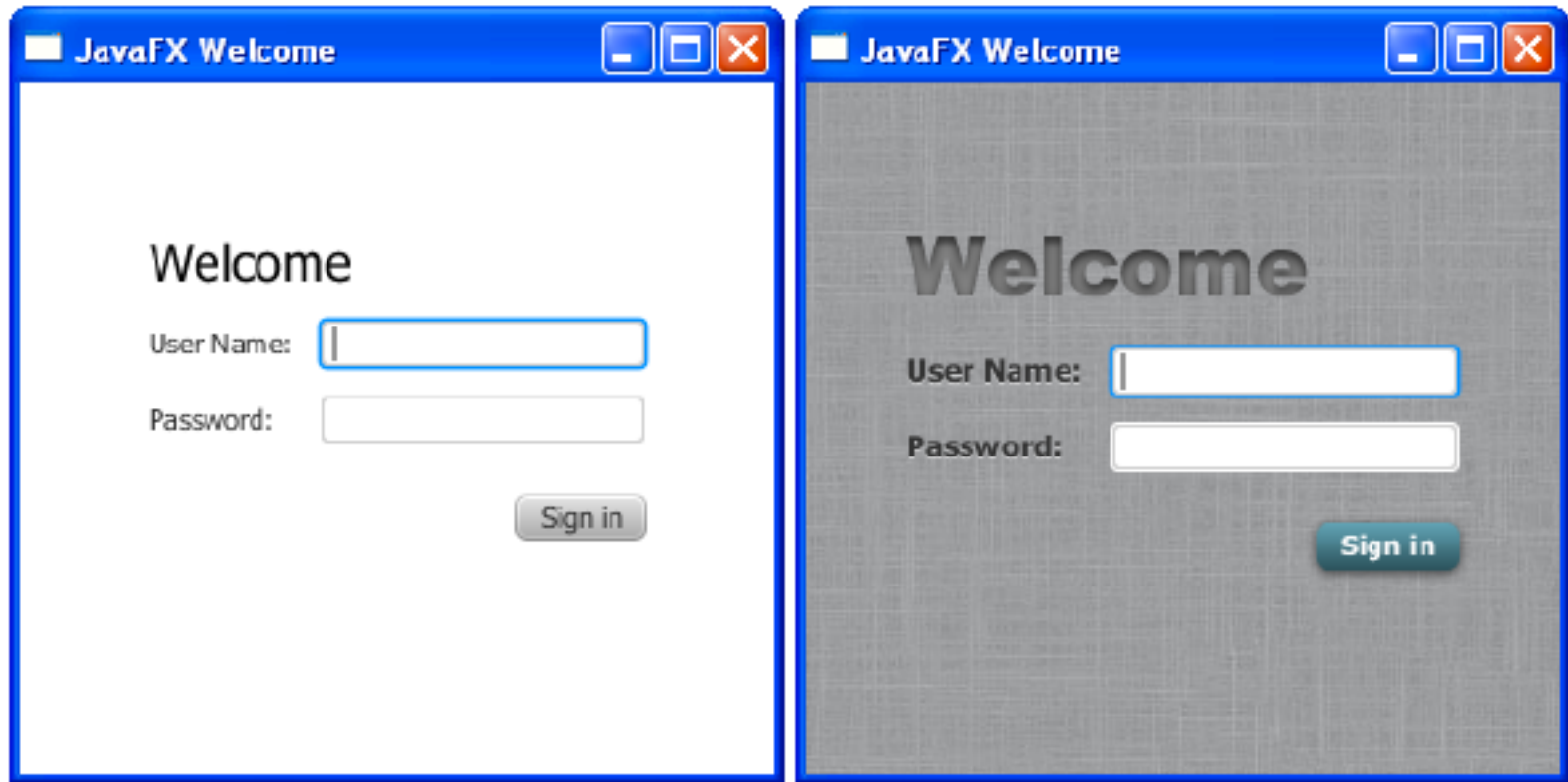
AWT just looks old. It is old.

Swing Example



Swing looks better, but is now a bit old. It was always hard to get Swing to look and behave like windows on the native OS.

JavaFX Example

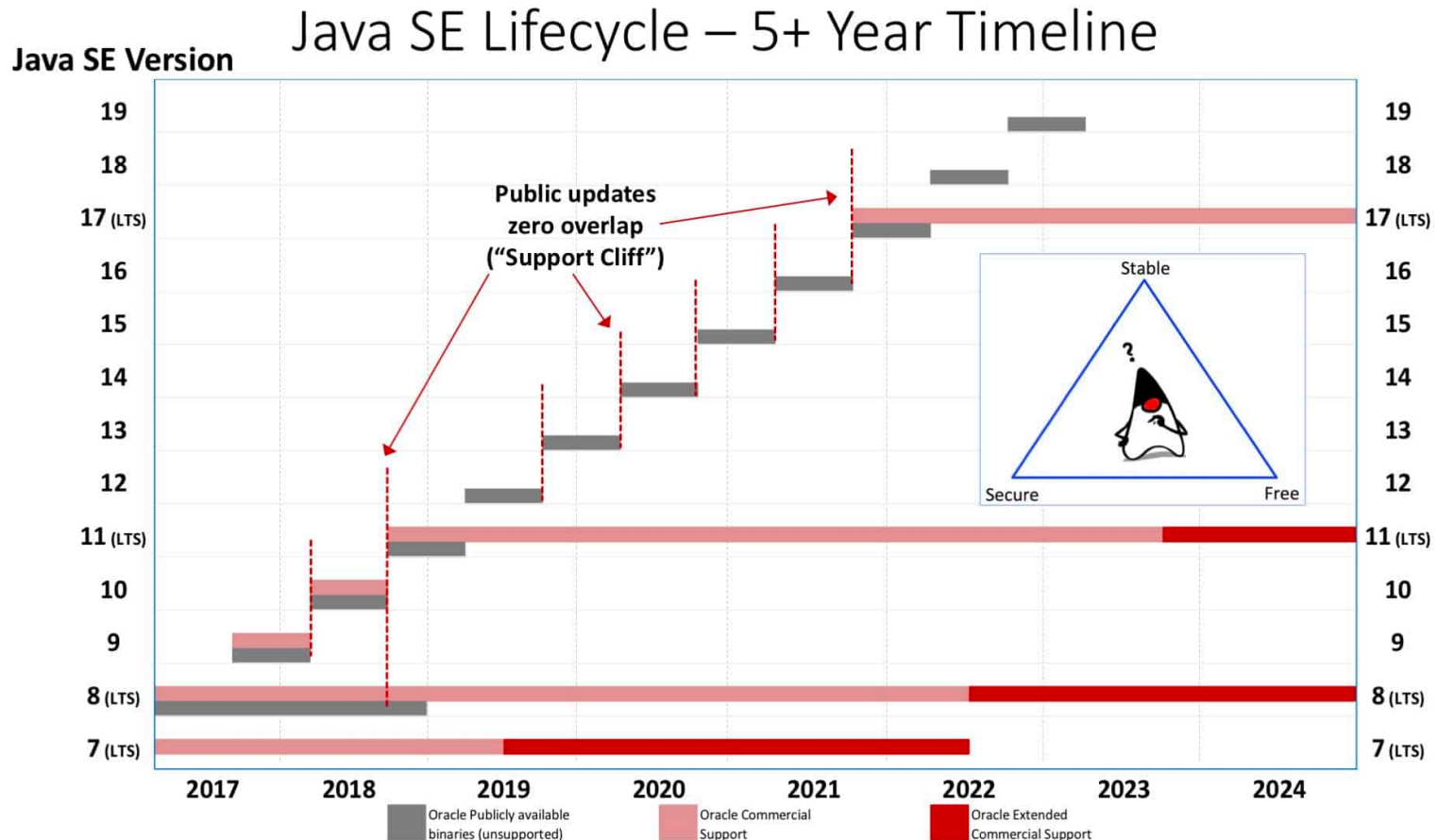


JavaFX looks fancy. It can even be styled using CSS.

Java 8 to Java 11, Modules, and JavaFX

Java is going through a **massive** change.

Both Java 8 and Java 11 are currently supported.

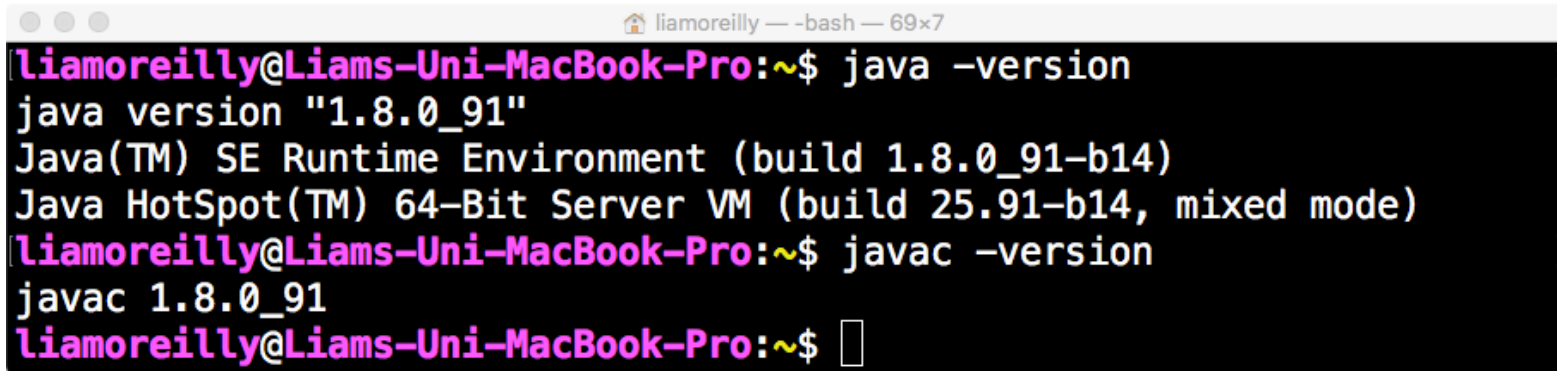


Java 8 to Java 11, Modules, and JavaFX (Cont.)

- Java 11 introduces the concept of modules:
 - This make working with Java currently tricky
 - Many tools are not caught up yet.
- Java 8 comes with JavaFX bundled in. Install Java 8 and you are done – JavaFX is part of it.
- Java 11 has removed JavaFX. You download JavaFX as a separate module and have to link it in to Java (via manual configuration).
- **We, the department, are sticking with Java 8 for now. I strongly suggest you do too. All my slides assume Java 8.**
 - If you want to use Java 11 (or newer) and JavaFX 11 (or newer) then see the install instructions on <https://openjfx.io/>

What Java Version Are You Running?

- Find out what version of Java you are running by running the following commands:

A screenshot of a macOS terminal window. The title bar at the top shows a home icon, the username 'liamoreilly', and the shell '-bash' with a window size of '69x7'. The terminal text shows the user 'liamoreilly' at the prompt 'liamoreilly@Liams-Uni-MacBook-Pro:~\$' running 'java -version'. The output is: 'java version "1.8.0_91"', 'Java(TM) SE Runtime Environment (build 1.8.0_91-b14)', and 'Java HotSpot(TM) 64-Bit Server VM (build 25.91-b14, mixed mode)'. Then, the user runs 'javac -version' and the output is 'javac 1.8.0_91'. The prompt returns to 'liamoreilly@Liams-Uni-MacBook-Pro:~\$' with a cursor.

```
liamoreilly@Liams-Uni-MacBook-Pro:~$ java -version
java version "1.8.0_91"
Java(TM) SE Runtime Environment (build 1.8.0_91-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.91-b14, mixed mode)
liamoreilly@Liams-Uni-MacBook-Pro:~$ javac -version
javac 1.8.0_91
liamoreilly@Liams-Uni-MacBook-Pro:~$
```

- Note: Java and Javac should be the same version!
- Note: Java 8 was technically known as Java 1.8 – hence the above version number.

Self-Learning

I am only going to get you started, plus a bit more, with JavaFX.

You should be now capable of picking up libraries like this yourself.

- This is an important skill to master.

Oracle “Home Page” for JavaFX:

- <https://docs.oracle.com/javase/8/javase-clienttechnologies.htm>

You will need to lookup the JavaDoc pages for many of the JavaFX classes to find out how they work.

Stages, Scenes and Panes

- JavaFX is all built around Stages.
- Imagine a stage in a theatre.
 - There is one stage.
- A stage is a window.
 - You could have multiple windows if you like – but that's getting complex.
- At any one time there can be a single scene on the stage.
- We can change from one scene to another scene.
- Within a scene we can place objects and lay them out.



The Structure of a JavaFX Application

Loads of JavaFX Imports
– You really need to use
an IDE to help you.

You must extend
application and
override the start
method. You will be
passed a Stage object.

Build up your Scene.
Set it to be on stage.
And show the stage.

(You should use
constants).

Main method just kicks off the JavaFX system.

```
*** JavaFX Imports ***

public class Main extends Application {
    public void start(Stage primaryStage) {
        // Create a new pane to hold our GUI
        Pane root = ...

        // Build a GUI

        // Create a scene based on the pane.
        Scene scene = new Scene(root, 400, 400);

        // Show the scene
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Example: Hello World

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

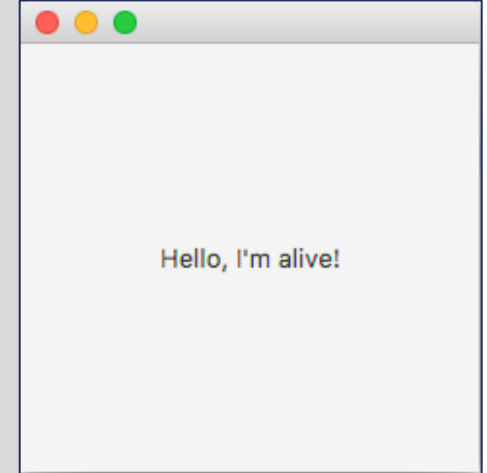
public class Main extends Application {
    public void start(Stage primaryStage) {
        // Create a new pane to hold our GUI
        FlowPane root = new FlowPane();
        root.setAlignment(javafx.geometry.Pos.CENTER);

        // Create a new label
        Label l = new Label("Hello, I'm alive!");
        root.getChildren().add(l);

        // Create a scene based on the pane.
        Scene scene = new Scene(root, 400, 400);

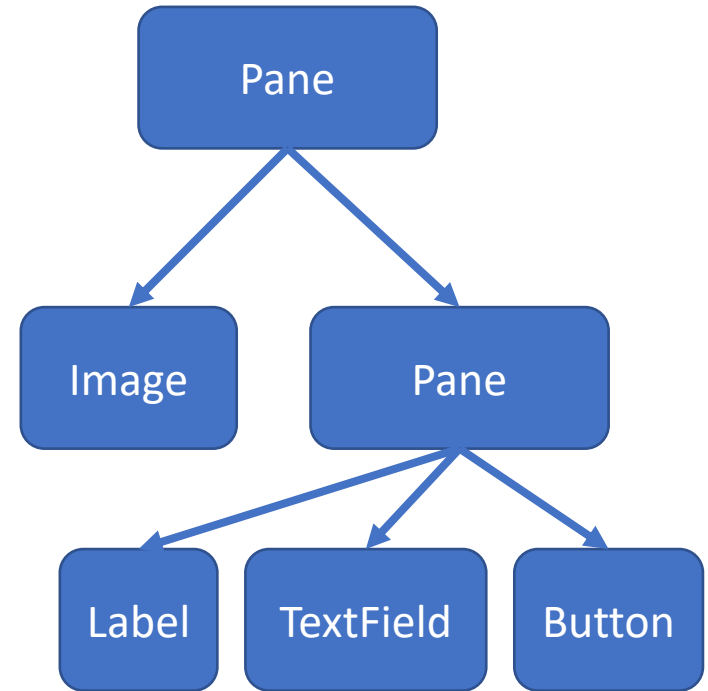
        // Show the scene
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



Scene Graphs

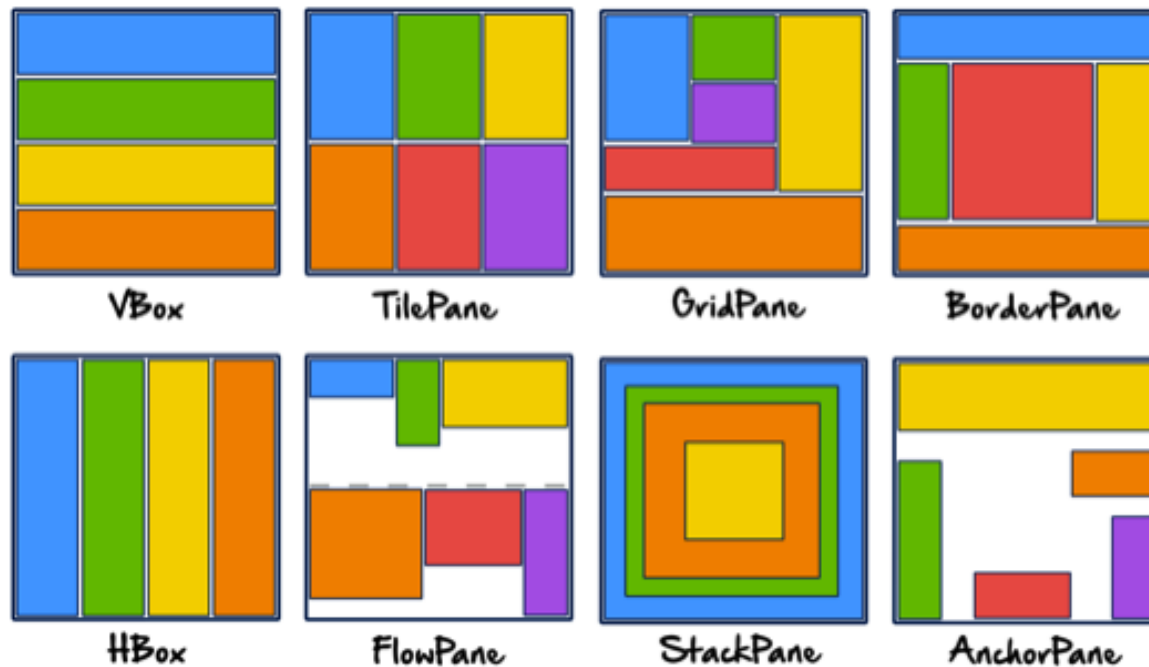
- JavaFX uses Scene Graphs.
- Each node is a visual/graphical object.
E.g.,
 - **Geometrical (Graphical) objects** – (2D and 3D) such as circle, rectangle, polygon, etc.
 - **UI controls** – such as Button, Checkbox, Choice box, Text Area, etc.
 - **Containers** – (layout panes) such as Border Pane, Grid Pane, Flow Pane, etc.
 - These can contain more nodes as children.
 - **Media elements** – such as audio, video and image objects.



- The tree is drawn to the screen using an in order traversal.
- Transformations to parent nodes affect child node.

Layout

- We will cover layout later on.
- Main Idea:
 - There are various panes that layout the children in different ways.
 - You must nest the panes in order to produce desirable layouts.



Events & Event Handling

- In a modern graphical user interface program, the user controls the program through the mouse and keyboard.
- The user can enter information into text fields, pull down menus, click buttons, and drag scroll bars in any order.
 - The program must react to the user commands.
 - The program can choose to receive and handle events such as “mouse move” or a button push “action event”.

Events & Event Handling (2)

- Programs must indicate which events it wants to receive.
- It does so by installing event listener objects.
 - An event listener object belongs to a class that you declare.
 - The methods of your event listener classes contain the instructions that you want to have executed when the events occur.
- To install a listener, you need to know the event source.
- You add an event listener object to selected event sources:
 - Examples: OK Button clicked, Cancel Button clicked, Menu Choice..
- Whenever the event occurs, the event source calls the appropriate methods of all attached event listeners

EventHandler

```
interface EventHandler<T extends Event> {  
    void handle(T event);  
}
```

- To handle events you need to create a class which implements EventHandler.
- It would be very awkward to create a new class for each individual event in your code.
 - The new class would need to be in its own file.
 - Implement EventHandler interface
 - Place the code for the event in the handle method.
 - Once that is done we can use the class:

```
Button button = new Button("Click Me!");  
EventHandler<ActionEvent> eh = new MyEventHandlerClass();  
button.setOnAction(eh);
```

- Solution: anonymous inner classes.

Example: Events

```
public void start(Stage primaryStage) {  
    // Create a new pane to hold our GUI  
    FlowPane root = new FlowPane();  
    root.setAlignment(javafx.geometry.Pos.CENTER);  
  
    // Create a few GUI elements  
    Button button = new Button("Click Me!");  
    Label outputLabel = new Label("I am a label");  
    root.getChildren().addAll(button, outputLabel);  
  
    // Handle a button event  
    button.setOnAction(new EventHandler<ActionEvent>() {  
        public void handle(ActionEvent event) {  
            outputLabel.setText("I was Clicked");  
        }  
    });  
  
    // Create a scene based on the pane.  
    Scene scene = new Scene(root, 400, 400);  
  
    // Show the scene  
    primaryStage.setScene(scene);  
    primaryStage.show();  
}
```



This is whole piece of code is the argument to the method `setOnAction`. See next slide.

Anonymous Inner Classes

```
button.setOnAction(new EventHandler<ActionEvent>() {  
    public void handle(ActionEvent event) {  
        *** Code to Execute on event goes here ***  
    }  
});
```

This strange syntax does a few things:

- First, it creates a class with no name (i.e., an anonymous inner class) that implements the `EventListener` interface.
 - This class provides an implementation for the `handle` method.
- Second, it creates an instance of this anonymous inner class and passes the instance to the `setOnAction` method.

The result is that when the button is clicked, the `handle` method is called.

This was the best way to do this in swing. But with JavaFX we can do even better!

Lambda Expressions

```
button.setOnAction(new EventHandler<ActionEvent>() {  
    public void handle(ActionEvent event) {  
        *** Code to Execute on event goes here ***  
    }  
});
```

Java 8 introduced some new syntax: **lambda expressions**.

The above can be shortened to

```
button.setOnAction(event -> {  
    *** Code to Execute on event goes here ***  
});
```

This only works for **functional interfaces** (interfaces with a single method). It also only works if the types match.

Basically, **functional programming** is starting to merge with **Object Oriented Programming** in Java and other languages.

Example: Calculator

```
public void start(Stage primaryStage) {
    FlowPane root = new FlowPane();
    root.setAlignment(javafx.geometry.Pos.CENTER);

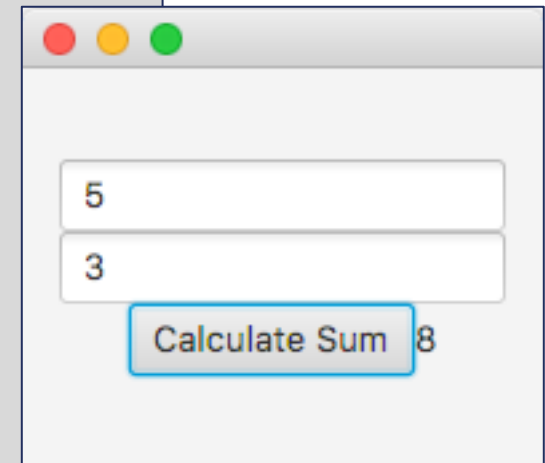
    // Create a GUI elements for the calculator
    TextField box1 = new TextField();
    TextField box2 = new TextField();
    Label result = new Label("No Result");
    Button calButton = new Button("Calculate Sum");

    // Add to root pane
    root.getChildren().addAll(box1, box2, calButton, result);

    // Handle a button event
    calButton.setOnAction(e -> {
        int x = Integer.parseInt(box1.getText());
        int y = Integer.parseInt(box2.getText());
        int z = x + y;
        result.setText(Integer.toString(z));
    });

    // Create a scene based on the pane.
    Scene scene = new Scene(root, 400, 400);

    // Show the scene
    primaryStage.setScene(scene);
    primaryStage.show();
}
```



Summary

- We have very briefly looked at some basics of JavaFX.
- You should be able to pick this up yourself.
 - We will look at a bit more JavaFX next time.
- Use the Internet to find tutorials or read the official Oracle documentation. They have some good introductory tutorials.
 - <https://docs.oracle.com/javase/8/javase-clienttechnologies.htm>