University of Nottingham
UK | CHINA | MALAYSIA

COMP2059
Developing Maintainable Software

LECTURE 06 – MAINTAINABLE GUI DEVELOPMENT (1/2)

Boon Giin Lee (Bryan)

University of Nottingham
UK | CHINA | MALAYSIA

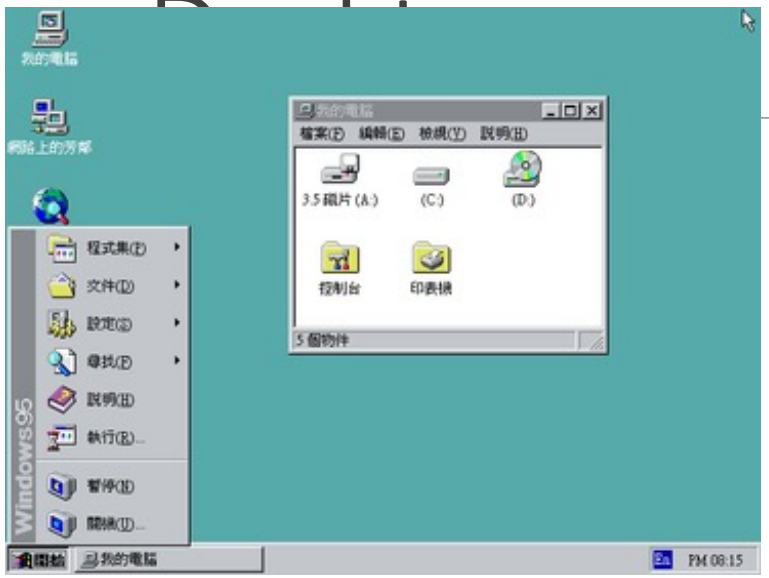
Graphical User Interfaces

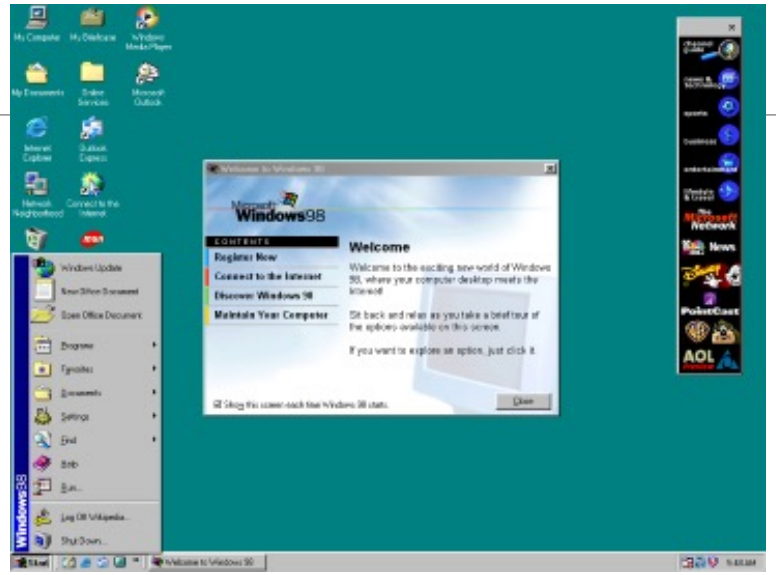
GUI

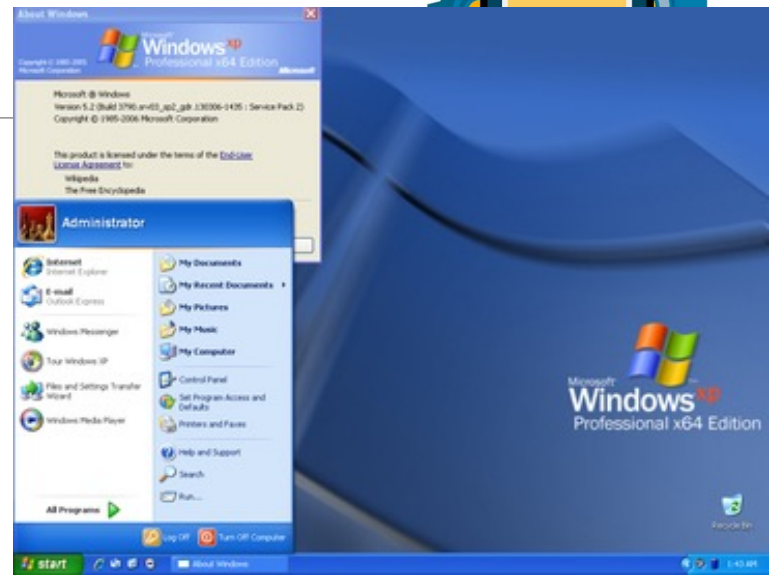
Why Talk About GUIs?

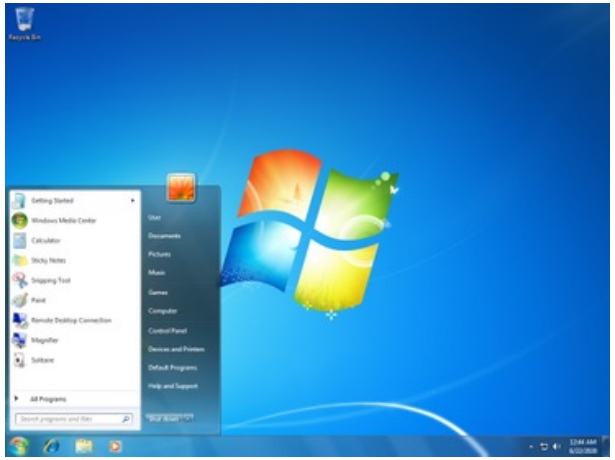
- What is it?
  - You know theses as **buttons**, **windows**, **dialog boxes** etc.
  - Often OS specific.
  - Important, not just for appearance, but usability too.

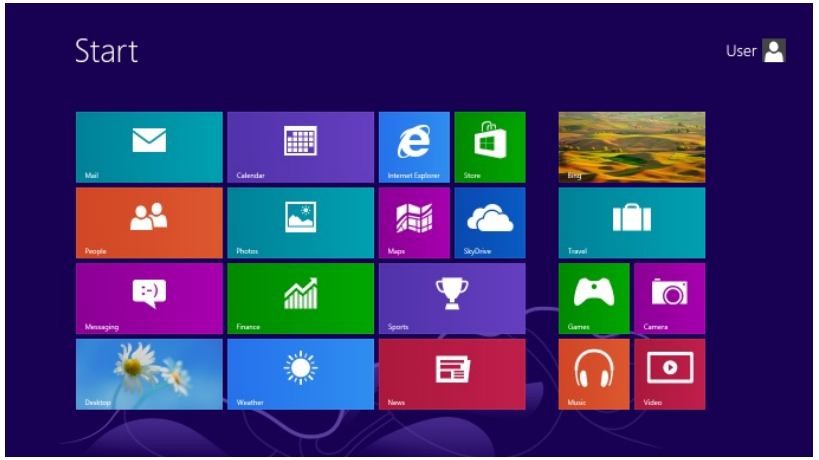
- Why do we need to learn this in DMS?
  - You may want to add new/different UI to existing code, OR
  - Understand the link between an UI and the code behind it in order to maintain it.
  - If write one from scratch, would want to do it in a maintainable way.


Windows95

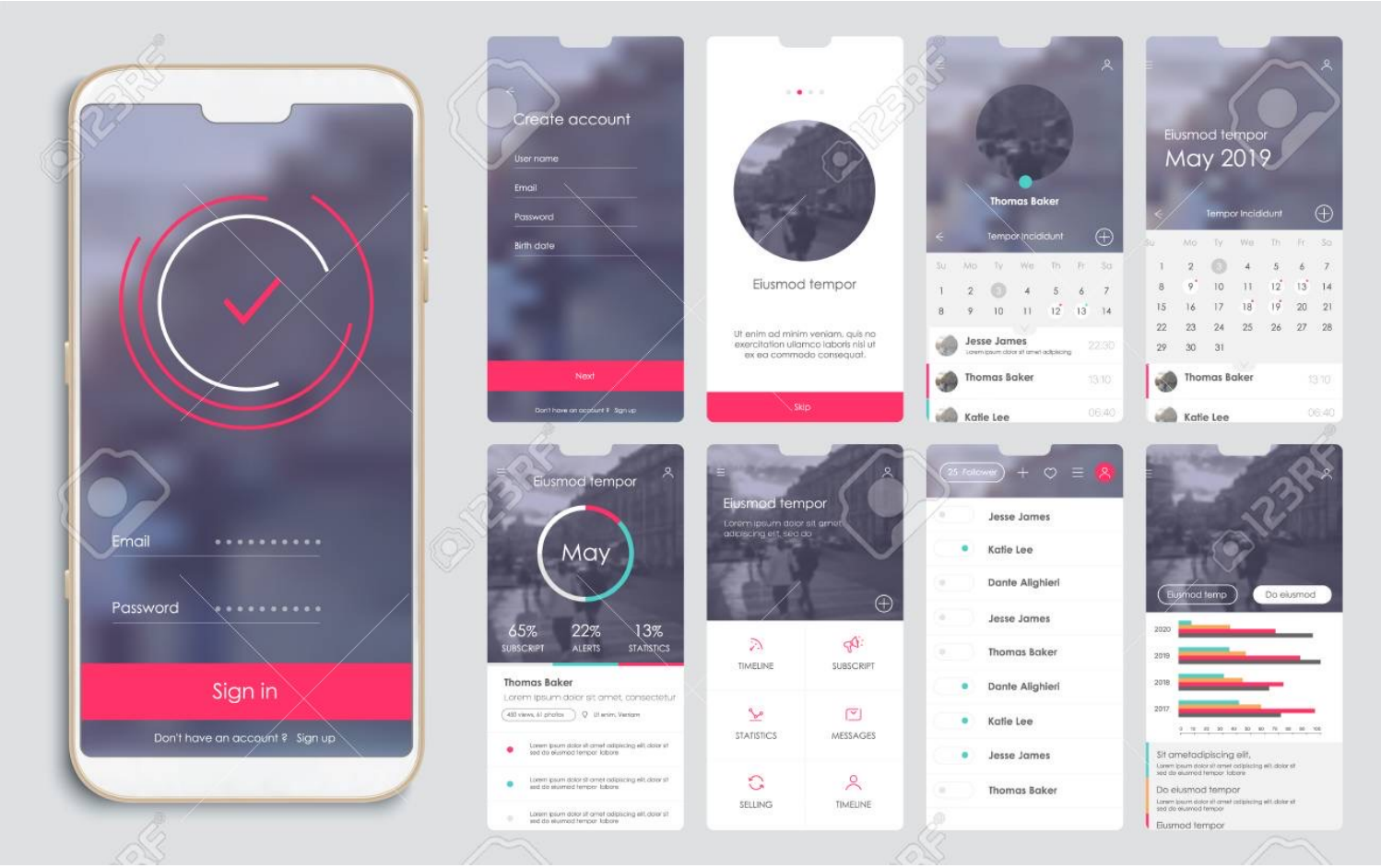

Windows98
Welcome


Windows XP
Professional x64 Edition
Administrator




Start
User

Mobile Application



Future?!?!

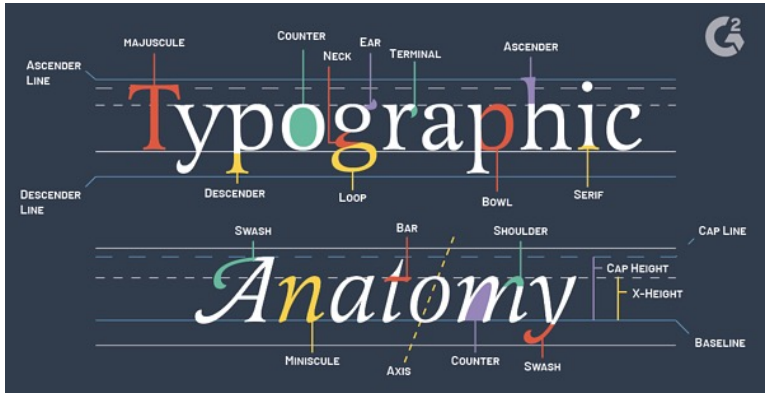







Design

- Being graphical and having the screen estate to work with, the designer has a lot of opportunity to customise the GUI (and to mess it up :D).
- Designing a good GUI is a challenging and specialist job.
  - This is a job for HCI specialist.
- Best practices for designing an interface [https://www.usability.gov/what-and-why/user-interface-design.html] .
  - Keep the interface simple.
  - Create consistency and use common UI elements.
  - Be purposeful in page layout.
  - Strategically use colour and texture.
  - Use typography to create hierarchy and clarity.
  - Make sure that the system communicates what's happening.
  - Think about the defaults.

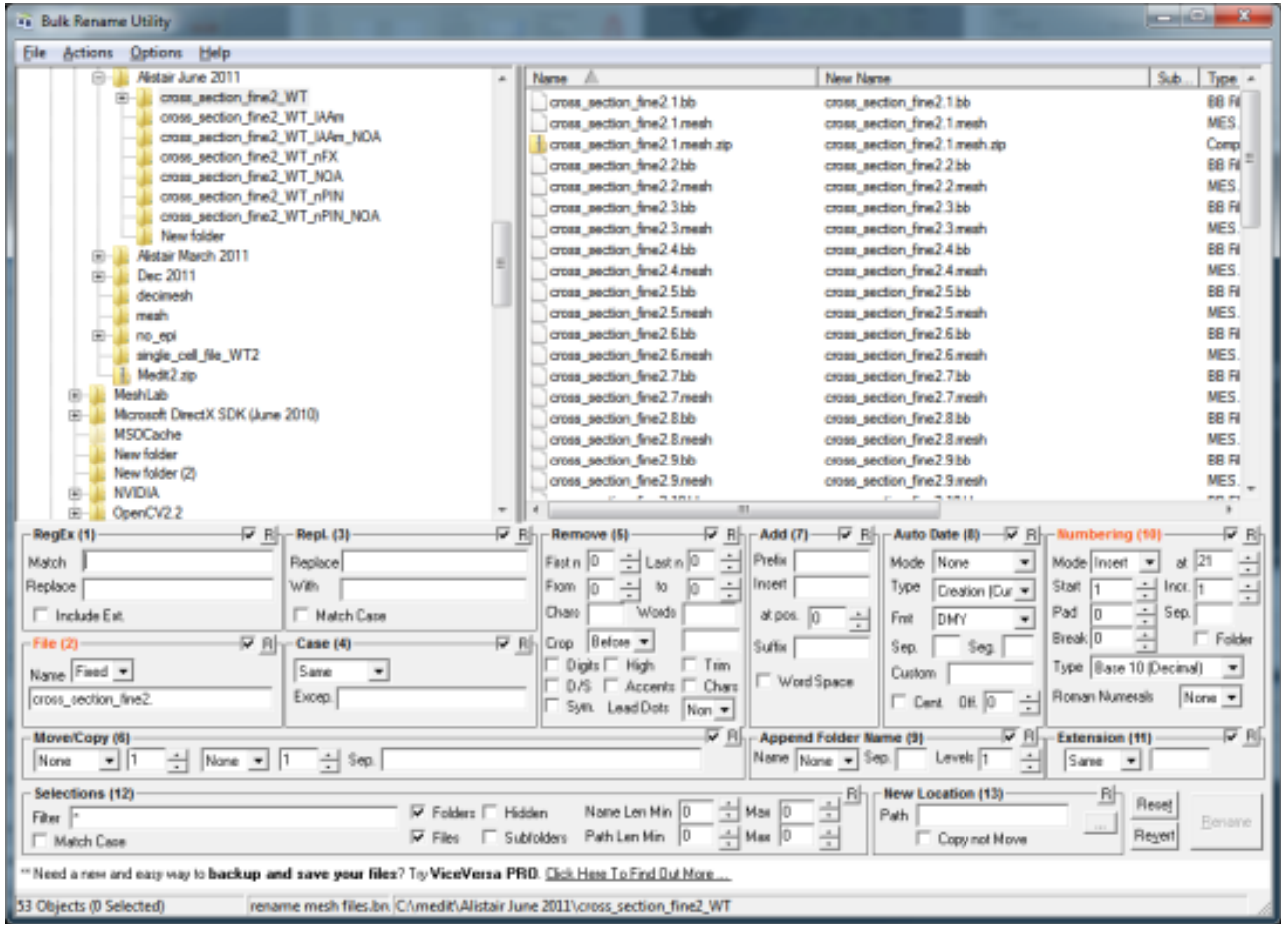


Appearance Over Function?

Over Complicated
What Does the User Need to Do?

Accessing the Controls That Needed

GUIs Not Always Way Forward!



University of Nottingham
UK | CHINA | MALAYSIA

Basics of GUI Programming

SWING AND JAVAFX

Basics of GUI Programming

- Event-driven.
  - Decides the order of execution depending whether click on a button, or select a drop-down, or choose a menu item etc.
- Implementing using a framework or library.
- Can be programmatically designed or drawn in a graphical editor.
- Various implementation options exist.
  - For different operating systems.
  - For different languages.

Swing

- Java's first attempt at a "modern" GUI approach.
  - Came about in the late 1990's …
  - Superseded/built on the earlier **AWT** (Abstract Window Toolkit).
  - Swing is lightweight, AWT is heavyweight.
    - (AWT relies on underlying native resources directly).
  - Swing still relies on AWT for some circumstances.
  - Can spot a Swing component as it starts with a **J**, e.g., `JFrame`.
  - Many existing Java GUIs use Swing, and it still has a strong following.
  - But Swing is being slowly retired in favour of JavaFX.

```java
package com.siebers;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class HelloSwingWorldApp extends JFrame {

    private HelloSwingWorldApp() { initUI(); }

    private void initUI() {
        setTitle("HelloSwingWorld");
        setSize( width: 400, height: 400);
        setLocationRelativeTo(null);
        JButton btn=new JButton( text: "Close App");
        btn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.out.println("Bye!");
                System.exit( status: 0);
            }
        });
        Container pane=getContentPane();
        pane.add(btn);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
```

```java
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            @Override
            public void run(){
                HelloSwingWorldApp window=new HelloSwingWorldApp();
                window.setVisible(true);
            }
        });
    }
}
```

```
module HelloSwingWorldApp.J12 {
    requires java.desktop;
}
```

JavaFX

- More recently introduced as part of JDK/JRE (until Java 10), but then removed from the JDK/JRE (from version 11 onwards) and available as external library.
  - JavaFX represents a big step forward.
  - Can deploy GUIs to tablet, phone, desktop etc.
  - Able to separate GUI code from program code using XML description file.

```java
package sample;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class Main extends Application {

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

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HelloJavaFXWorld");

        Button btn = new Button( s: "Close App");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Bye!");
                System.exit( status: 0);
            }
        });
```

```java
        StackPane root = new StackPane();
        root.getChildren().add(btn);

        primaryStage.setScene(new Scene(root,  v: 400,  v1: 400));
        primaryStage.show();
    }
}
```

The `StackPane` layout pane places all the nodes into a single stack where every new node gets placed on the top of the previous node.

Swing vs JavaFX

- Swing
  - A swing application is a class which extends `JFrame`.
  - Add components (e.g., buttons) to a `JPanel` which is then added to the frame.
  - Uses event handling.
  - Animation only possible if the code is written ourselves.
- JavaFX
  - The display area/window is called a `Stage`, the area of the GUI is called the `Scene`.
  - A scene is represented as a scene graph.
  - Controls, groups, layouts etc. are subclasses of the `Node` class.
  - Uses event handling.
  - Idea of Properties.
    - Can listen to changes of the variable.
    - Can bind properties together.
  - Formatting uses Cascading Style Sheets (CSS).
  - Special effects/animations supported.

Getting Started with JavaFX

- Requirements depend very much on IDE and Java version.
  - General
    - IntelliJ has everything needed to be embedded.
    - Good to have graphical editor to help with the design.
  - Java 11+
    - Check out https://openjfx.io/openjfx-docs/ for instructions of importing JavaFX.
    - Check out the instructions in first lecture slides.

```java
public class Main extends Application {

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

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HelloJavaFXWorld");

        Button btn = new Button( s: "Close App");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Bye!");
                System.exit( status: 0);
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);

        primaryStage.setScene(new Scene(root,  v: 400,  v1: 400));
        primaryStage.show();
    }
}
```

```java
package sample;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
```

`launch` reads the `Application` and then invokes `start`.

Execution then moves to the JavaFX thread.

JavaFX Theatre Analogy

The whole play contains several `Scenes`.

All the action `Scenes` take place on a `Stage`.

JavaFX's Stage and Scene

- Stage.
  - Think of it as an application window.
  - Depending on OS, there may be only one.
  - Equivalent to Swing's `JFrame` (or `JDialog`).

- Scene.
  - Equivalent to a content pane.
  - Can be considered as "view page".
  - Holds other objects (JavaFX `Node` objects).

```java
public class Main extends Application {

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

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HelloJavaFXWorld");

        Button btn = new Button( s: "Close App");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Bye!");
                System.exit( status: 0);
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);

        primaryStage.setScene(new Scene(root,  v: 400,  v1: 400));
        primaryStage.show();
    }
}
```

This is the window, and can set its text in the title bar etc.

The Node Class

- Fundamental to JavaFX.
- Used to represent controls, layouts, shapes etc.
- Can apply effects (transform, translate etc.) to nodes.
  - E.g., Button.

javafx.scene.control

Class Button

java.lang.Object
  javafx.scene.Node
    javafx.scene.Parent
      javafx.scene.layout.Region
        javafx.scene.control.Control
          javafx.scene.control.Labeled
            javafx.scene.control.ButtonBase
              javafx.scene.control.Button

All Implemented Interfaces:

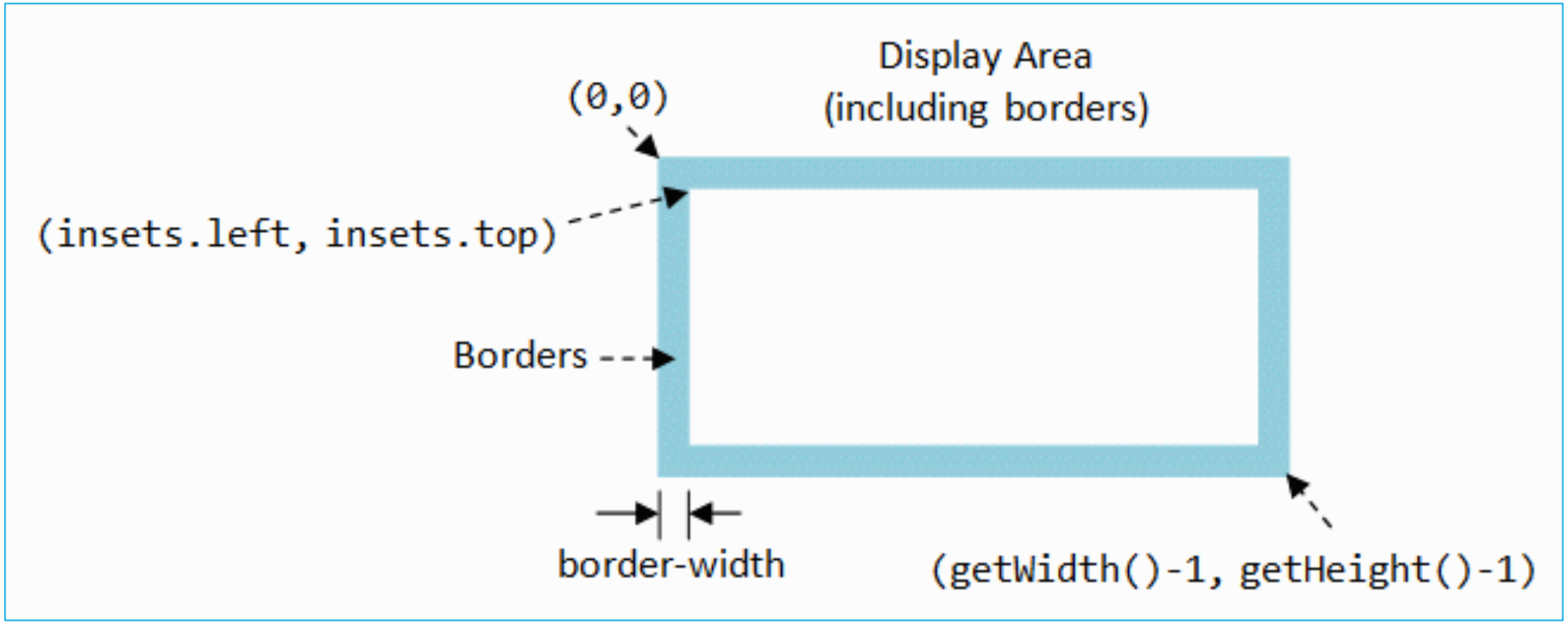
`Styleable, EventTarget, Skinnable`

Some JavaFX Controls

```java
public class Main extends Application {

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

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HelloJavaFXWorld");

        Button btn = new Button( s: "Close App");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Bye!");
                System.exit( status: 0);
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);

        primaryStage.setScene(new Scene(root,  v: 400,  v1: 400));
        primaryStage.show();
    }
}
```

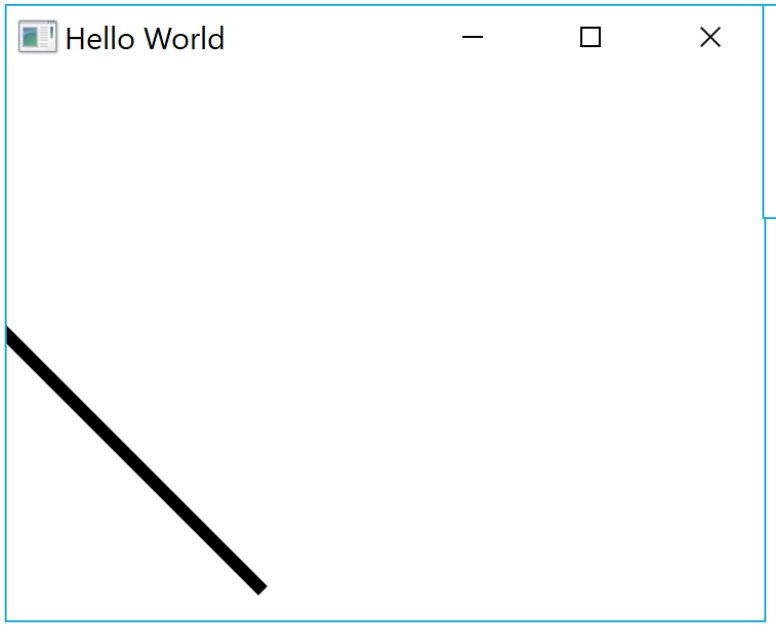
These are examples of Node objects.

Screen Coordinates in GUI



Adding Simple 2D Graphics

- Let's look an example of drawing a line simple `Hello World` window …




```
Line line = new Line( v: 0,  v1: 100,  v2: 100,  v3: 200);
line.setStrokeWidth(5);
root.getChildren().add(line);
```

Built-In Layout Panes

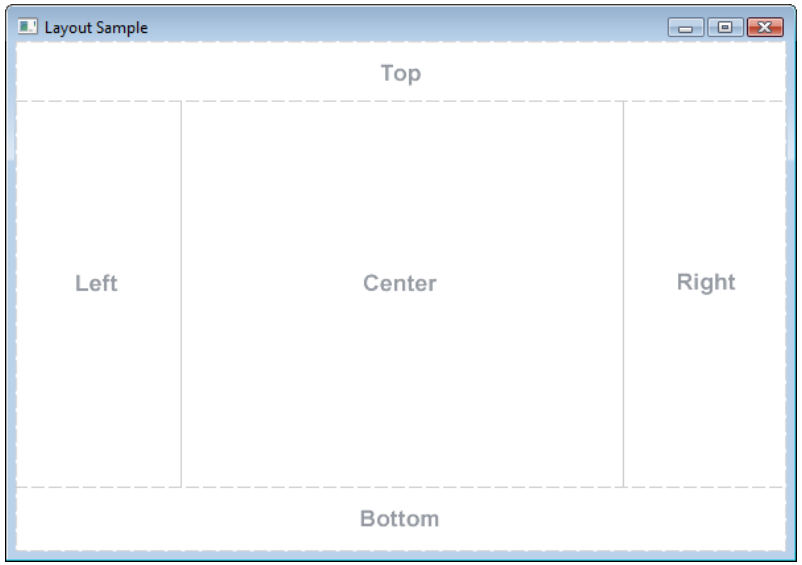
- BorderPane
  - Provides five regions in which to place nodes – top, bottom, left, right and centre.
- HBox / VBox
  - Provides an easy way for arranging a series of nodes in a single row / column.
- StackPane
  - Places all the nodes within a single stack with each new node added on top of the previous.
- GridPane
  - Allows to create a flexible grid of rows and columns in which to lay out nodes; nodes can be placed in any cell in the grid and can span cells as needed.

Built-In Layout Panes

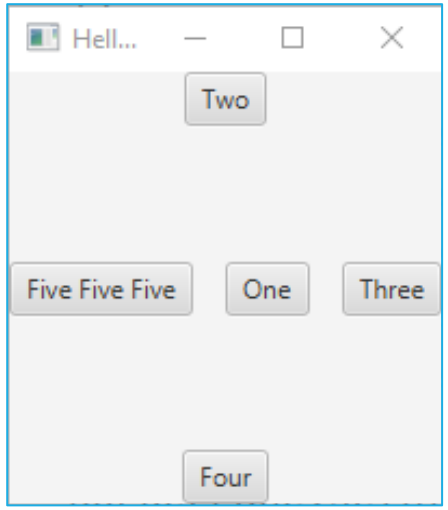
- FlowPane
  - Nodes are laid out consecutively and wrap at the boundary set for the pane; nodes can flow vertically (in columns) or horizontally (in rows).
- TilePane
  - Similar to a FlowPane; places all the nodes in a grid in which each cell, or tile, is the same size.
  - Nodes can be laid out horizontally (in rows) or vertically (in columns).
- AnchorPane
  - Allows to anchor nodes to the top, bottom, left side, right side, or centre of the pane; as the window is resized, the nodes maintain their position relative to their anchor point.

For more details, see https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

BorderPane

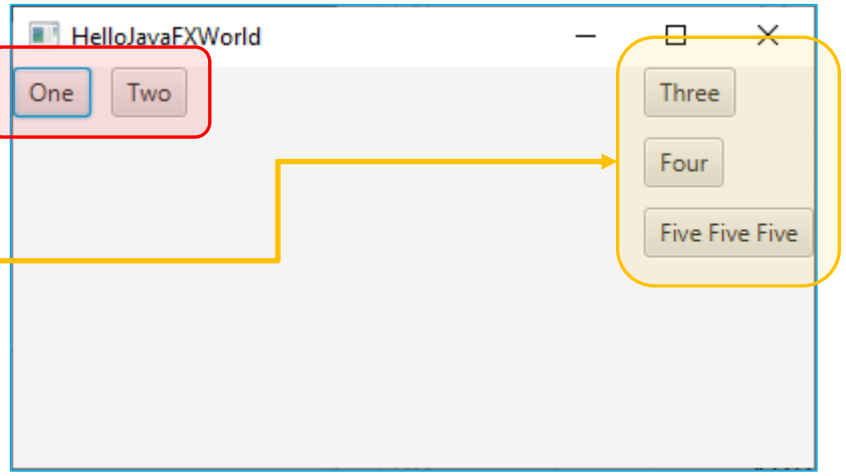



```
BorderPane.setAlignment(btn1, Pos.CENTER);
BorderPane.setAlignment(btn2, Pos.TOP_CENTER);
BorderPane.setAlignment(btn3, Pos.CENTER_RIGHT);
BorderPane.setAlignment(btn4, Pos.BOTTOM_CENTER);
BorderPane.setAlignment(btn5, Pos.CENTER_LEFT);
BorderPane root = new BorderPane(btn1, btn2, btn3, btn4, btn5);
```

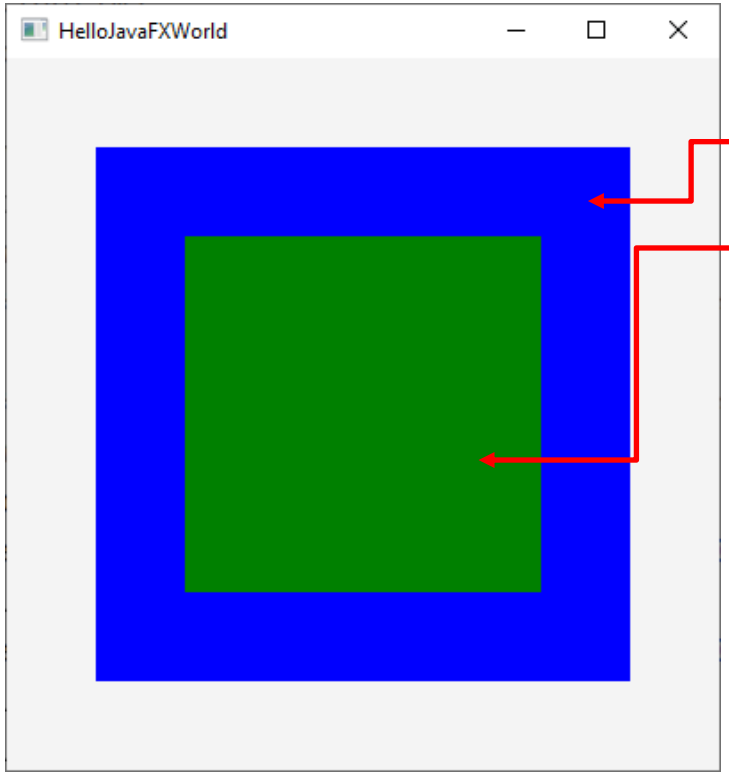


HBox / VBox

```
// using BorderPane with HBox and VBox
HBox hBox = new HBox();
VBox vBox = new VBox();
hBox.getChildren().addAll(btn1, btn2);
vBox.getChildren().addAll(btn3, btn4, btn5);

// set the gap between nodes
hBox.setSpacing(10);
vBox.setSpacing(10);

BorderPane root = new BorderPane();
root.setLeft(hBox);
root.setRight(vBox);
```



StackPane

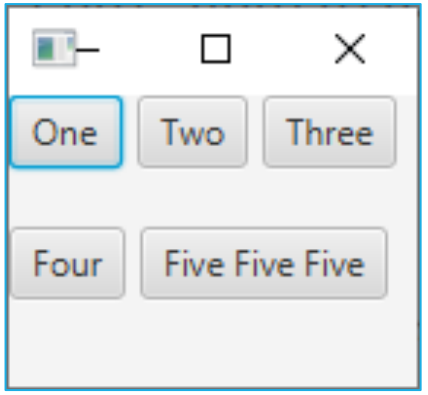



```
StackPane root = new StackPane();
Rectangle rect1 = new Rectangle( v: 0,  v1: 0,  v2: 300,  v3: 300);
rect1.setFill(Color.BLUE);
Rectangle rect2 = new Rectangle( v: 0,  v1: 0,  v2: 200,  v3: 200);
rect2.setFill(Color.GREEN);
root.getChildren().addAll(rect1, rect2);
```

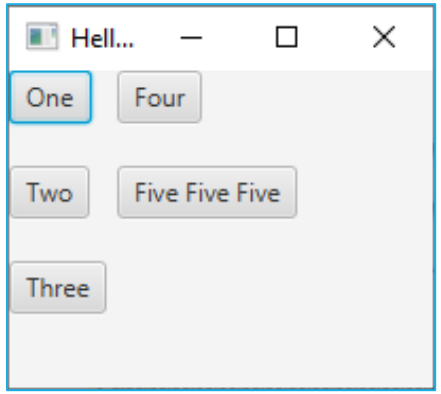
GridPane


```
GridPane root = new GridPane();
root.add(btn1,  i: 0,  i1: 0);          // column 1, row 1
root.add(btn2,  i: 1,  i1: 2);          // column 2, row 3
root.add(btn3,  i: 3,  i1: 1);          // column 4, row 2
root.add(btn4,  i: 1,  i1: 0);          // column 2, row 1
root.add(btn5,  i: 2,  i1: 2);          // column 3, row 3

root.setHgap(10);                      // set horizontal gap/spacing
root.setVgap(20);                      // set vertical gap/spacing
```

FlowPane




```
FlowPane root = new FlowPane();
root.setHgap(5);
root.setVgap(20);
root.getChildren().addAll(btn1, btn2, btn3, btn4, btn5);
```

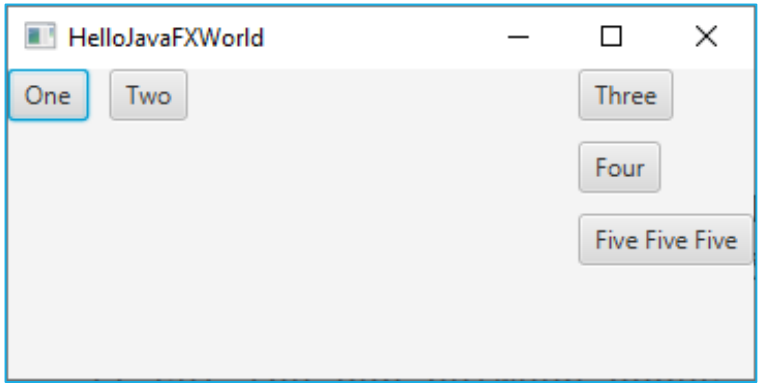



```
FlowPane root = new FlowPane();
root.setOrientation(Orientation.VERTICAL);
root.setHgap(5);
root.setVgap(20);
root.getChildren().addAll(btn1, btn2, btn3, btn4, btn5);
```

TilePane


```
TilePane root = new TilePane();
root.setHgap(5);
root.setVgap(20);
root.getChildren().addAll(btn1, btn2, btn3, btn4, btn5);
```

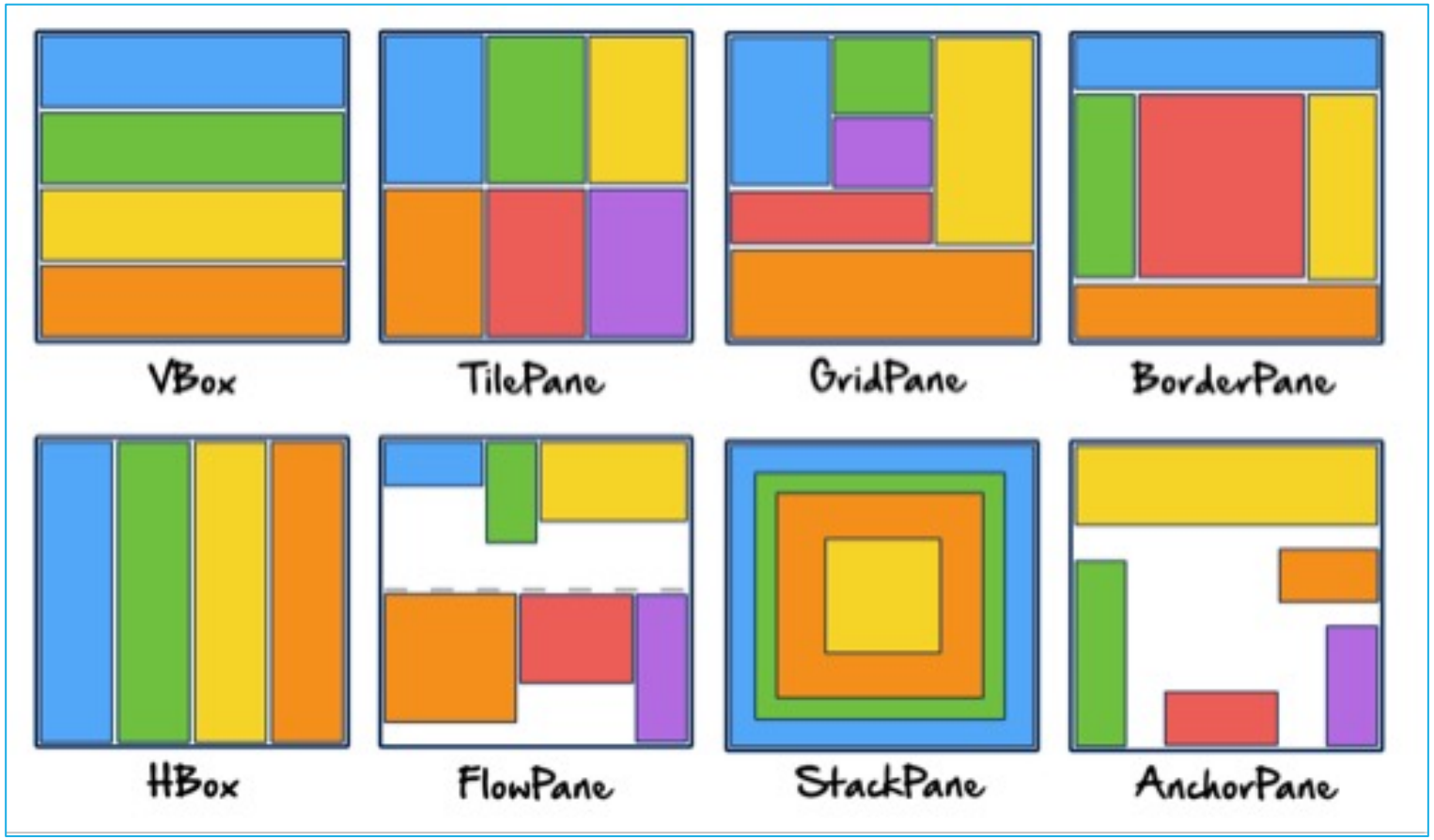
AnchorPane




```
BorderPane root = new BorderPane();
root.setLeft(hBox);
root.setRight(vBox);
```


```
AnchorPane root = new AnchorPane();
root.getChildren().addAll(hBox, vBox);
AnchorPane.setLeftAnchor(hBox, aDouble: 0.0);
AnchorPane.setRightAnchor(vBox, aDouble: 110.0);
```

Layout Overview



Properties & Binding

- JavaFX properties are often used in conjunction with binding, a powerful mechanism for expressing direct relationships between variables.
- When objects participate in bindings, changes made to one object will automatically be reflected in another object.
- Can also add a change listener to be notified when the property's value has changed.

See "Using JavaFX Properties and Binding": https://docs.oracle.com/javafx/2/binding/jfxpub-binding.htm

Properties & Binding Example

```
TextField textFieldInput = new TextField();

Label labelOutput = new Label( s: "Default Text");

BorderPane root = new BorderPane();
root.setTop(textFieldInput);
root.setCenter(labelOutput);
BorderPane.setAlignment(btn, Pos.CENTER);
root.setBottom(btn);

// using bind property
labelOutput.textProperty().bind(textFieldInput.textProperty());

// using event listener
textFieldInput.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observableValue, String oldValue, String newValue) {
        labelOutput.setText(newValue);
    }
});
```

Properties & Binding

- An `ObservableValue` is an entity that wraps a value and allows to observe the value for changes.
- The abstract class `Number` is the superclass of platform classes representing numeric values that are convertible to the primitive types: `byte`, `double`, `float`, `int`, `long` and `short`.

```
Slider slider = new Slider( v: 0,  v1: 100,  v2: 0);
root.setTop(slider);
slider.valueProperty().addListener(new ChangeListener<Number>() {
    @Override
    public void changed(ObservableValue<? extends Number> observableValue, Number oldValue, Number newValue) {
        labelOutput.setText("Value is " + (int)slider.getValue());
    }
});
```

See http://tutorials.jenkov.com/java/lambda-expressions.html for in-depth introduction to lambda expressions.

Lambda Expressions

- It is to be noticed that it is rather laborious to wire up events to buttons.

```
btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Bye!");
        System.exit( status: 0);
    }
});
```

- Is there a better way?
  - YES! Java 8 introduced **Lambda expressions**.

```
btn.setOnAction(event -> {
    System.out.println("Bye!");
    System.exit( status: 0);
});
```

Lambda Expressions (Slide 40)

```
// using event listener
textFieldInput.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observableValue, String oldValue, String newValue) {
        labelOutput.setText(newValue);
    }
});

// using event listener with lambda expression
textFieldInput.textProperty().addListener((observableValue, oldValue, newValue) -> {
        labelOutput.setText(newValue);
});
```

Lambda Expressions (Slide 41)

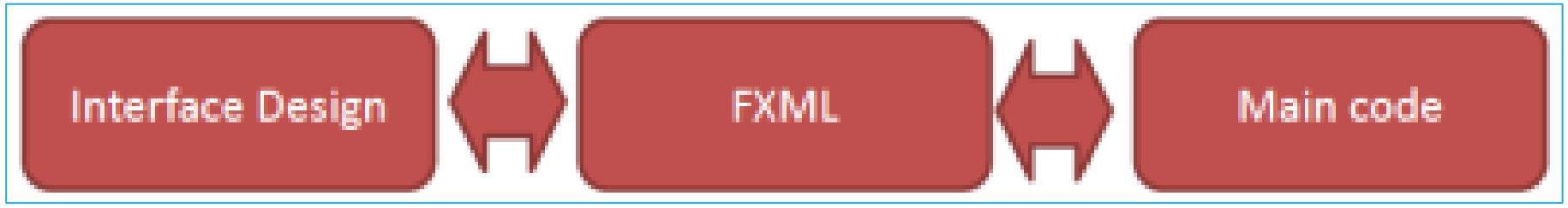
```
Slider slider = new Slider( v: 0,  v1: 100,  v2: 0);
root.setTop(slider);
slider.valueProperty().addListener(new ChangeListener<Number>() {
    @Override
    public void changed(ObservableValue<? extends Number> observableValue, Number oldValue, Number newValue) {
        labelOutput.setText("Value is " + (int)slider.getValue());
    }
});
```

```
slider.valueProperty().addListener((observableValue, oldValue, newValue) -> {
        labelOutput.setText("Value is " + (int)slider.getValue());
});
```

Designing with FXML

JavaFX and FXML

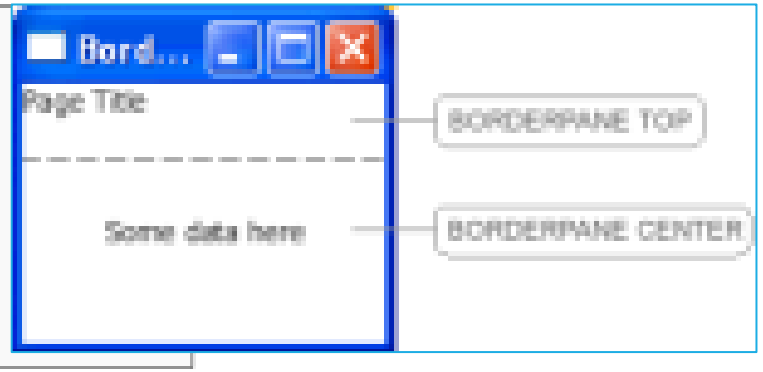
- Previous shows how simple GUIs can be built in code.
- BUT if the GUI is going to get complicated, JavaFX supports describing the layout using FXML (FX Markup Language).
- Supports the idea of separating "Design" and "Functionality".
  - Languages like C# in Visual Studio also support this separation of concerns.



JavaFX and FXML – From Oracle's Tutorial

Example 1-1 Java Code for a User Interface

```
BorderPane border = new BorderPane();
Label toppanetext = new Label("Page Title");
border.setTop(toppanetext);
Label centerpanetext = new Label ("Some data here");
border.setCenter(centerpanetext);
```



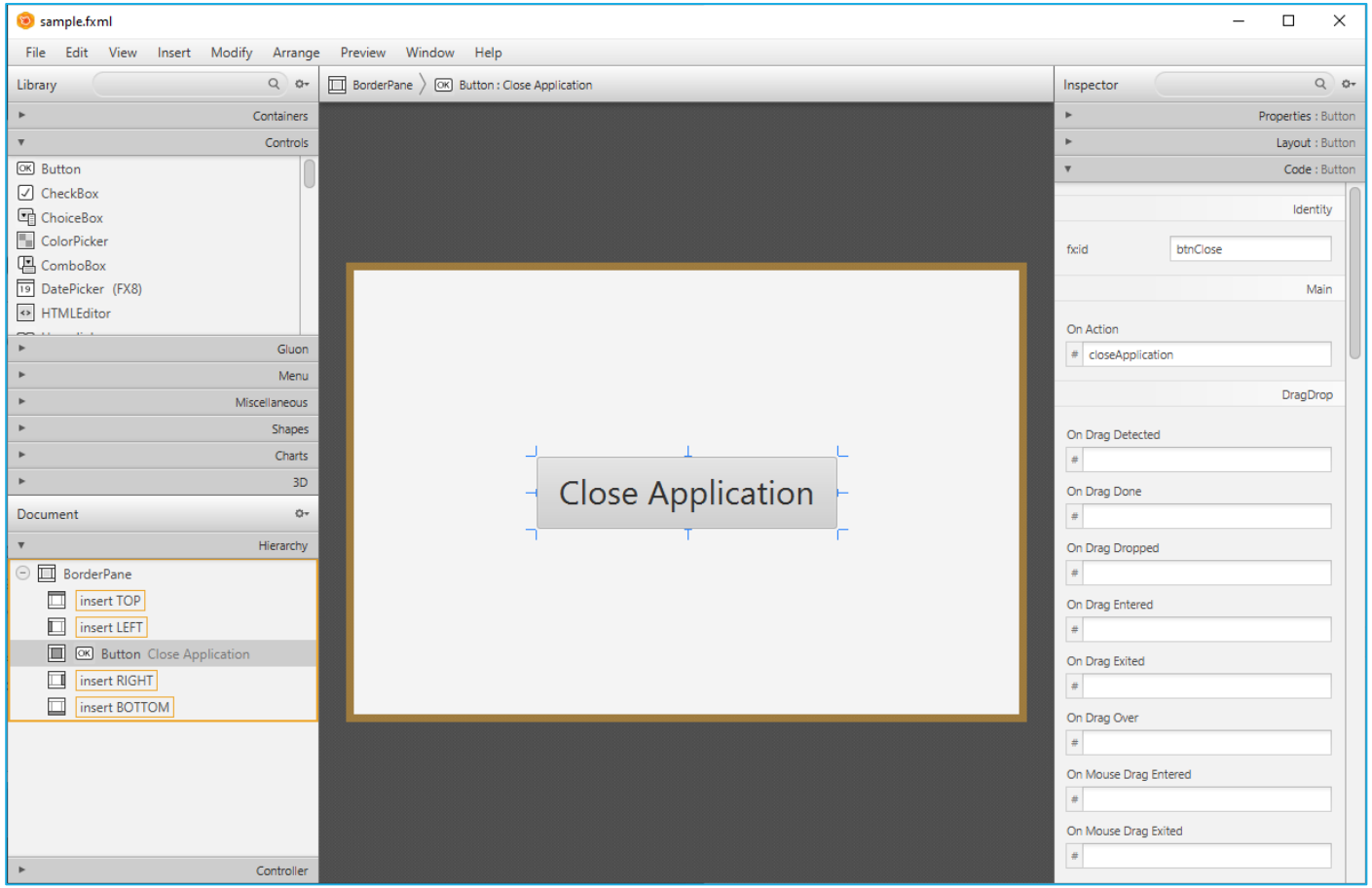
Example 1-2 FXML Markup for a User Interface

```
<BorderPane>
    <top>
        <Label text="Page Title"/>
    </top>
    <center>
        <Label text="Some data here"/>
    </center>
</BorderPane>
```

Simpler, less code, more intuitive organisation, and …

FXML Can Be Created Using Software

- SceneBuilder
  - A design tool for created FXML files graphically.



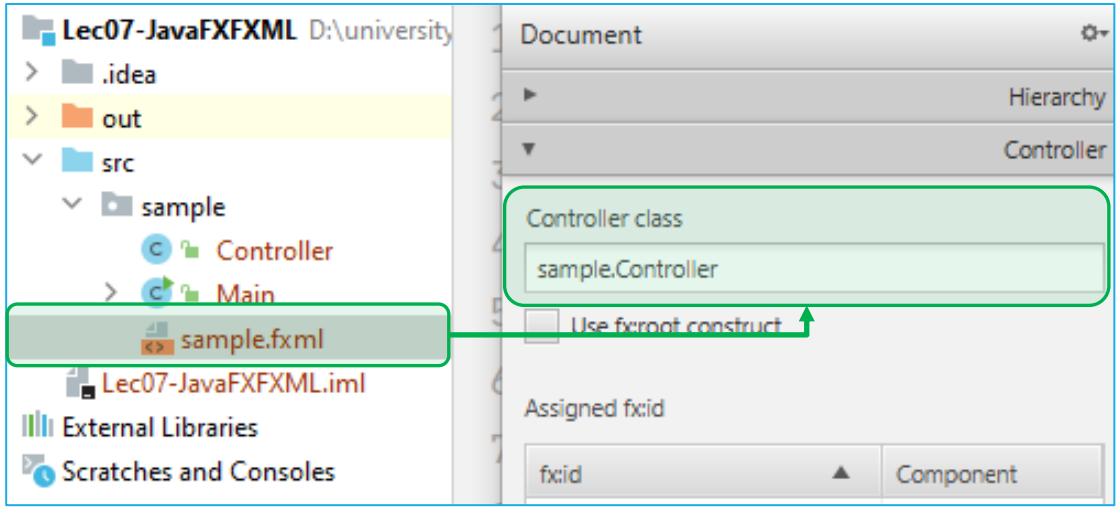
… Generates FXML Script …

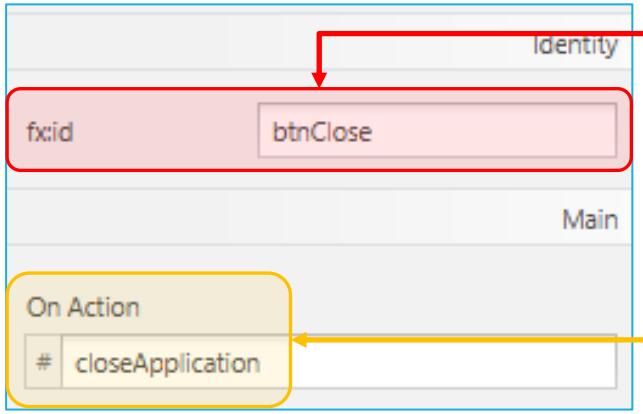
```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.text.Font?>
<BorderPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0"
            prefWidth="600.0" xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1">
    <center>
        <Button fx:id="btnClose" mnemonicParsing="false" onAction="#closeApplication" text="Close Application"
                BorderPane.alignment="CENTER">
            <font>
                <Font size="30.0"/>
            </font>
        </Button>
    </center>
</BorderPane>
```

Connect FXML Script to Code

- Define controller class in FXML file (if not done automatically).
- Give button an ID.
- Tell button which method to call.
- Copy sample controller skeleton.
  - View – Show Sample Controller Skeleton.
- Write controller code.
- Bonus
  - SceneBuilder offers a preview.

Connect FXML Script to Code






```
public class Controller {
    @FXML
    Button btnClose;

    @FXML
    private void closeApplication(ActionEvent event) {
        System.out.println("Bye!");
        System.exit( status: 0);
    }
}
```

Example: Mouse Event

On Mouse Clicked
mouseClick
On Mouse Dragged
#
On Mouse Entered
mouseEnter
On Mouse Exited
mouseExit

```
@FXML
private void mouseClick(MouseEvent event) {
    System.out.println("[INFO] Mouse Click!");
}

@FXML
private void mouseEnter(MouseEvent event) {
    System.out.println("[INFO] Mouse Enter!");
}

@FXML
private void mouseExit(MouseEvent event) {
    System.out.println("[INFO] Mouse Exit!");
}
```

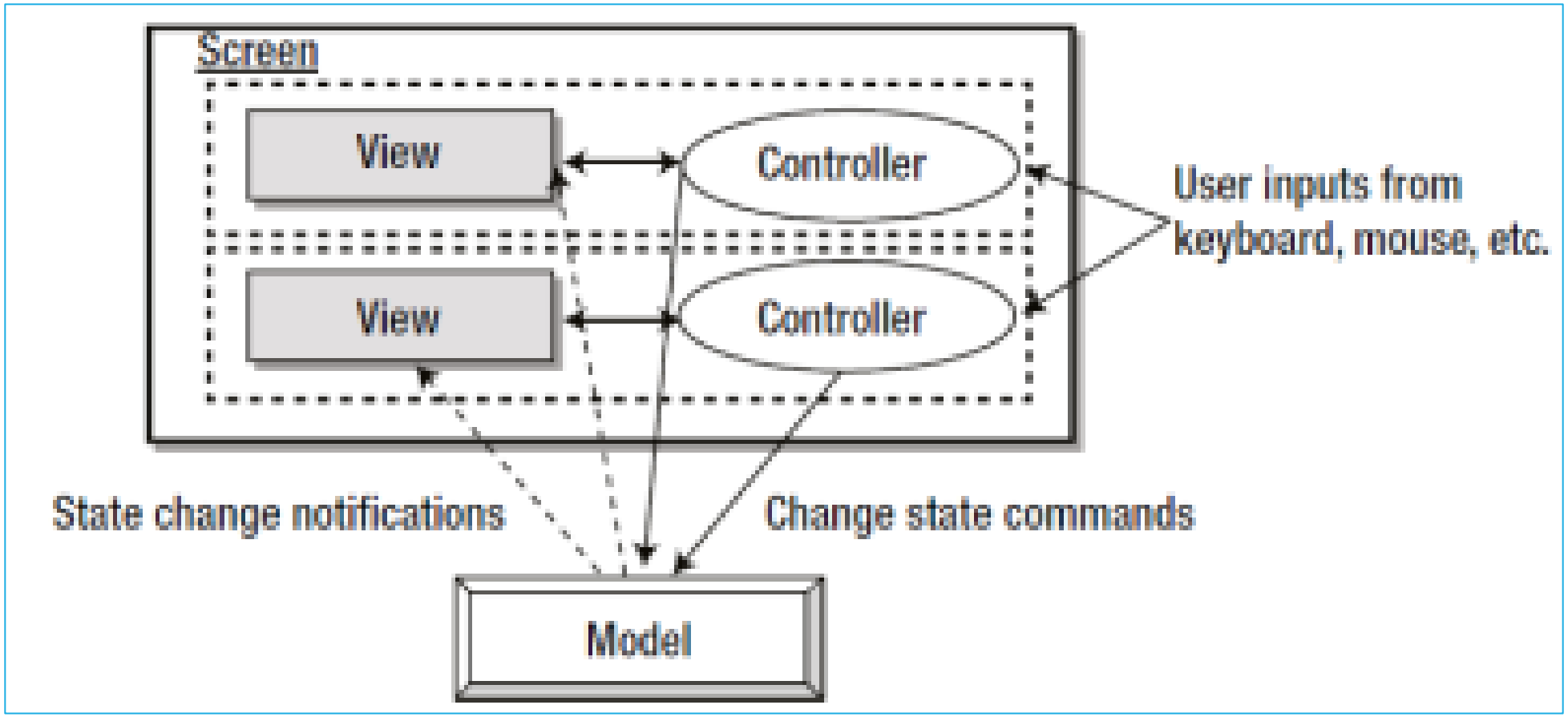
In Summary: Why Use FXML?

- UI designers might not be programmers.
- The designers can use external software (such as SceneBuilder) to design the look of the interface, whilst the programmers can build the functionality.
  - FXML glues two aspects together.
- Building GUIs visually rather than programmatically makes intuitive sense.
  - Though some still prefer to code it.
- Event handling is simplified (matter of opinion …).
- Maintenance: can fix code without touching GUI design or vice versa and it helps testing as can test `Control` without the UI.

MVC Design Pattern

MODEL-VIEW-CONTROLLER

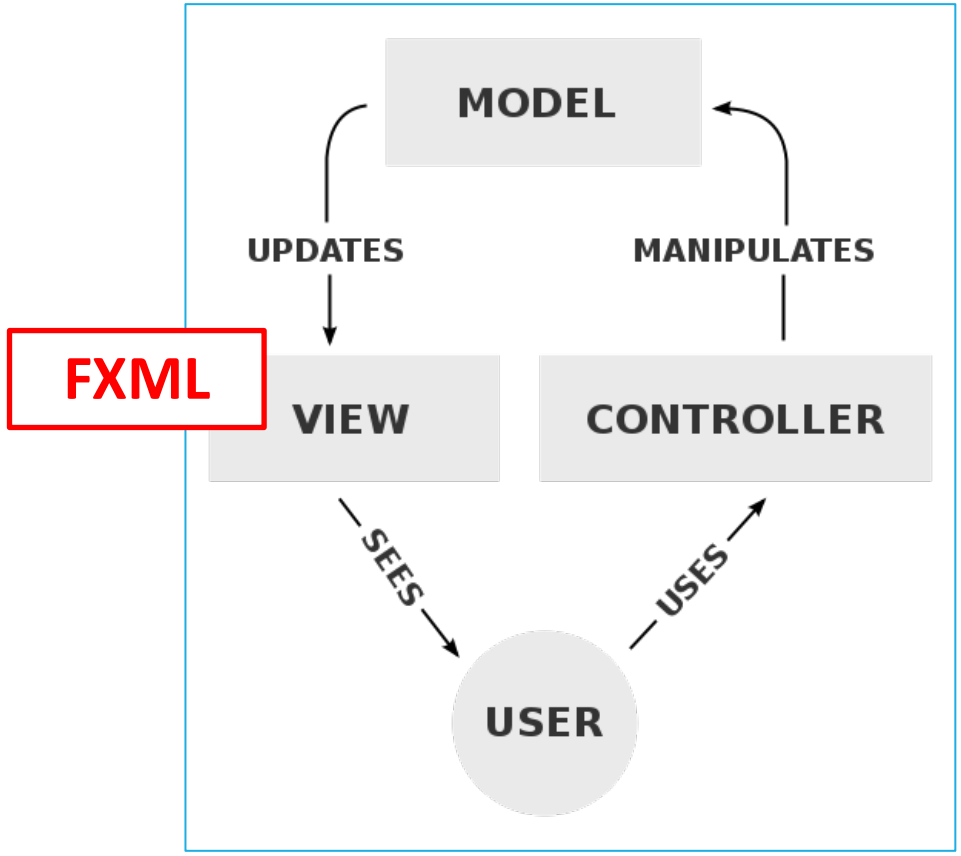
MVC


Screen
View
Controller
View
Controller
User inputs from keyboard, mouse, etc.
State change notifications
Change state commands
Model

MVC

- The *model* consists of the domain objects that model the real-world problems.
- The *view* and *controller* consist of the presentation objects that deal with the presentation such as input, output, and user interactions with GUI elements.
  - The *controller* accepts the inputs from the users and decides what to do with it.
  - The *view* displays the output on the screen.
  - Each *view* is associated with a unique *controller* and vice versa.
- The *model* is not aware of any specific *views* and *controllers*; but *views* and *controllers* are *model* specific.
- The *views* and *model* always stay in synchronization. The *model* notifies *views* about changes in its state, so *views* can display the updated data.

MVC Pattern


MODEL
UPDATES
MANIPULATES
FXML
VIEW
CONTROLLER
SEES
USES
USER

MVC

- What additional design pattern is embedded in the MVC?
  - Observer Pattern
    - The *model* provides a way for *views* to subscribe to its *state change modifications*.
    - Any interested *views* subscribe to the *model* to receive state change notifications.
    - The *model* notifies all *views* that had subscribed whenever a *model's* state changes.
- Variants of the MVC.
  - Application model MVC.
  - Model-View-Presenter (MVP).
  - Passive View MVP.

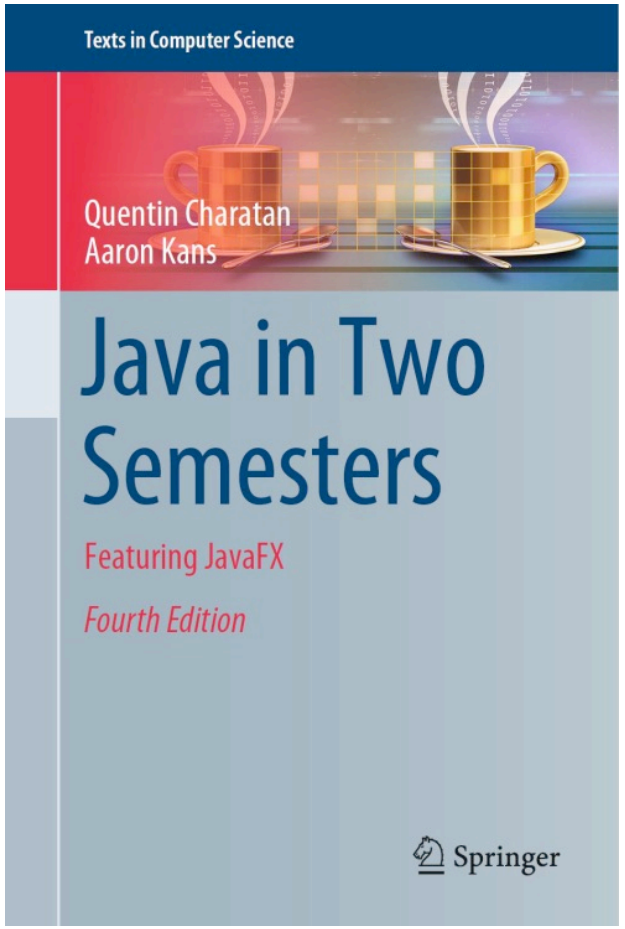
Useful Resources

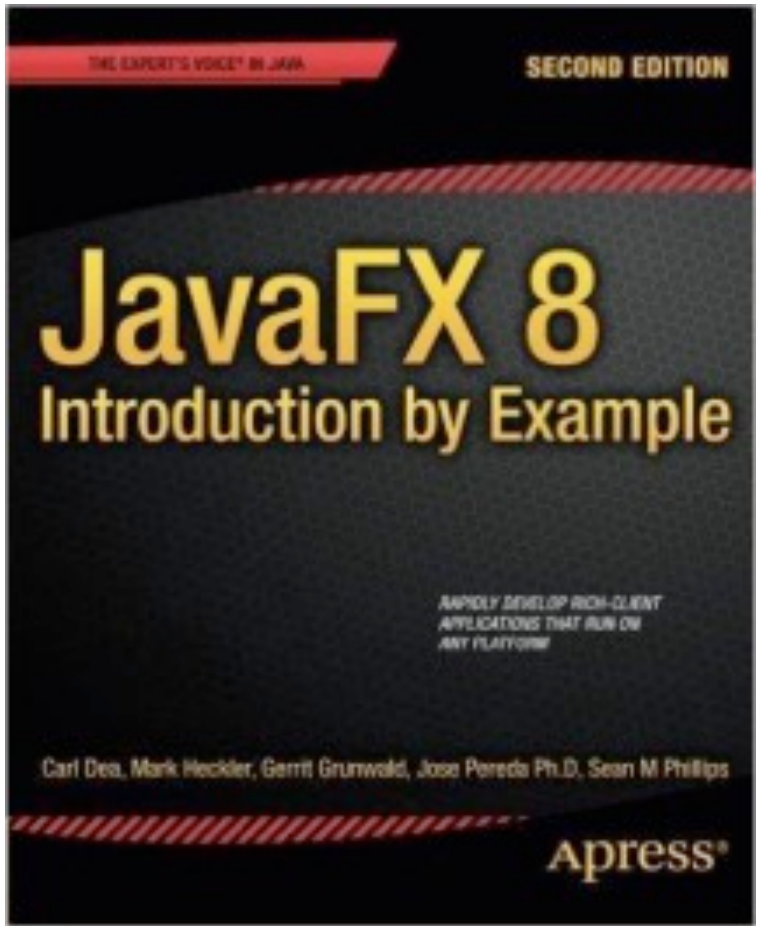
FOR GUI SKILLS IN MORE DEPTH

Links

- JavaFX Properties and Binding.
  - https://docs.oracle.com/javafx/2/binding/jfxpub-binding.htm
- Lambda Expression.
  - http://tutorials.jenkov.com/java/lambda-expressions.html
- JavaFX Layout Pane.
  - https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm
- Using FXML to Create a User Interface.
  - https://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm

References


Texts in Computer Science
Quentin Charatan
Aaron Kans
Java in Two Semesters
Featuring JavaFX
Fourth Edition
Springer


SECOND EDITION
JavaFX 8
Introduction by Example
Carl Dea, Mark Heckler, Gerrit Grunwald, Jose Pereda Ph.D, Sean M Phillips
Apress

Put Your Mind in Maintenance Mode