



CMPS 251

Read Chapter 12



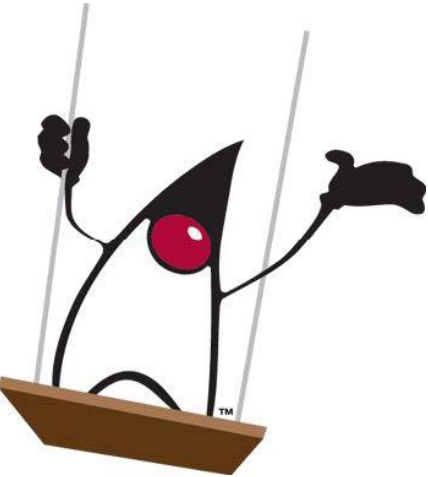
Graphical User Interfaces (GUI)

Dr. Abdelkarim Erradi
CSE@QU

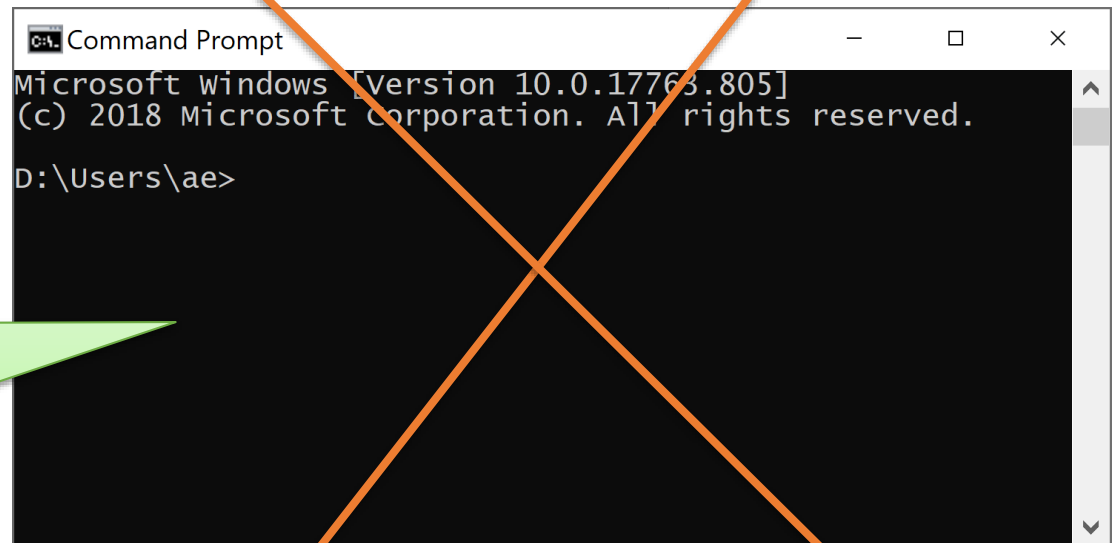
Outline

- GUI Programming Model
- Model-View-Controller (MVC) Pattern
- JavaFX Layout
- Handling Events

GUI Programming Model

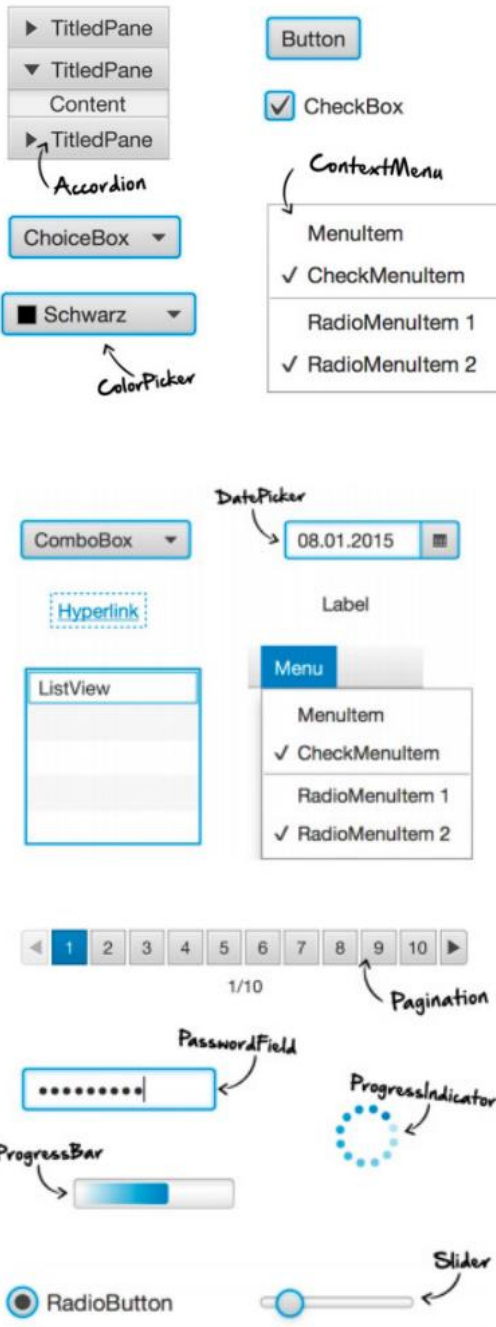


You have open
holidays!
We might send you
to the **Museum** 😊



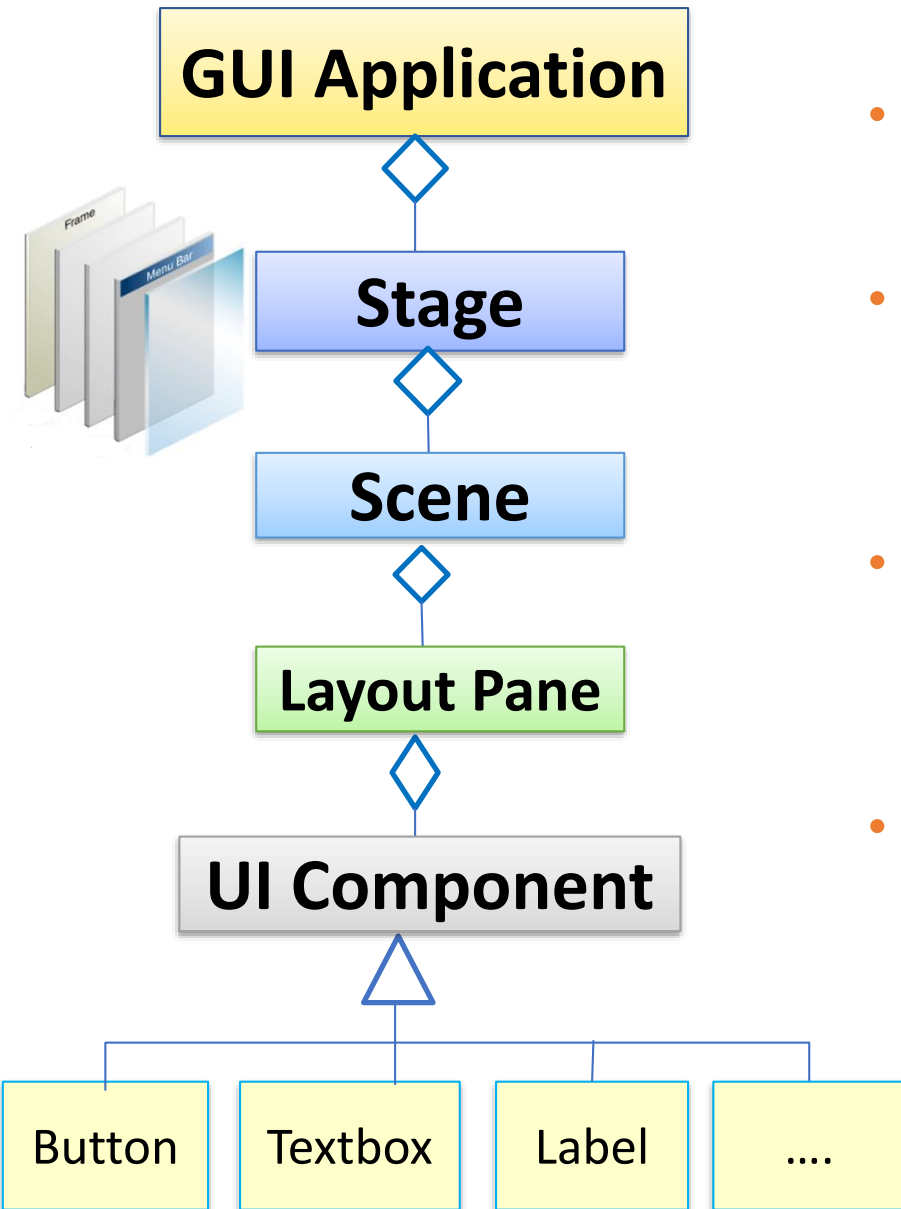
What is a GUI?

- **Graphical User Interface (GUI)** provides a visual User Interface (واجهة الاستخدام) for the users to interact with the application
 - Instead of a Character-based interface provided by the console interface 'the scary black screen' ➤
- **JavaFX** can be used for creating GUI



GUI Programming Model

IMPORTANT

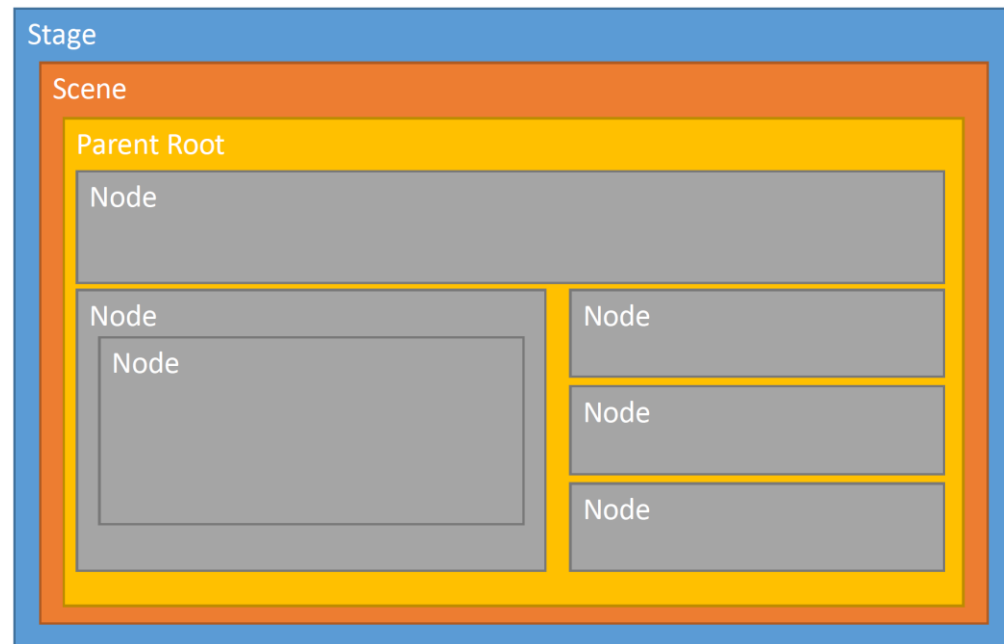
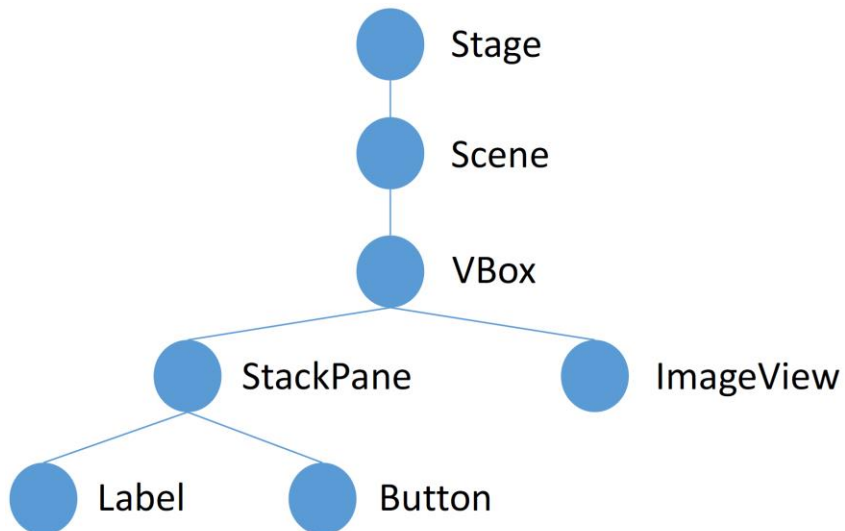


- GUI of an application is made up of **Windows** (JavaFX calls it **Stage**)
- A window has a **container** (called **Scene**) to host the UI root layout container
- UI Components are first added to a root **layout container** (such as VBox) then placed in the Scene
- UI Components **raise Events** when the user interacts with them (such as a MouseClicked event is raised when a button is clicked).
 - Programmer write **Event Handlers** to respond to the UI events

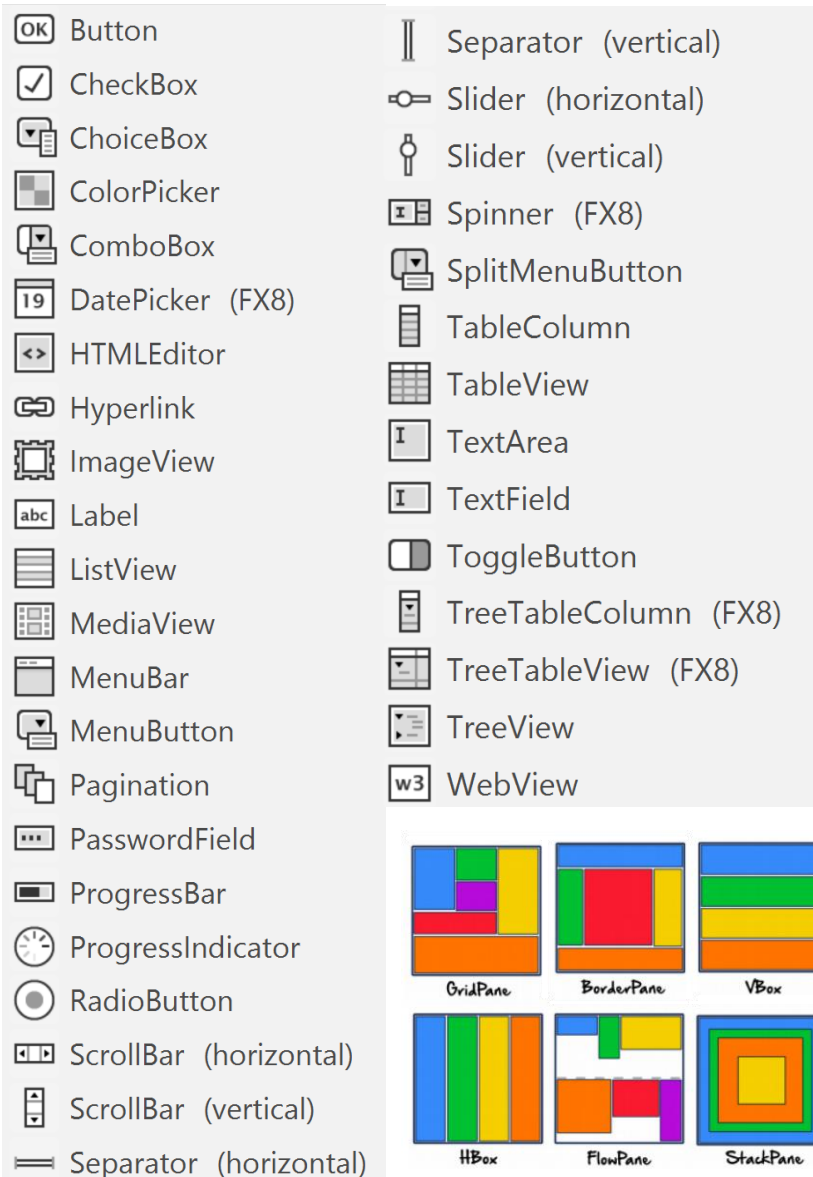
Structure of JavaFX application

Stage = **Window** where a scene is displayed

Scene = **Container** to host the UI root layout container

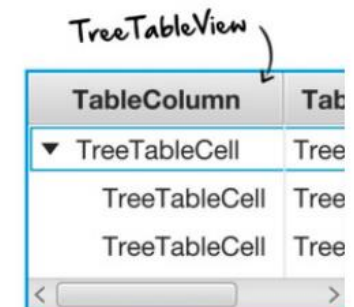
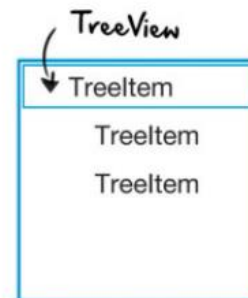
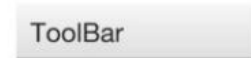
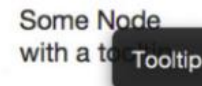
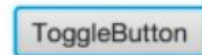
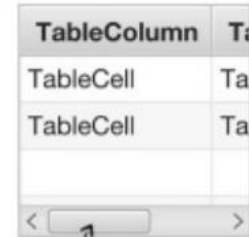
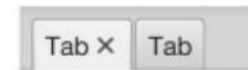
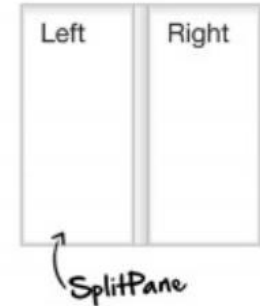
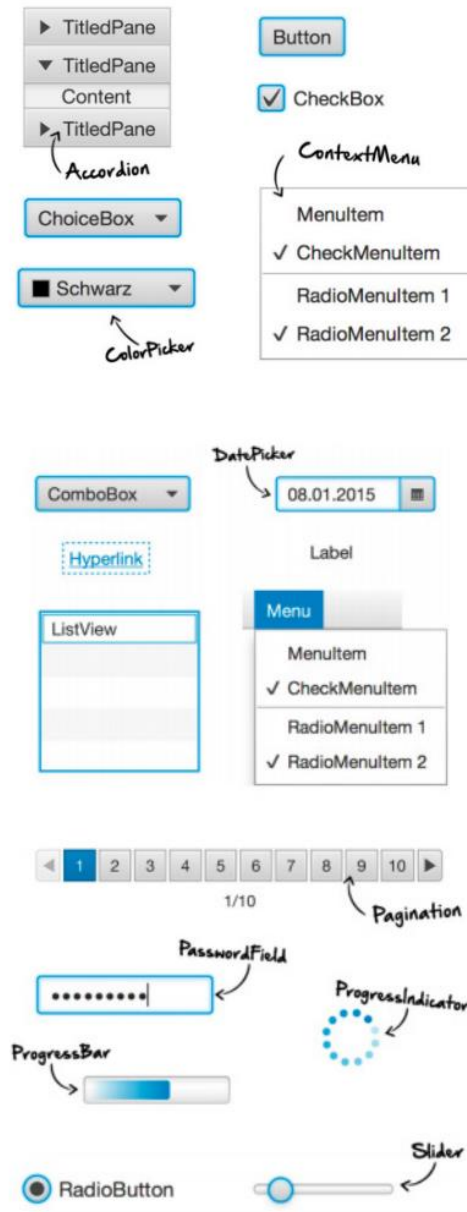


What Makes up ?

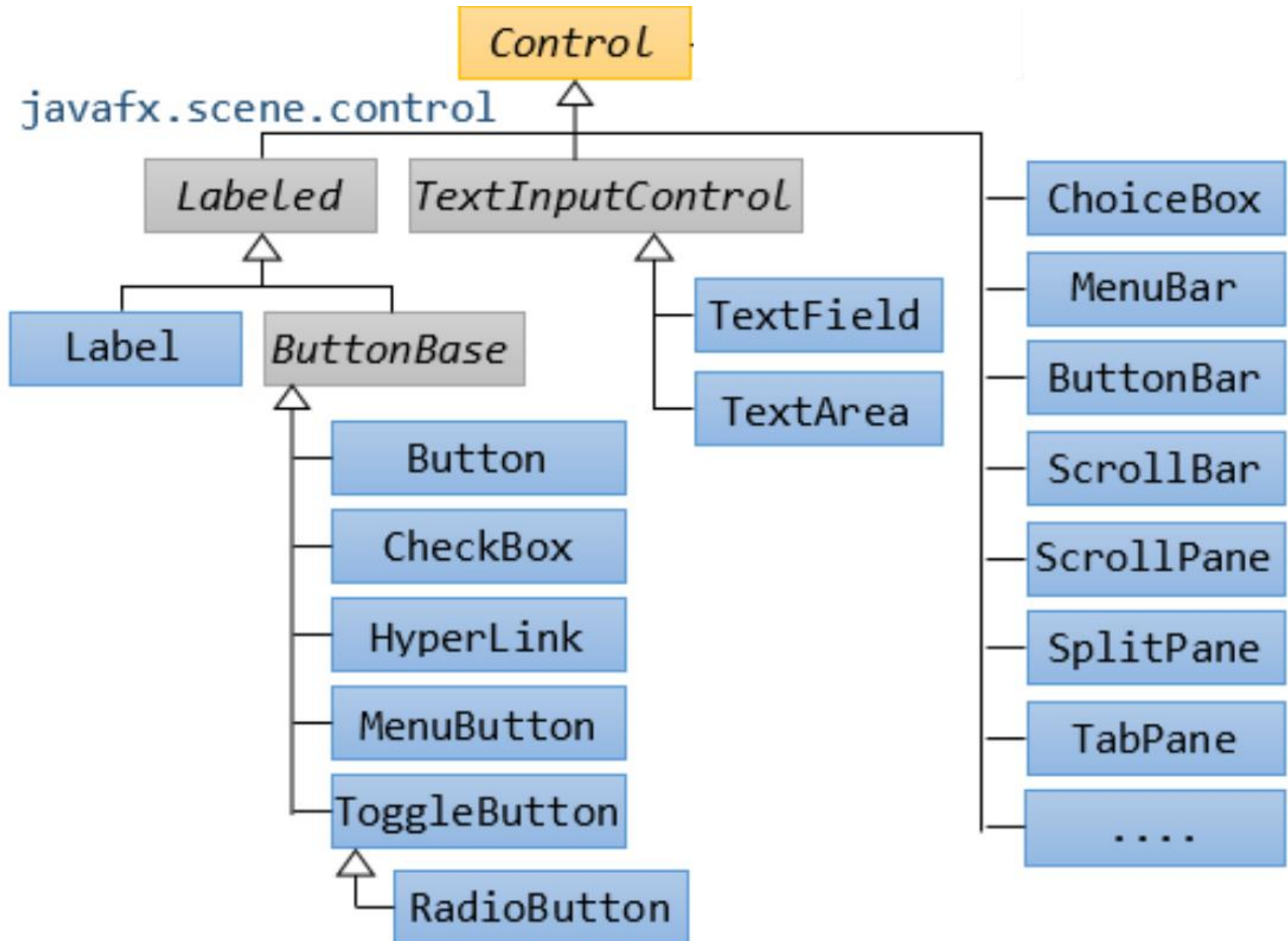


- **UI components**
 - Set of pre-built UI components that can be composed to create a GUI
 - e.g. buttons, text-fields, menus, tables, lists, etc.
- **Layout containers**
 - Control placement/positioning of components in the form (e.g., VBox and HBox)

JavaFX UI Components



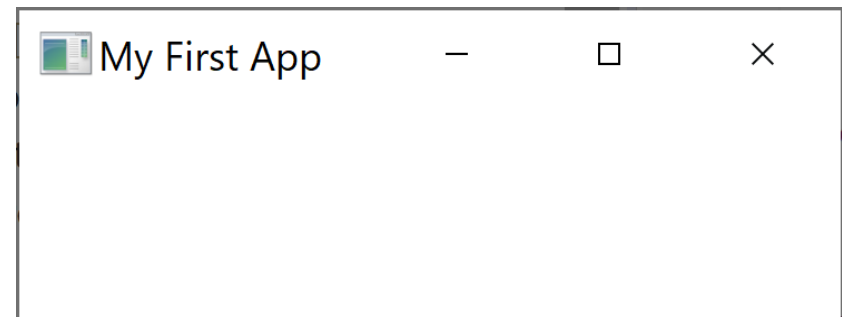
JavaFX UI Components Hierarchy



Creating JavaFX GUI: Stage (1/2)

- Create a class that extends `javafx.application.Application`
- Implement the `start(Stage stage)` method to build and display the UI
 - `start()` is called when the app is launched
- JavaFX automatically creates an instance of `Stage` class and passes to `start()`
 - when `start()` calls `stage.show()` a window is displayed

```
public class App extends Application {  
    @Override  
    public void start(Stage stage) {  
        stage.setTitle("My First App");  
        stage.show();  
    }  
  
    public static void main(String args[]) {  
        Launch(args);  
    }  
}
```



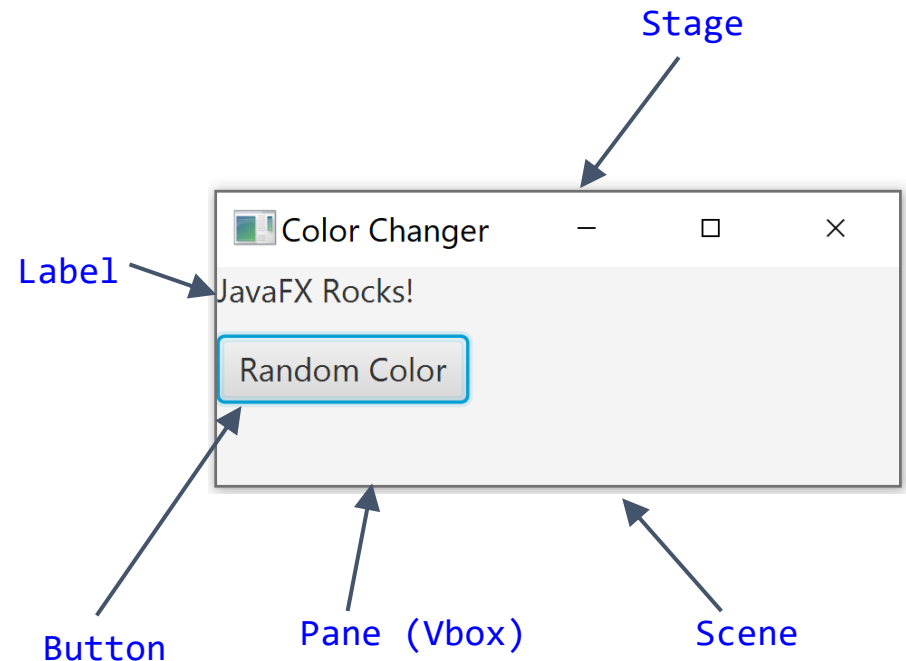
Creating JavaFX GUI : Scene (2/2)

- Create a **scene** (instance of `javafx.scene.Scene`) within the `start` method as the top-level container for the UI components
 - then pass the `scene` to the `stage` using the `setScene` method
- UI components (a Button, a Label...) can be added to a layout container (e.g., VBox) then added to the `Scene` to get displayed

```
public void start(Stage stage) {  
    VBox root = new VBox();  
    Label label = new Label("JavaFX Rocks!");  
    Button button = new Button("Submit");  
    root.getChildren().addAll(label, button);  
    Scene scene = new Scene(root, 200, 200);  
    stage.setScene(scene);  
    stage.show();  
}
```

JavaFX Application: ColorChanger

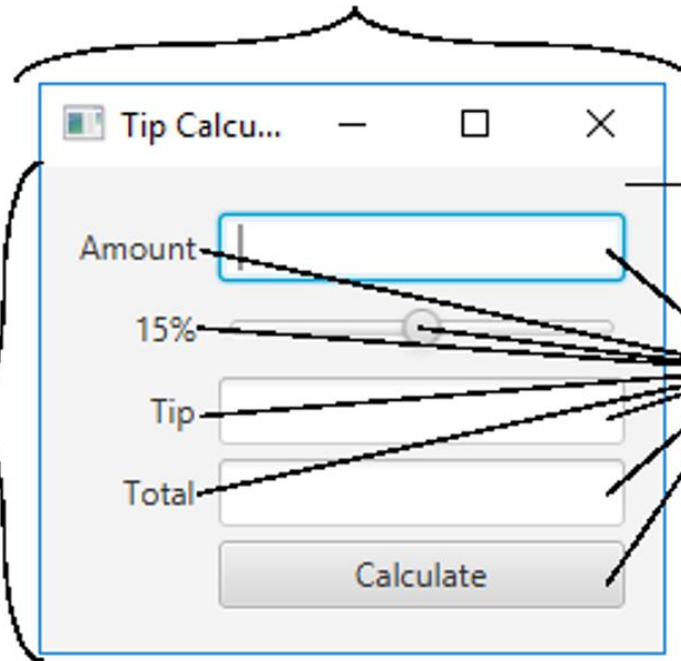
- App that contains text reading “JavaFX Rocks!” and a **Button** that randomly changes text’s color with every click



JavaFX App Components

The window is known as the stage

The stage contains a scene graph of nodes



The root node of this scene graph is a layout container that arranges the other nodes

Each of the JavaFX components in this GUI is a node in the scene graph



Label component

ImageView component

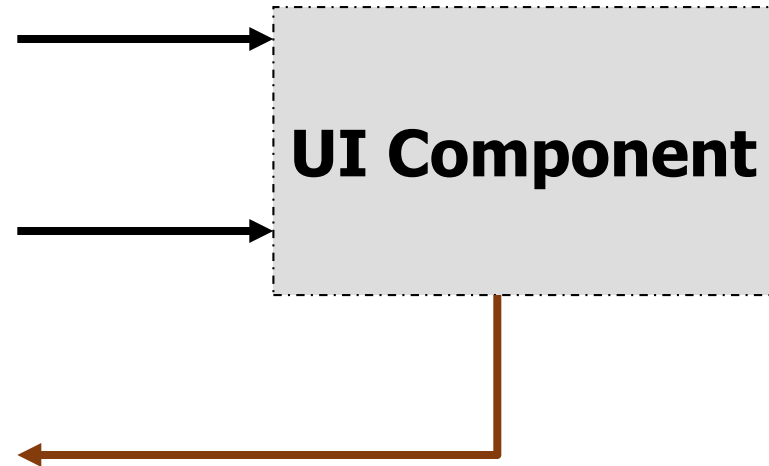
UI Component

- UI component is a class that has:

Attributes

Methods

Events



Using a UI Component



1. Create it

```
Button button = new Button("Press me");
```

Button

2. Initialize it / configure it

```
button.setTextFill( Color.BLUE );
```

3. Add it to a layout container

```
vBox.add(button);
```

Steps 1 to 3
can be done
using **Scene
Builder**

4. Listen to and handle its events

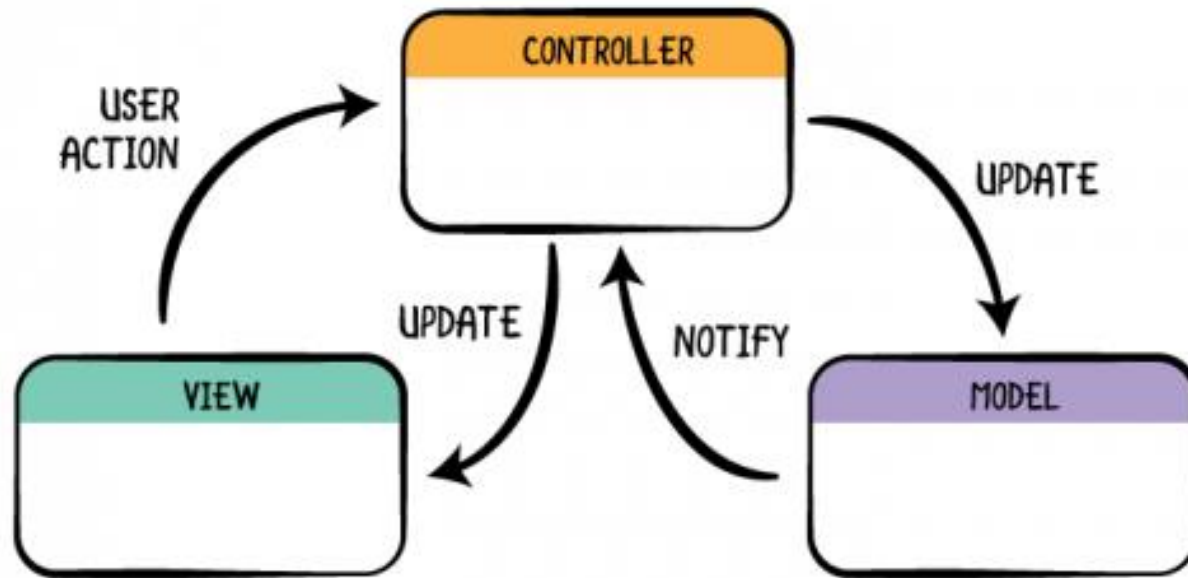
```
// Register an event handler
```

```
button.setOnMouseClicked( event ->  
    System.out.println(event) );
```



IMPORTANT

Building GUI Applications using the Model-view-controller (MVC) Pattern



MVC-based Application



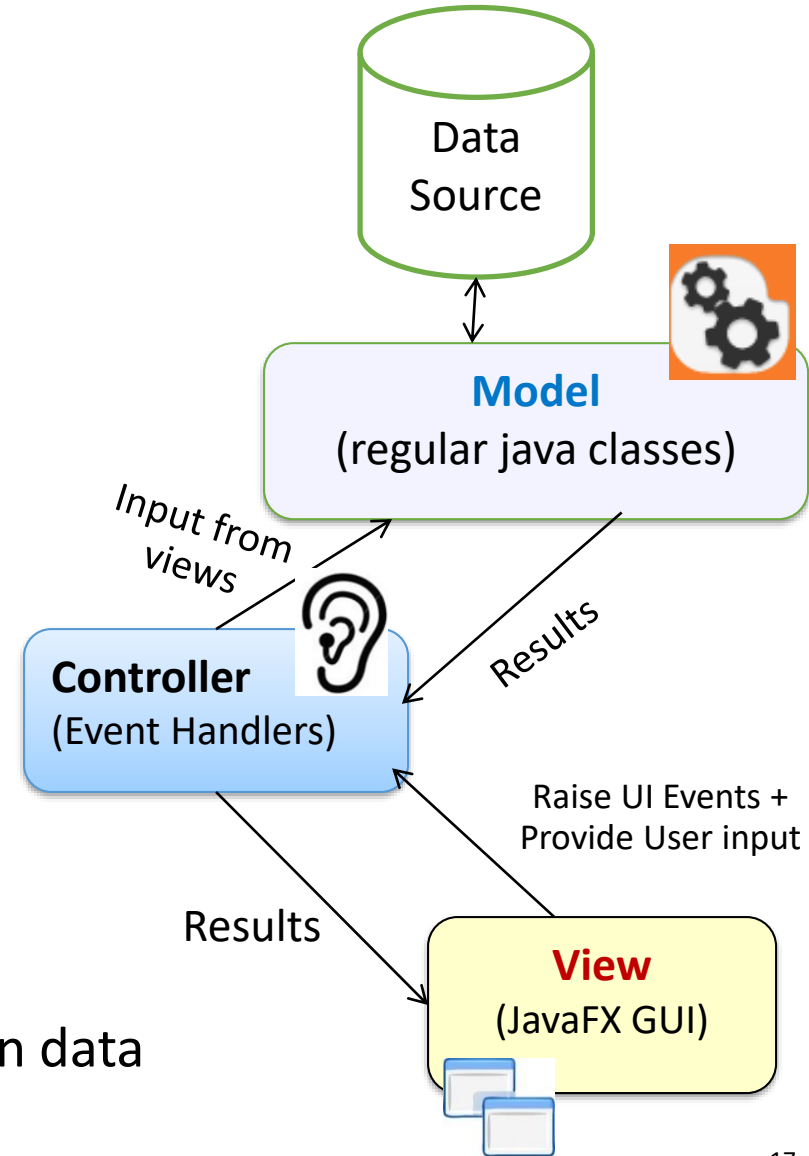
View

- Gets input from the user
- Notifies the controller about UI events
- Displays output to the user

Controller

- Handles events raised by the view
- Instructs the model to perform actions based on user input
e.g. request the model to get the list of courses
- Passes the results to the view to display the output

Model – implements business logic and computation, and manages the application data



Implementing MVC with JavaFX

1. Define the **model** (*ordinary Java classes*) to represent data and encapsulate computation
2. Build the **view** using *JavaFX Components* to collect input from the user and displays the results received from the controller
3. Use a **controller** to listen to and **handle events** raised by the view
 - Controller coordinates the execution of the request, get the request parameters from the View, calls the model to obtain the results (i.e., objects from the model)
 - Pass the results to the view to display the output

Advantages of MVC

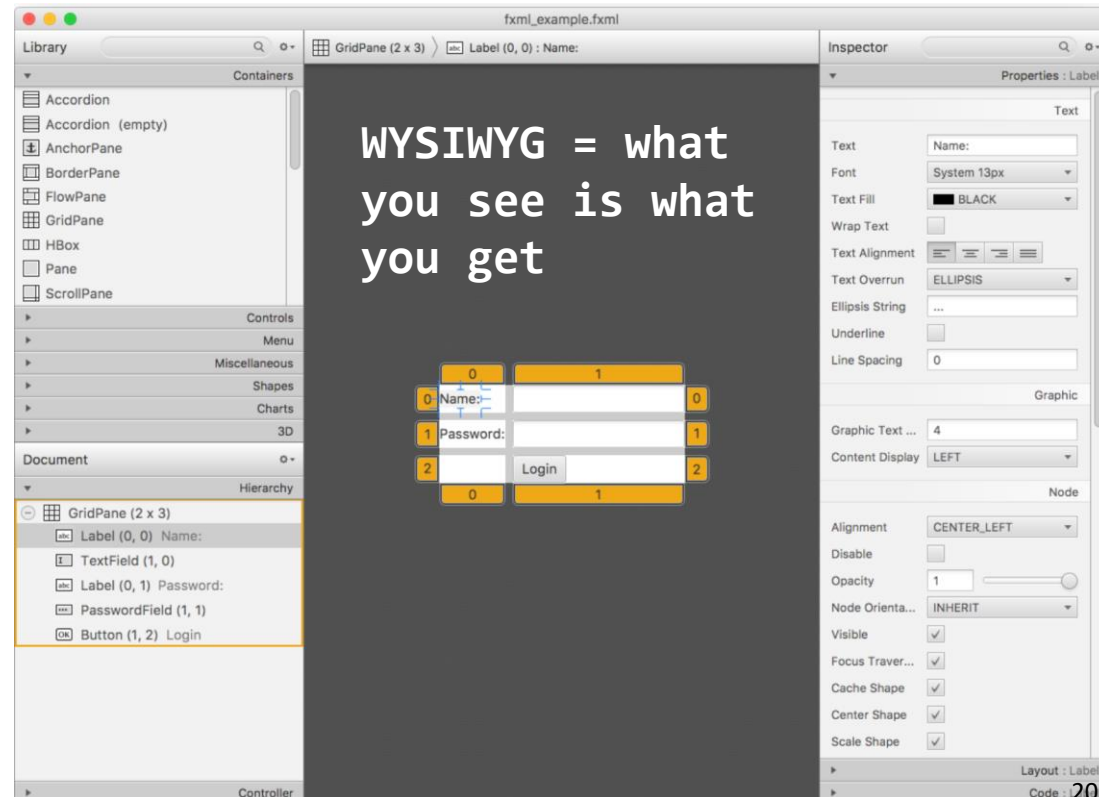


- ***Separation of concerns***
 - Views, controller, and model are **separate components**
 - Computation is not intermixed with Presentation. Consequently, code is cleaner, flexible and easier to understand and change.
 - Allow changing a component without significantly disturbing the others (e.g., UI can be completely changed without touching the model)
- **Reusability**
 - The same model can be used by different views (e.g., JavaFX view, Web view and Mobile view)

MVC is widely used and recommended particularly for interactive applications with GUI

Building the View using FXML

- You can create the View using Java code or FXML
- FXML is an XML-based language that defines the **structure** and **layout** of the View
- FXML allows a **clear separation** between the view of an app and the logic
- SceneBuilder is a WYSIWYG editor for FXML



Loading FXML file into a stage

```
@Override
```

```
public void start(Stage stage) throws Exception {  
    Parent root =  
        FXMLLoader.Load(getClass().getResource("welcome.fxml"));  
    stage.setTitle("Welcome to JavaFX");  
    stage.setScene(new Scene(root, 400, 300));  
    stage.show();  
}
```

FXML Controller

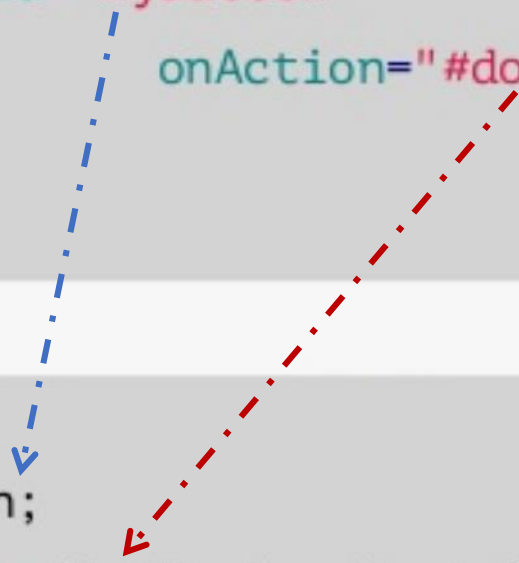
- FXML file is associated with a **Controller** class that implements all the events handlers
 - Referenced by **fx:controller** attribute in the FXML
- The Controller defines:
 - attributes associated to the *view components*
 - FXML elements to be accessed programmatically in the controller must be named using **fx:id** attribute
 - *event handlers* (e.g., handle button onAction event)

```
public class LoginController {  
    @FXML private Button loginBtn;  
  
    @FXML void handleLogin(ActionEvent event) {  
        System.out.println("Login pressed");  
    }  
}
```

FXML + Controller

```
<BorderPane fx:controller="com.SuperController">  
  <center>  
    <Button text="Click !" fx:id="myButton"  
            onAction="#doSomething" />  
  </center>  
</BorderPane>
```

```
public class SuperController {  
  @FXML private Button myButton;  
  @FXML private void doSomething(ActionEvent e) {}  
}
```

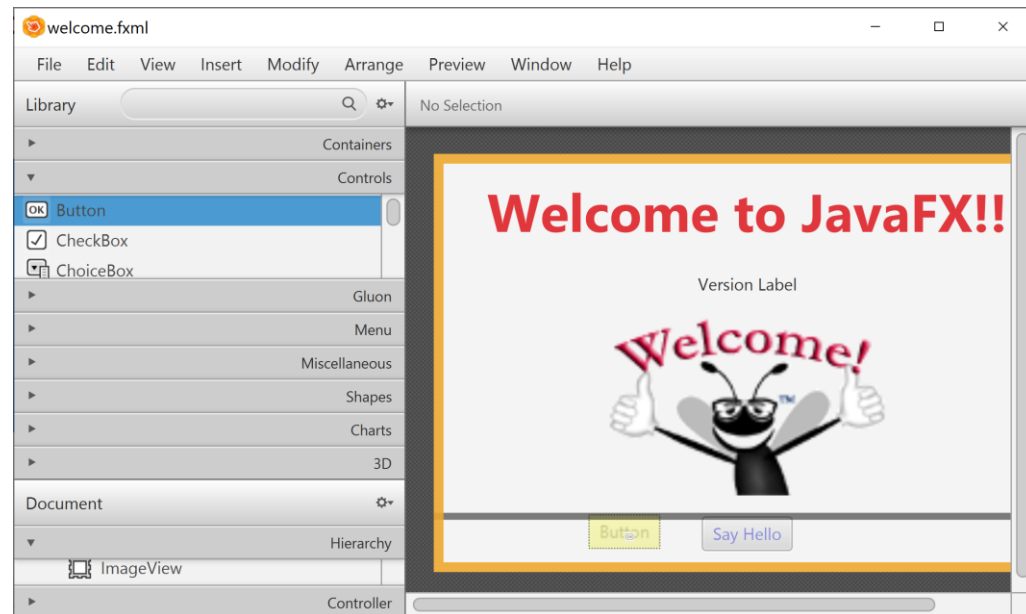


Use Java code to create
JavaFX UI, works for
you...



Don't listen to him,
use FXML

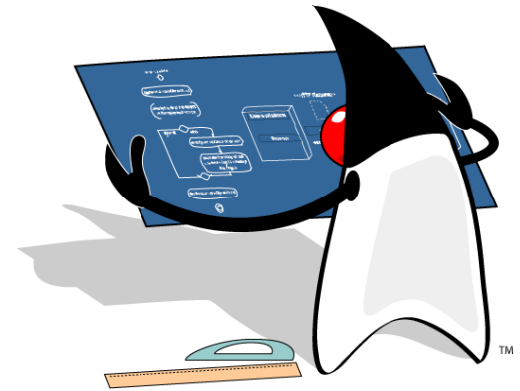
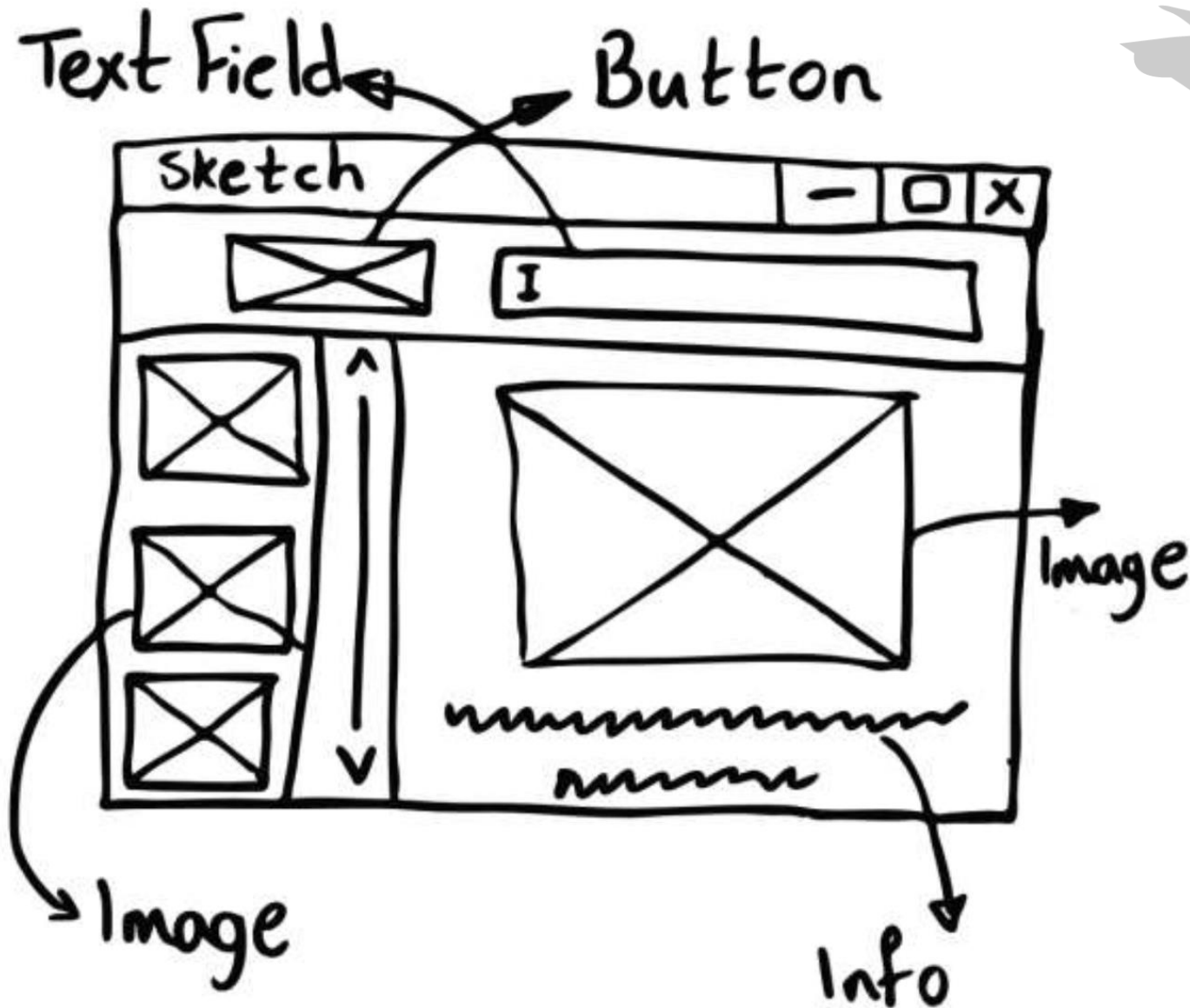
```
VBox root = new VBox();  
Label label = new Label("JavaFX Rocks!");  
Button button = new Button("Random Color");  
button.setTextFill(Color.BLUE);  
root.getChildren().addAll(label, button);  
root.setSpacing(20);  
root.setAlignment(Pos.CENTER);
```



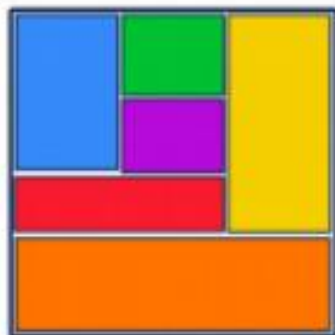
Steps to creating a GUI Interface

1. Design it on paper (sketch)
 - Decide what information to present to user and what input they should supply
 - Decide the UI components and the layout on paper
2. Create a view and add components to it (using either SceneBuilder or java code)
 - Use layout panes to group and arrange components
3. Add event handlers to respond to the user actions (event driven programming)
 - Do something when the user presses a button, moves the mouse, change text of input field, etc.

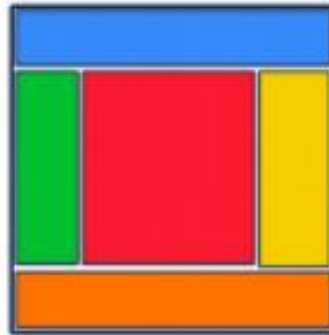
UI Sketch - Example



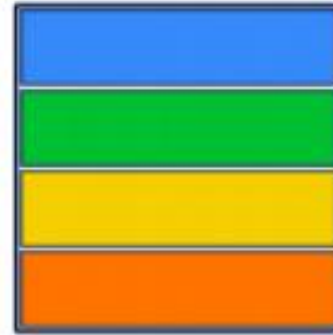
Layouts



GridPane



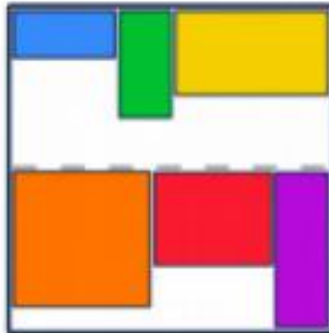
BorderPane



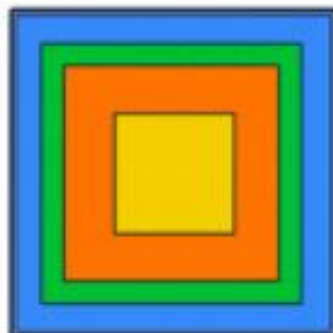
VBox



HBox



FlowPane



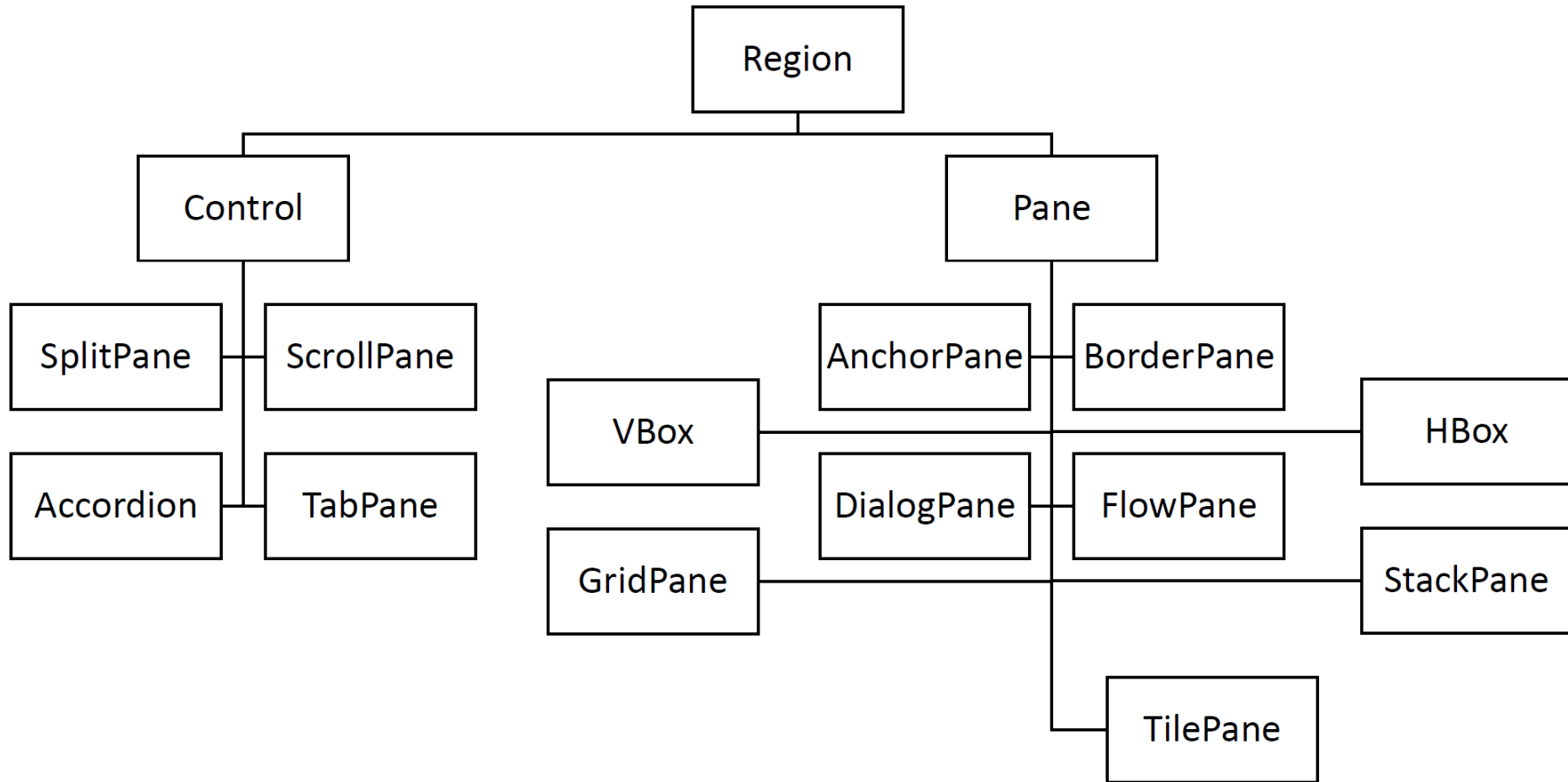
StackPane

Layouts



- Layout classes are called **Panes** in JavaFX
- Layout Pane automatically **controls** the **size** and **placement** of components in a container
 - Frees programmer from handling/hardcoding positioning of UI elements
 - As the window is resized, the UI components reorganize themselves based on the rules of the layout

JavaFX Containers Hierarchy



Common Layouts



VBox



HBox



BorderPane



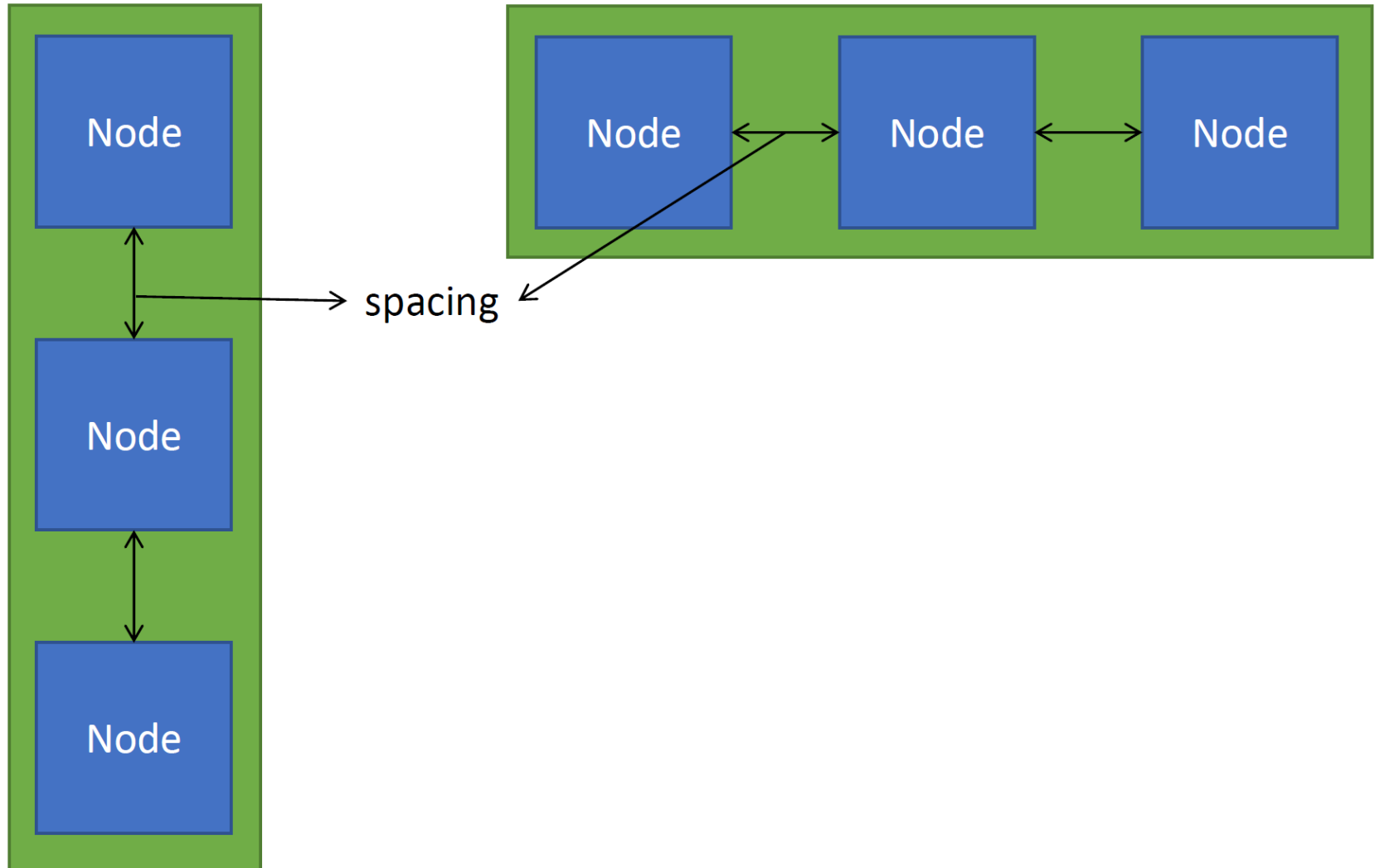
FlowPane



GridPane

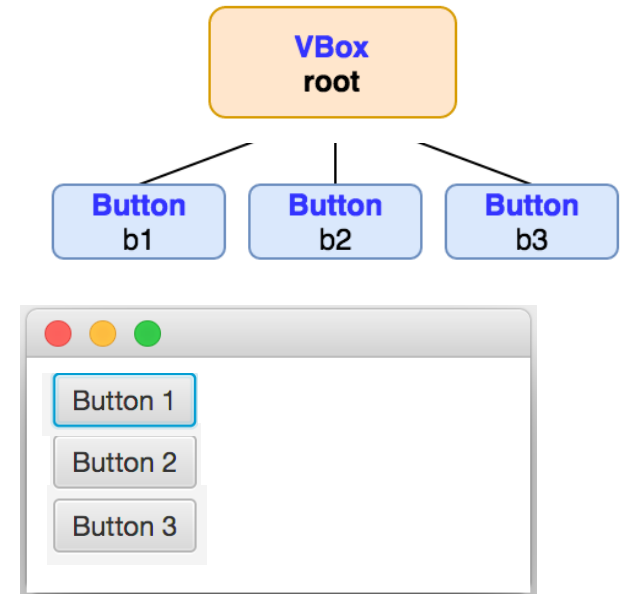
- **VBox** - displays UI elements in a vertical line
- **HBox** - displays UI elements in a horizontal line
- **BorderPane** - provides five areas: top, left, right, bottom, and center.
- **FlowPane** - lays out its child components either vertically or horizontally. Can wrap the components onto the next row or column if there is not enough space in a row/column.
- **GridPane** - displays UI elements in a grid (e.g., a grid of 2 rows by 2 columns)

VBox & HBox



VBox Example

- **VBox** layout pane creates an easy way for arranging child components in a *single vertical column*
 - Create a VBox layout container
 - Add 3 buttons to the VBox



```
/* Within App class */  
@Override  
public void start(Stage stage) {  
    //code for setting root, stage, scene ...  
}
```

```
VBox root = new VBox();
```

```
Button b1 = new Button("Button 1");  
Button b2 = new Button("Button 2");  
Button b3 = new Button("Button 3");  
root.getChildren().addAll(b1,b2,b3);
```

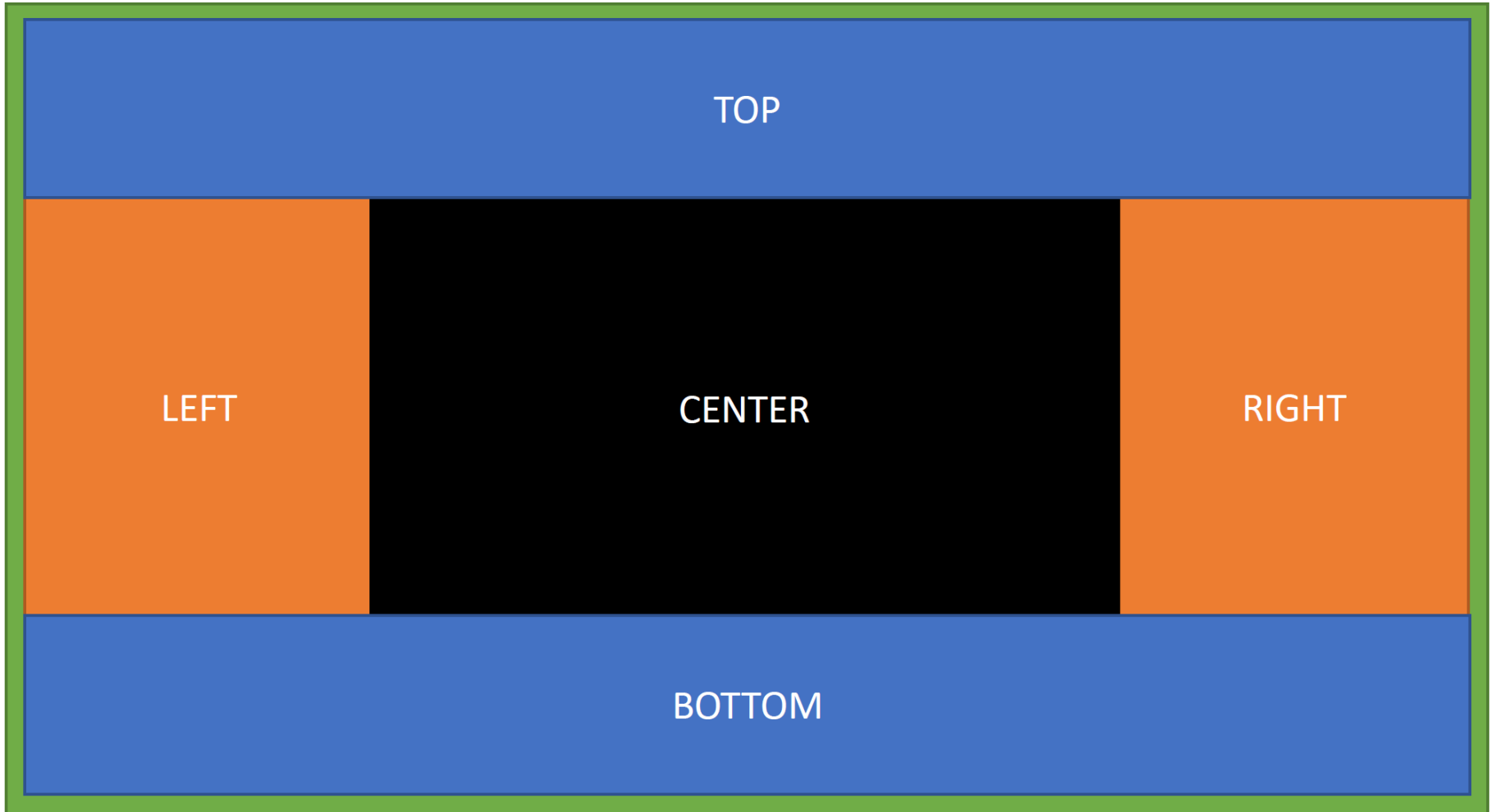
```
}
```

Order matters - order
buttons added effects
order displayed
(b1, b2, b3) vs. (b2, b1, b3)

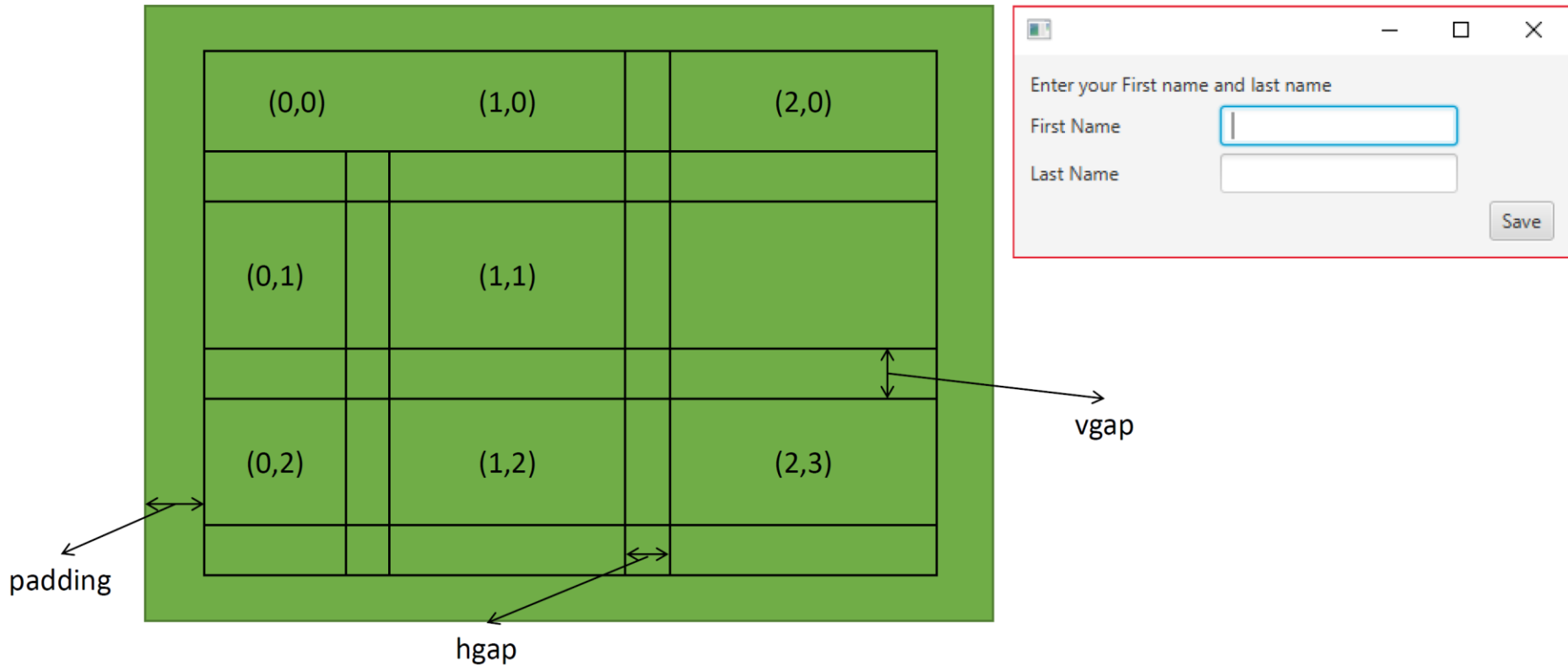
Customizing VBox layout

- We can customize vertical spacing *between* children using VBox's `setSpacing(double)` method
- Can also set positioning of child components
 - Default positioning is in `TOP_LEFT` (Top Vertically, Left Horizontally)
 - Can change Vertical/Horizontal positioning of column using VBox's `setAlignment(Pos position)` method
 - e.g. `Pos.BOTTOM_RIGHT` represents positioning on the bottom vertically, right horizontally
 - full list of `Pos` constants can be found [here](#)

BorderPane

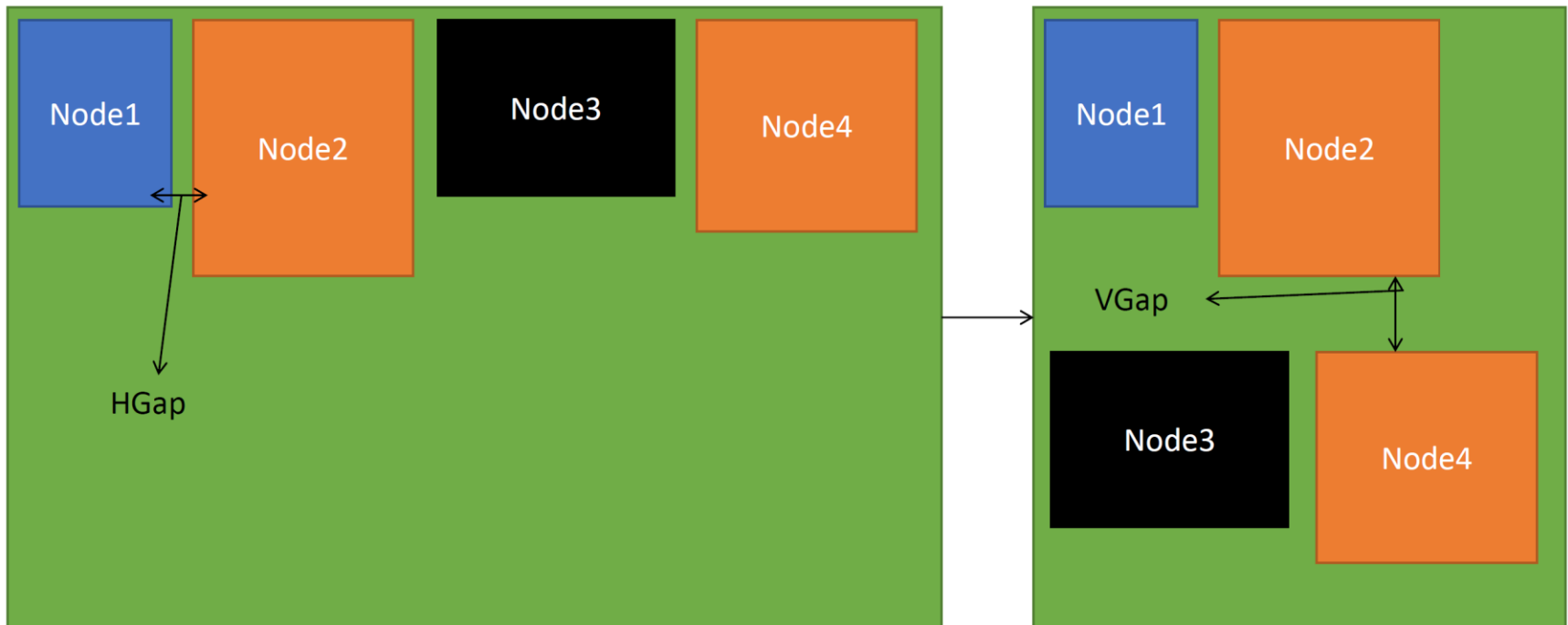


GridPane

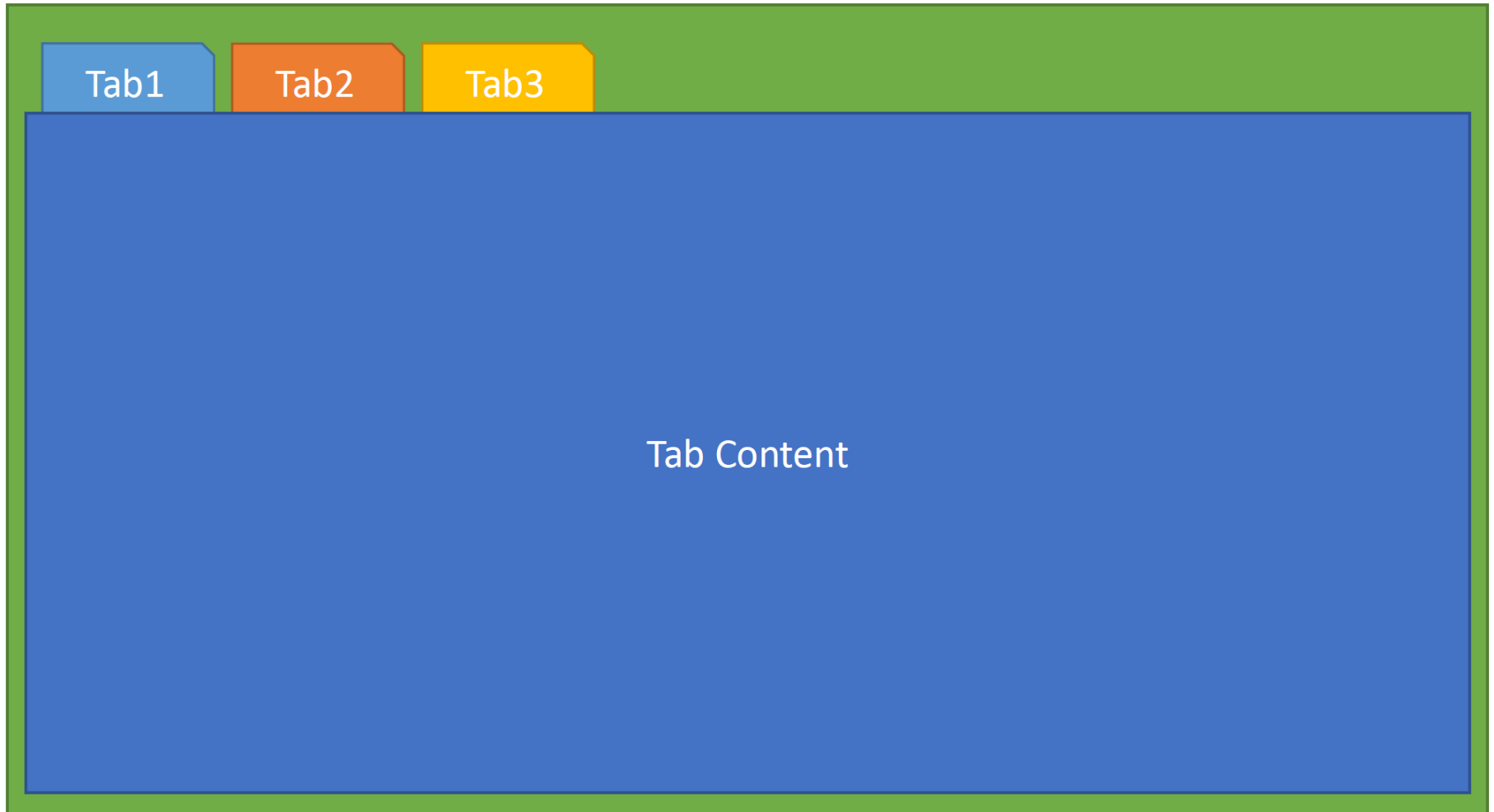


FlowPane

- With **FlowPane** the components are arranged from left to right and top to bottom manner in the order they were added

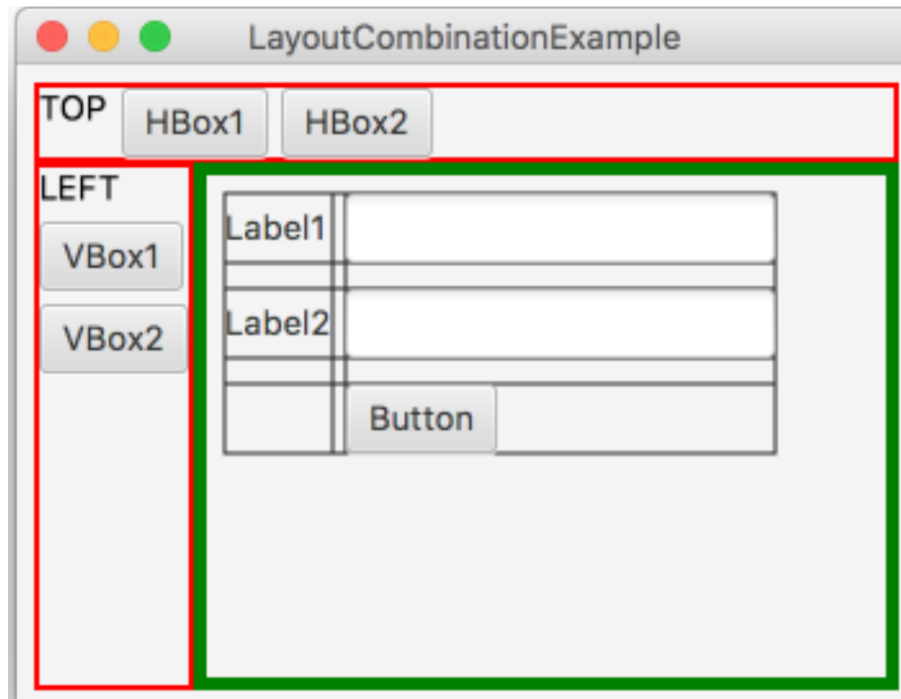


TabPane



Complex Layouts

- For more complex views you can combine different layouts to group components
 - e.g., a BorderPane that contains VBox and HBox panes

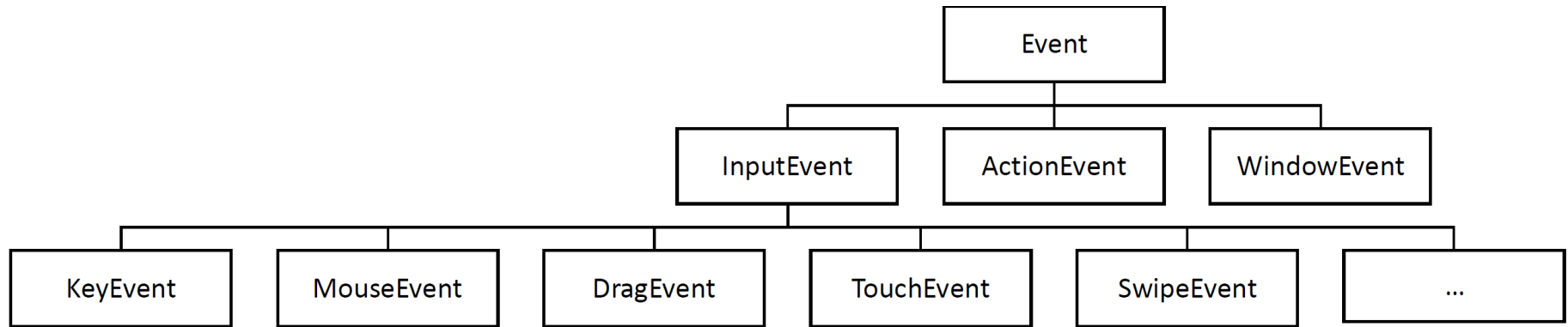


Handling Events

What is Event Driven Programming?

- GUI programming model is based on **event driven programming**
- An **event** is a signal that some something of interest to the application has occurred
 - Keyboard (key press, key release)
 - Mouse Events (clicked, mouse enters, mouse leaves)
 - Input focus (gained, lost)
 - Window events (starting, closing, maximize, minimize)
- When an event is triggered, an event handler can run to respond to the event. e.g.,
 - When the button is clicked -> load the data from a file into a list
 - When a mouse is moved over a button -> show a tooltip

Event Hierarchy



Handling Events using Lambdas



```
btn.setOnMouseClicked(event ->
    handleMouseEvent(event));
```

// Or use method reference

```
btn.setOnMouseClicked(
    this::handleMouseEvent);
```

```
private void handleMouseEvent(MouseEvent event) {
    System.out.println(event);
}
```