CMPS 251

Read Chapter 12 and 13

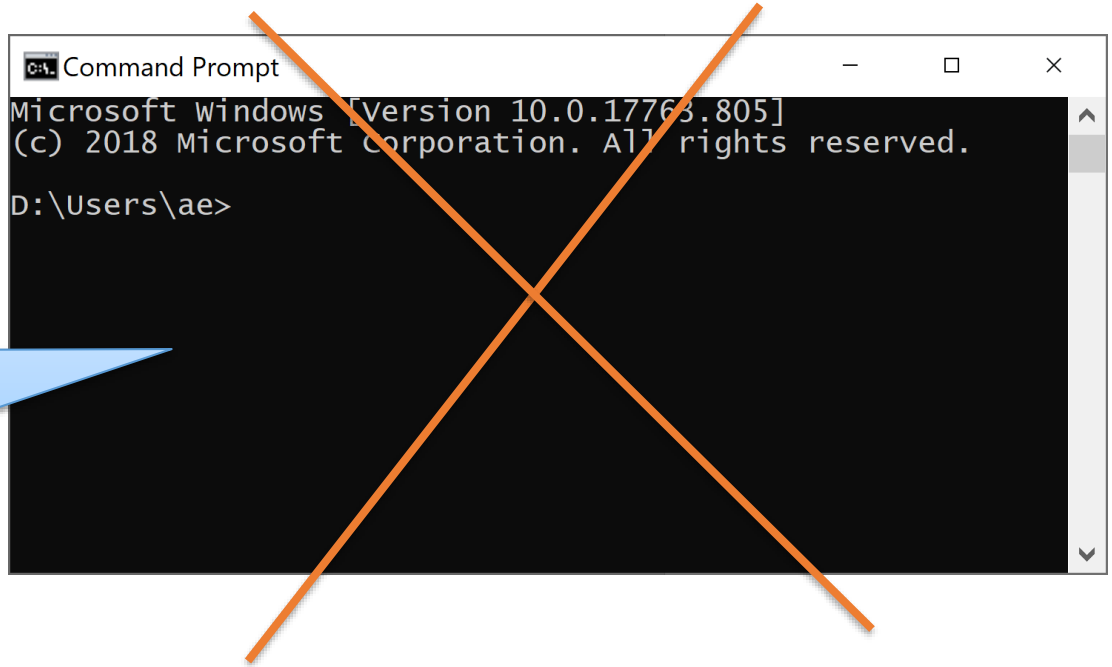
Graphical User Interfaces (GUI)

Dr. Abdelkarim Erradi
CSE@QU

Outline

- GUI Programming Model
- JavaFX Layout
- Handling Events
- Model-view-controller (MVC) Pattern
- Commonly used JavaFX UI Components
- Properties and Bindings

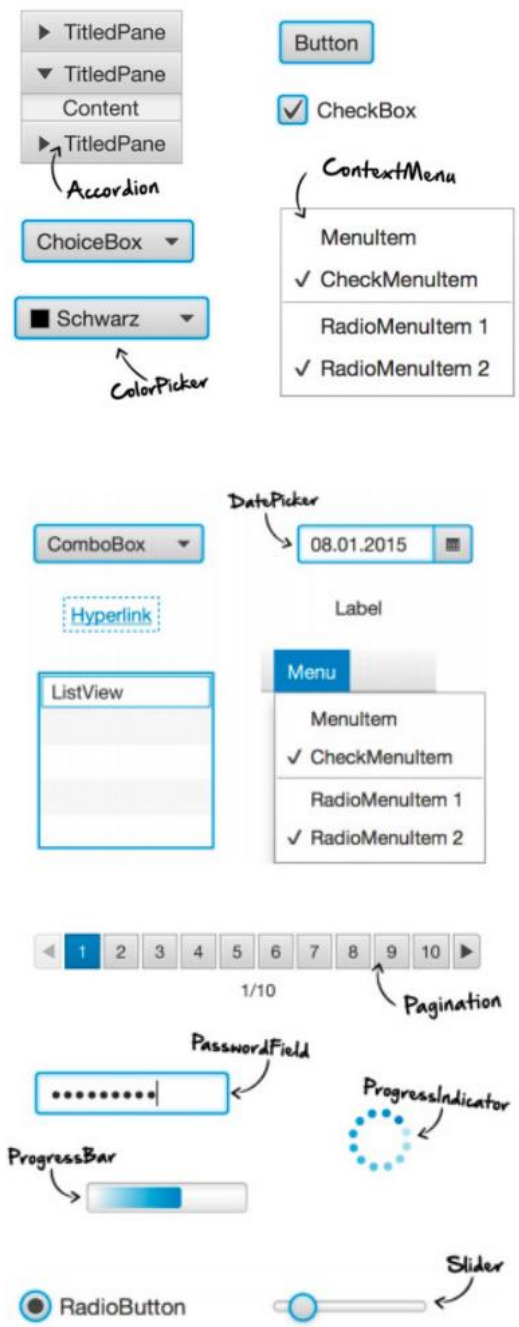
GUI Programming Model



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

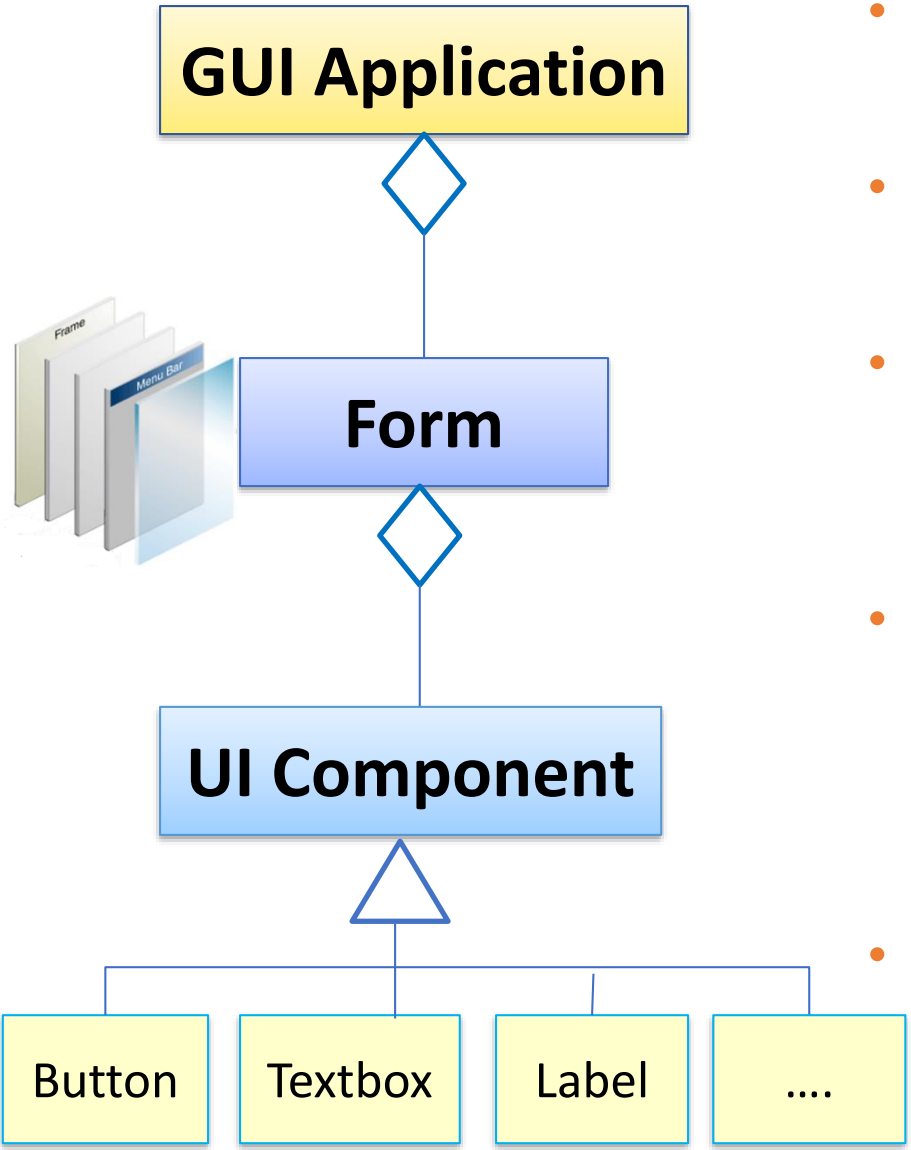
Back

What is a GUI?



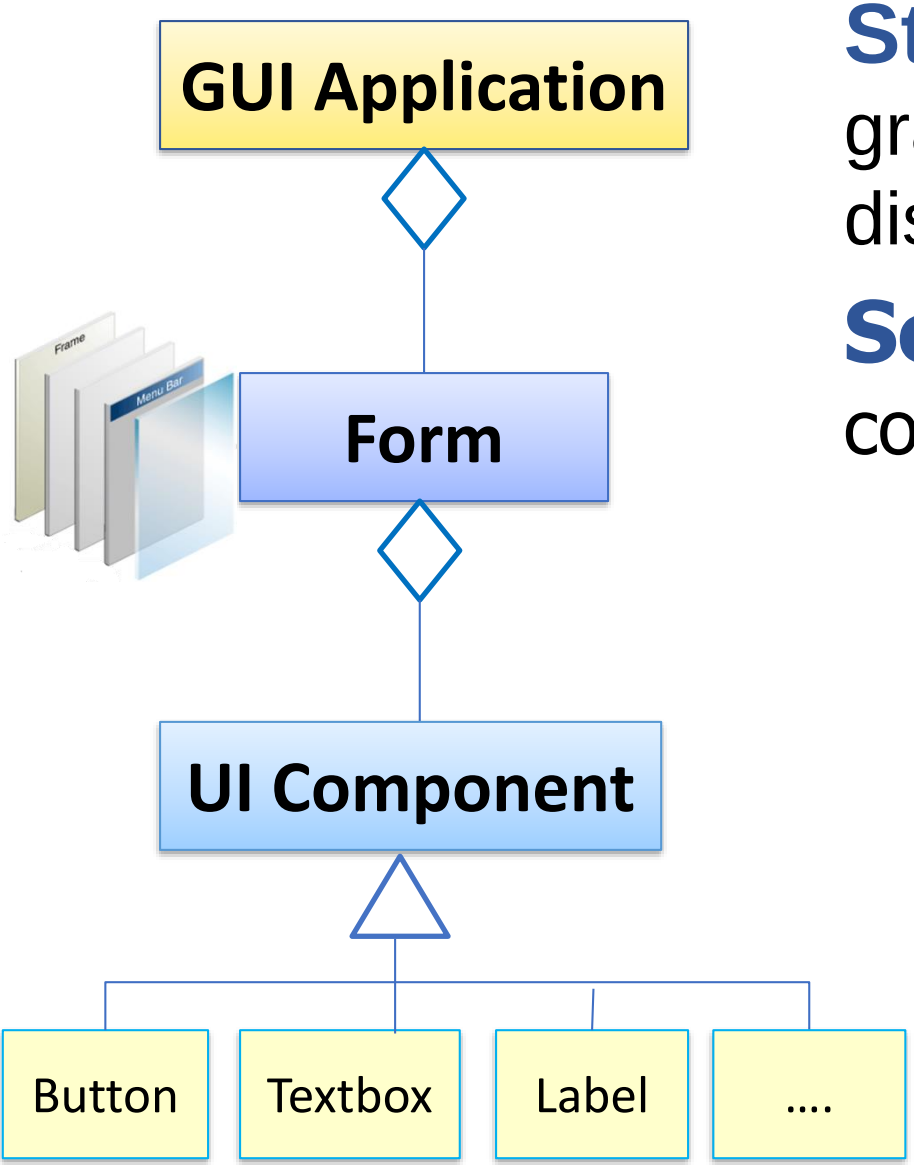
- **Graphical User Interface (GUI)** provides a visual User Interface (واجهة الاستخدام) for the users to interact with the application
  - Instead of a Character-based interface provided by the console interface 'the scary black screen'
- **JavaFX** can be used for creating GUI

GUI Programming Model



- GUI of an application is made up of **Forms** (JavaFX calls it **Stage**)
- Each form has **container** (called **Scene**) to place UI **Components**
- UI Components are typically placed in a 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

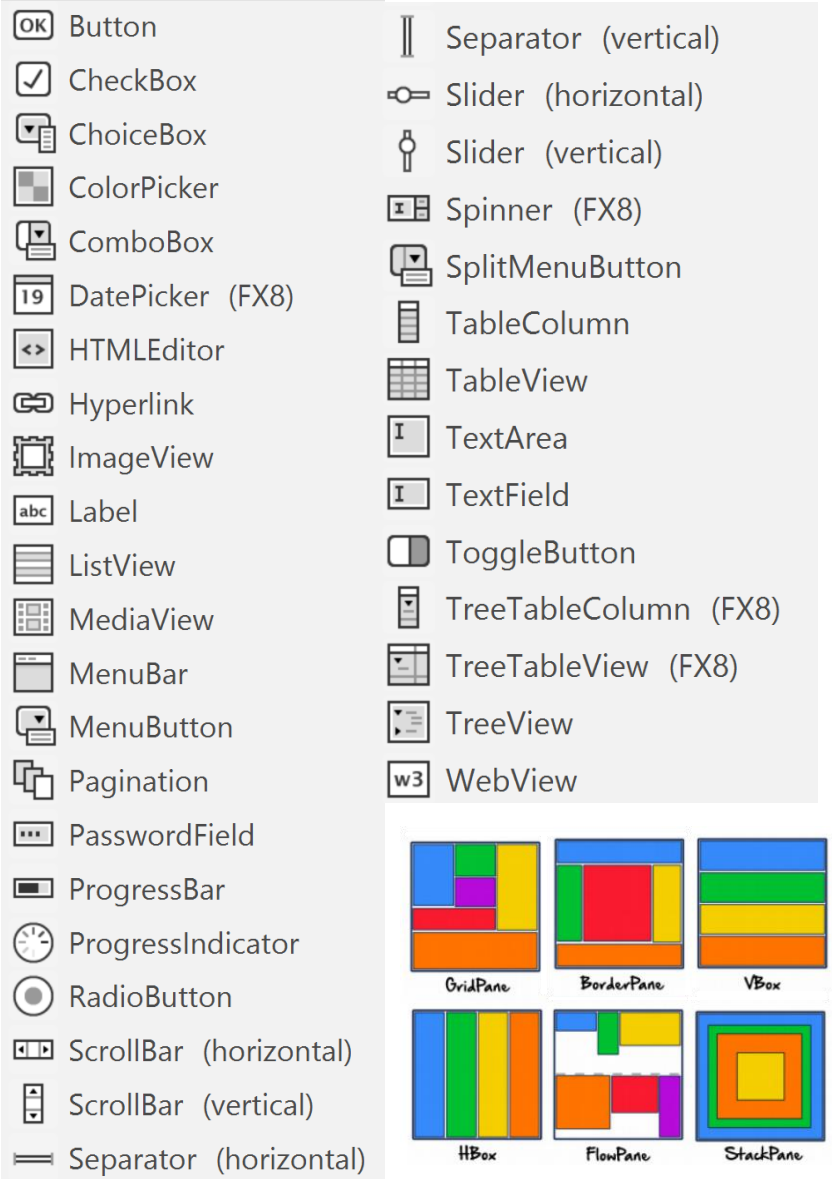
Structure of JavaFX application



Stage = **Form** where all graphic elements will be displayed

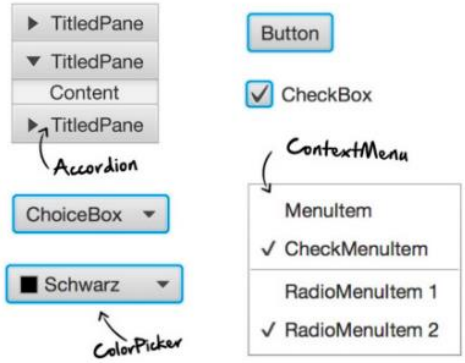
Scene = **Container** for all UI components to be displayed

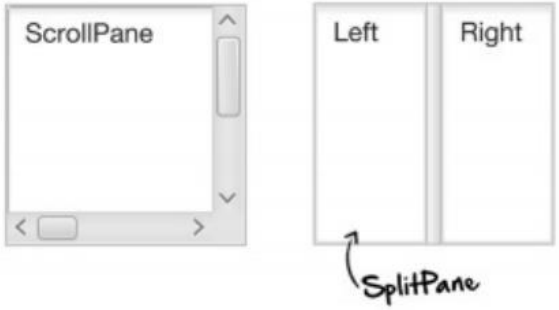
What Makes up JavaFx ?

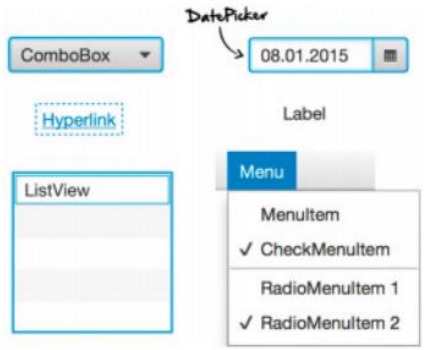


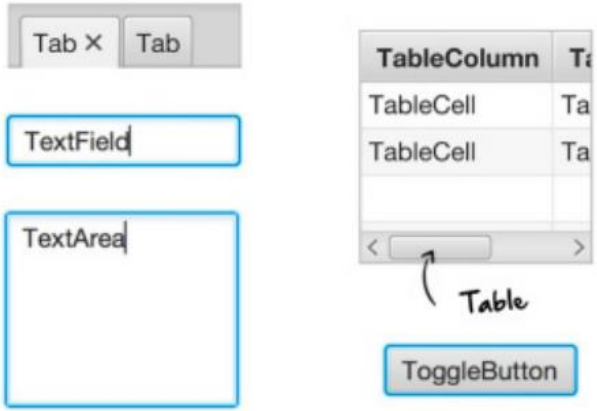
- **UI components**
  - Set of pre-built UI components that can be composed to create a GUI
  - e.g. buttons, text-fields, menus, tables, lists, etc.
- **Layout containers**
  - Control placement/ positioning of components in the form (e.g., VBox and HBox)

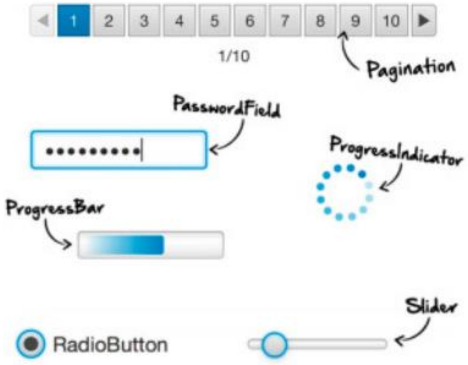
JavaFX UI Components

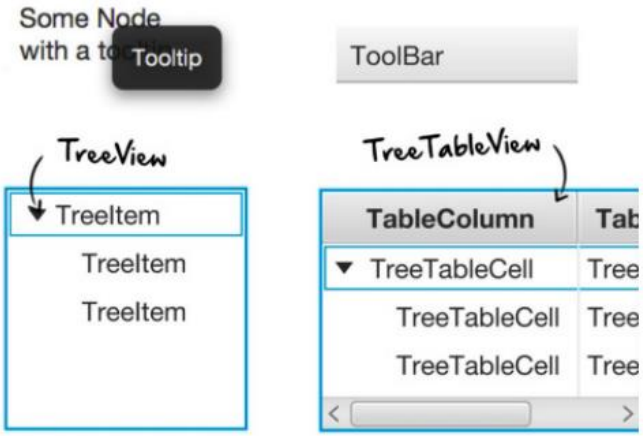




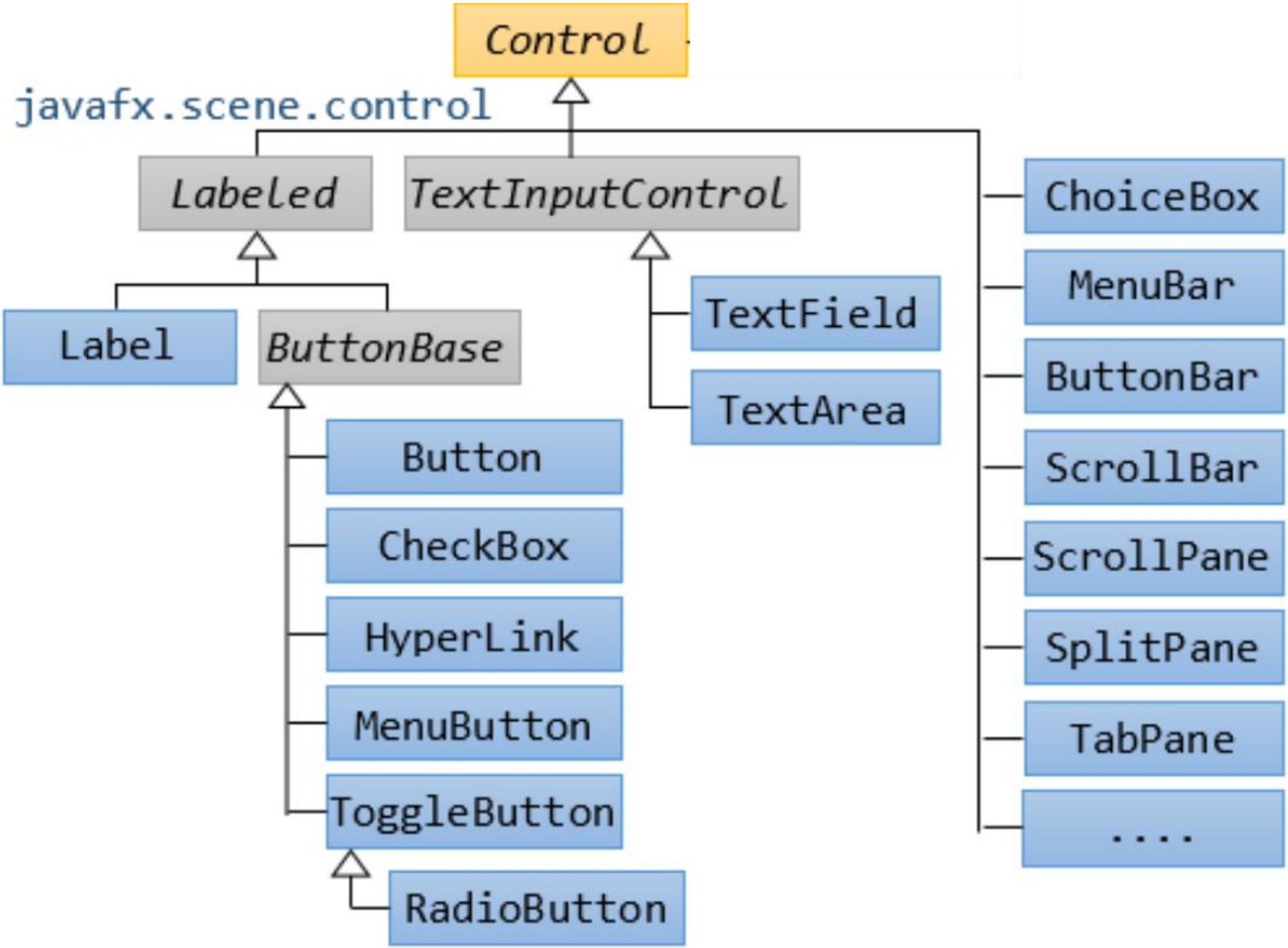








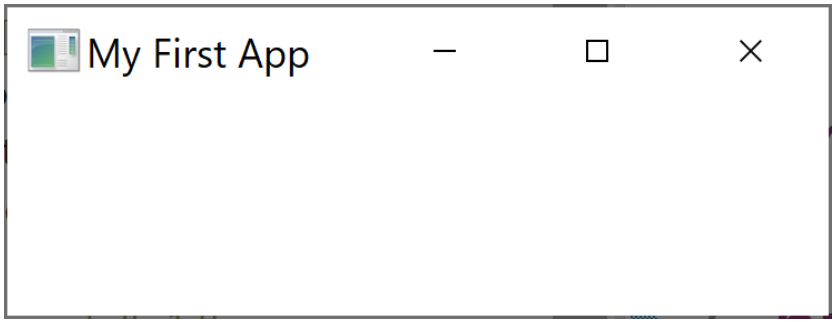
JavaFX UI Components



Creating JavaFX GUI: Stage (1/2)

- Create a class that extends `javafx.application.Application`
- Implement the `start(Stage stage)` method
  - `start()` is called when the app is launched
- JavaFX automatically creates an instance of **`Stage`** class which is passed into `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);
 }
}
```



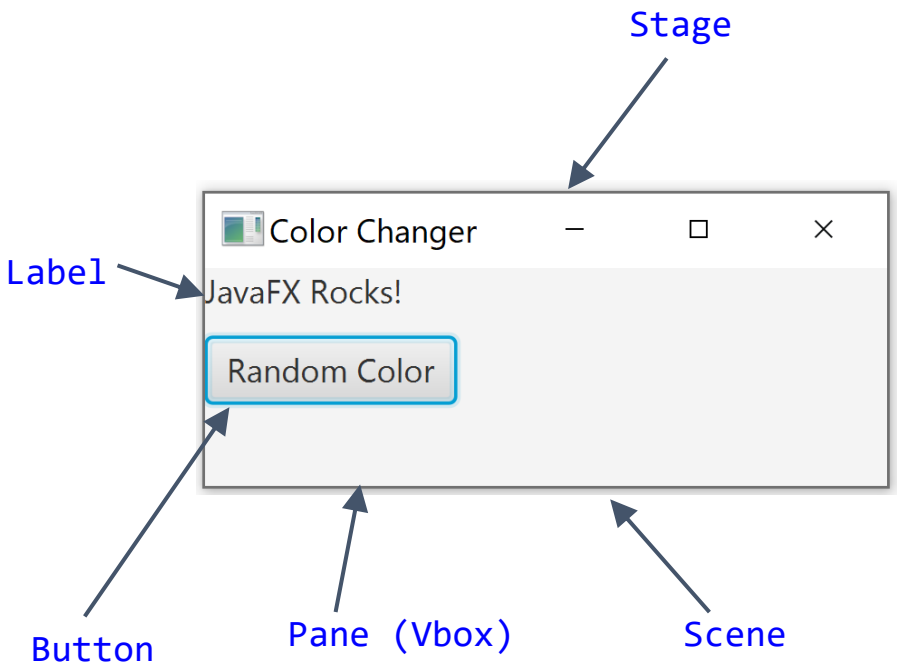
Creating JavaFX GUI : Scene (2/2)

- 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();
}
```

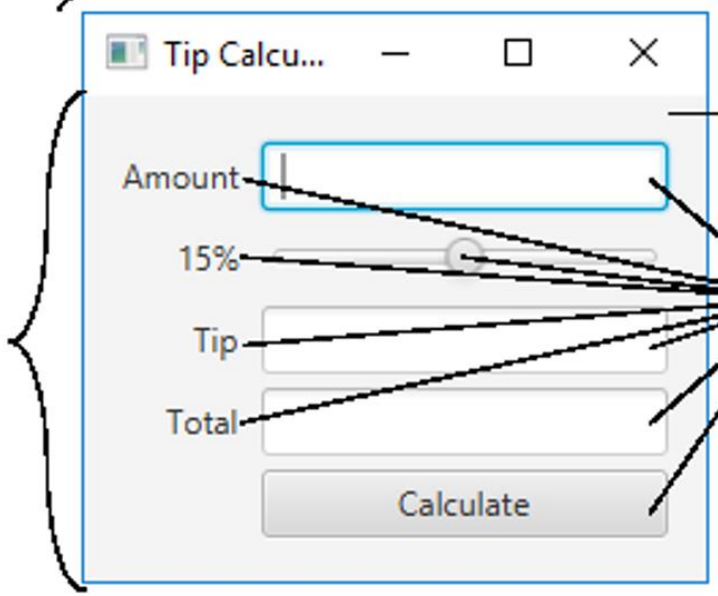
JavaFX Application: ColorChanger

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

The stage contains a scene graph of nodes

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

Label component

ImageView component

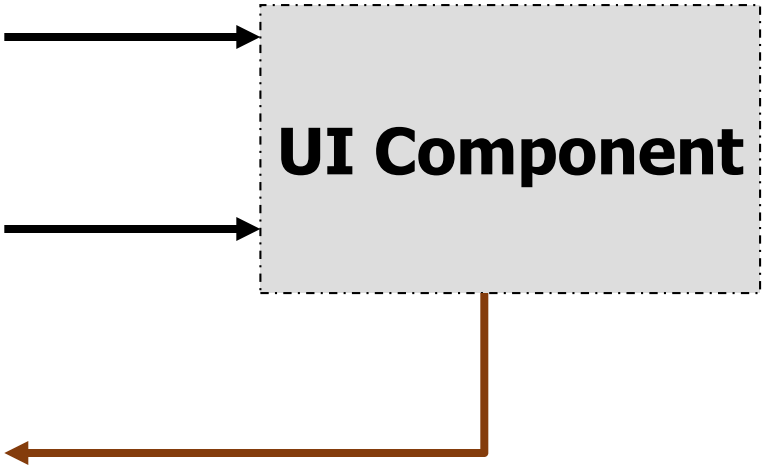
UI Component

- 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**

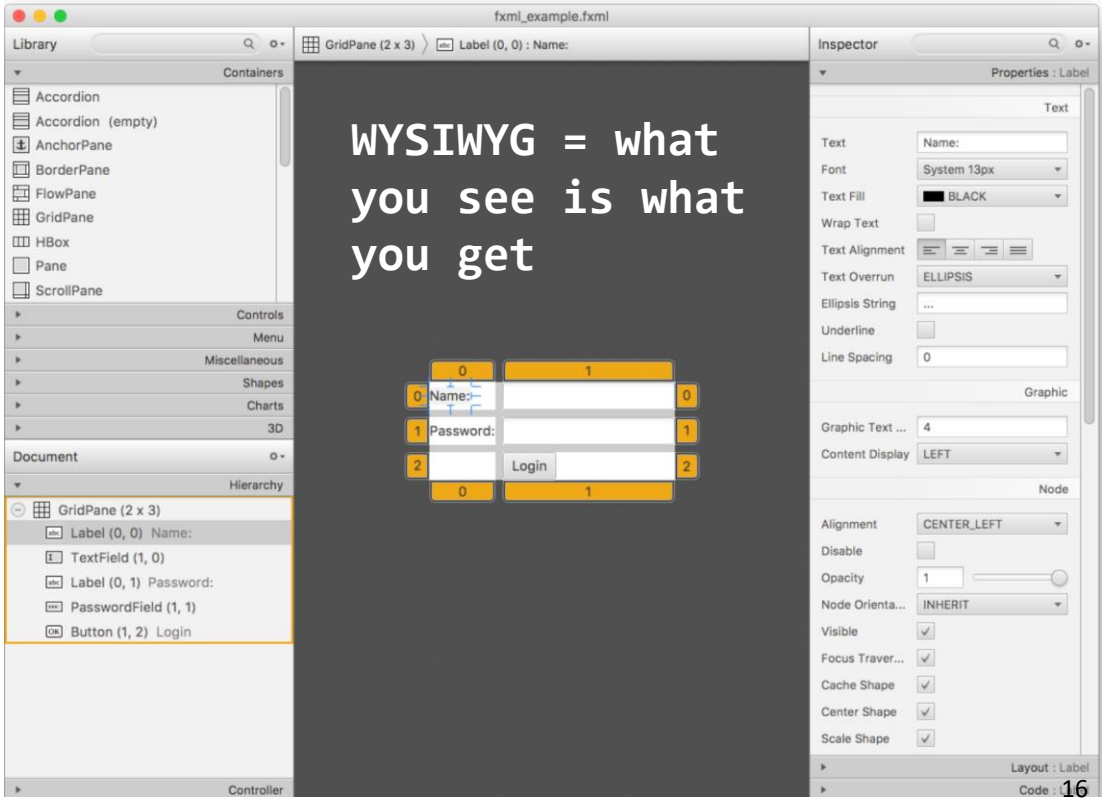
```
vBox.add(button);
```

4. **Listen to and handle its events**

```
// Register an event handler
button.setOnMouseClicked( event ->
      System.out.println(event) );
```

FXML

- You can create JavaFX UI using code or FXML
- FXML is an XML-based language that defines the **structure** and **layout** of JavaFX UI
- 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

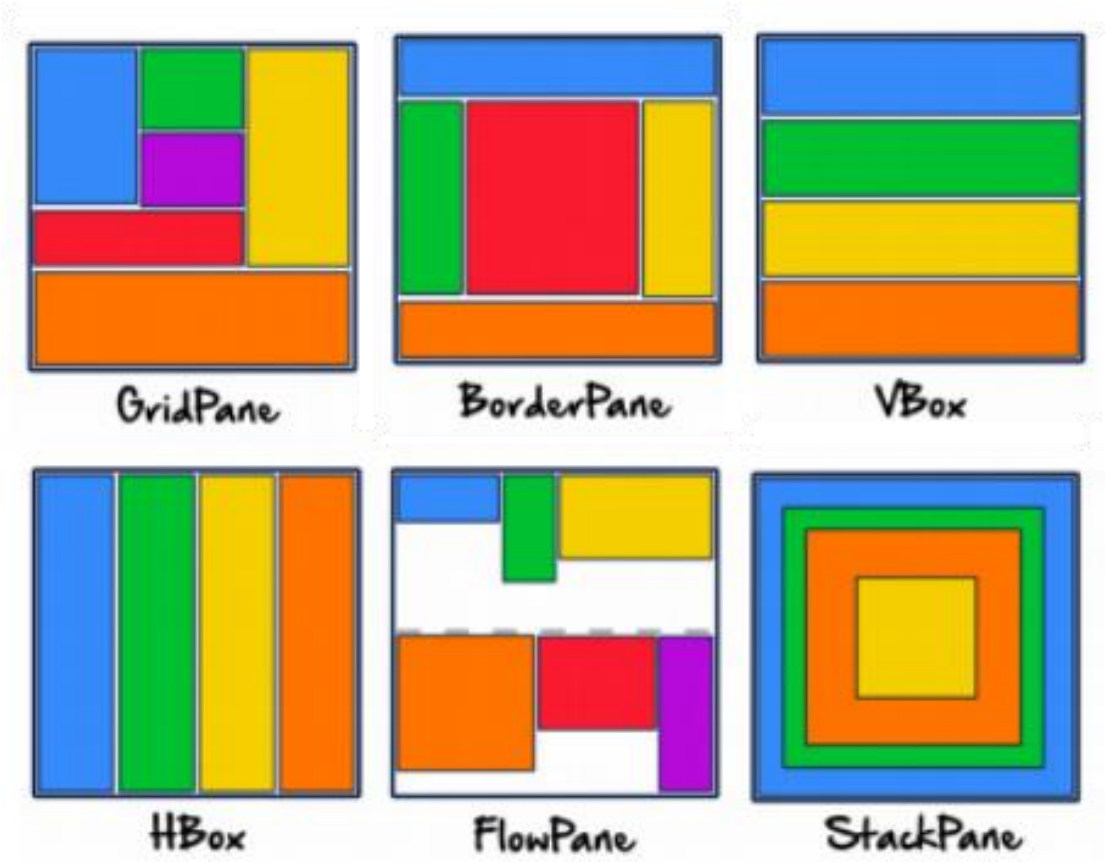
- FXML can be associated with a **Controller** class that implements all the events handlers

```
public class LoginController
{
  @FXML
  private Button loginBtn;

  @FXML
  void handleLogin(ActionEvent event)
  {
     System.out.println("Login pressed");
  }
}
```

Steps to creating a GUI Interface

1. Design it on paper
   - Decide what information to present to user and what input they should supply
2. Choose components and containers
   - Decide the components and layout on paper
3. Create a form and add components to it (SceneBuilder can be used)
   - Use layout panels to group and arrange components
4. 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.

Layouts

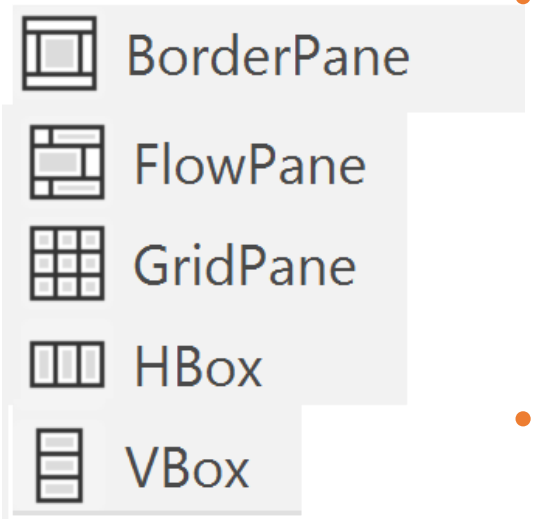


Back

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 positioning of UI elements
  - As the window is resized, the UI components reorganize themselves based on the rules of the layout

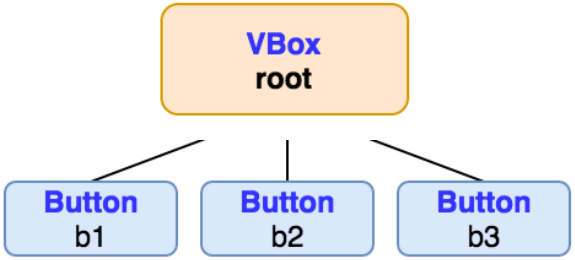
Common Layouts



- **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)
- **HBox** - displays UI elements in a horizontal line
- **VBox** - displays UI elements in a vertical line.

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);
    }
```

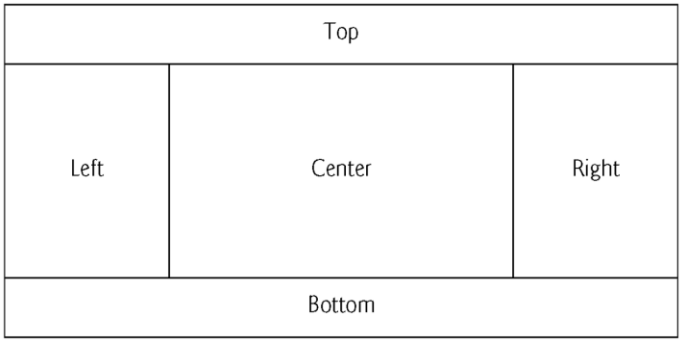
Order matters - order buttons added effects order displayed
(b1, b2, b3) vs. (b2, b1, b3)

Customizing VBox layout

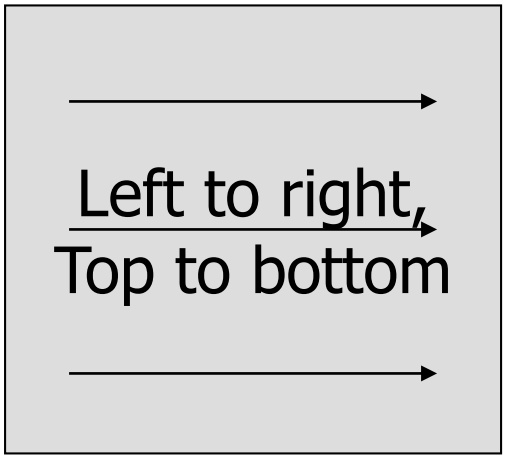
- We can customize vertical spacing *between* children using VBox's `setSpacing(double)` method
- Can also set positioning of child components
  - Default positioning is in `TOP_LEFT` (Top Vertically, Left Horizontally)
  - Can change Vertical/Horizontal positioning of column using VBox's `setAlignment(Pos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

Important Layout Managers

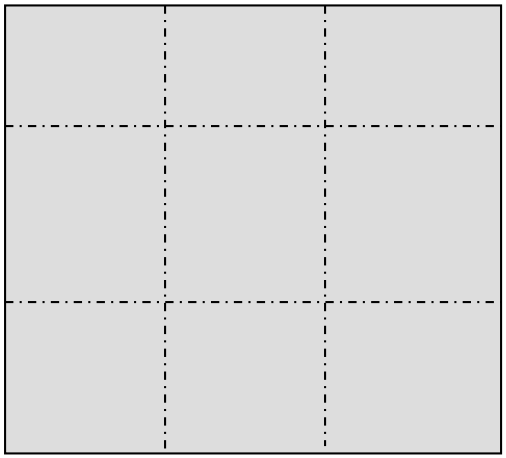
BorderPane



FlowPane



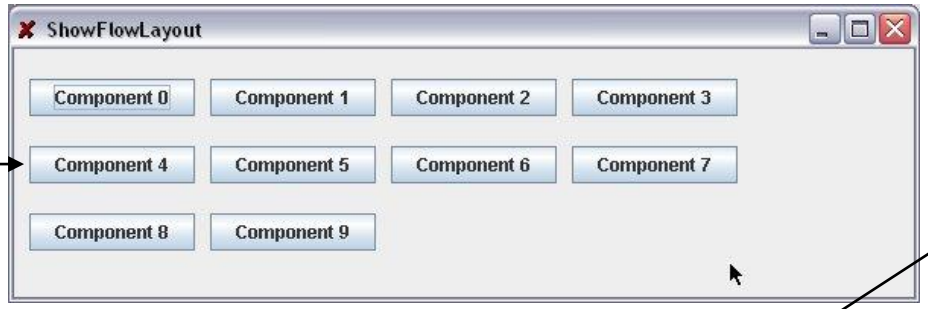
GridPane



FlowPane

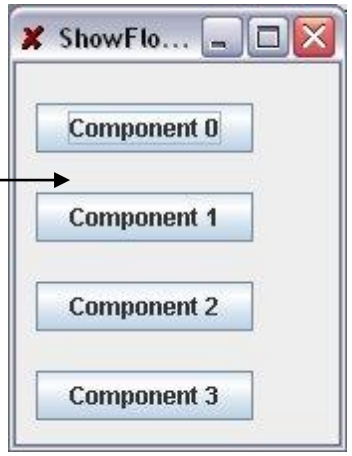
- With **flow pane**, the components arrange themselves from left to right and top to bottom manner in the order they were added

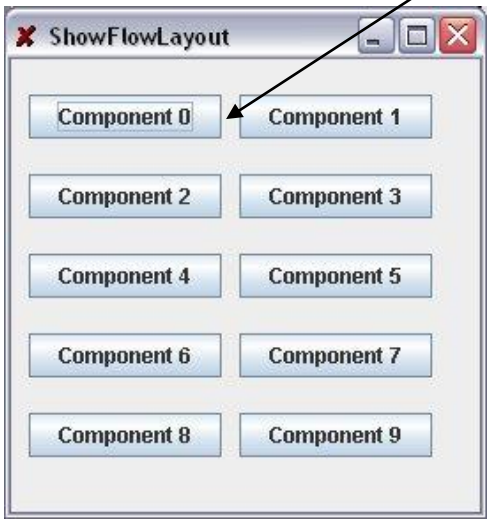
Rows/buttons are left aligned using `FlowPane.LEFT`



Horizontal gap of 10 pixels

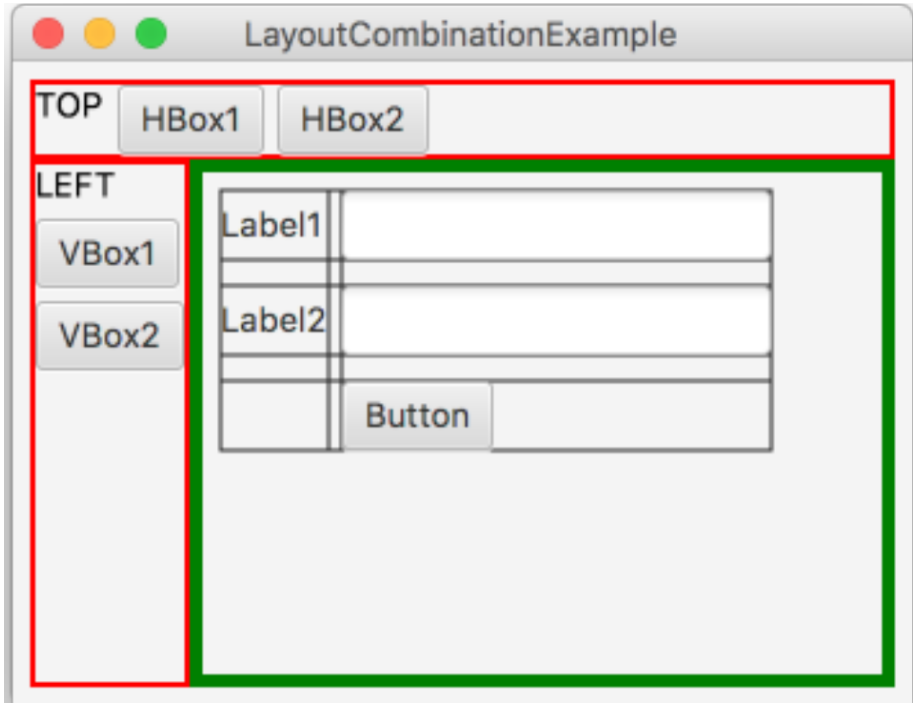
Vertical gap of 20 pixels





Complex Layouts

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



Handling Events

Back

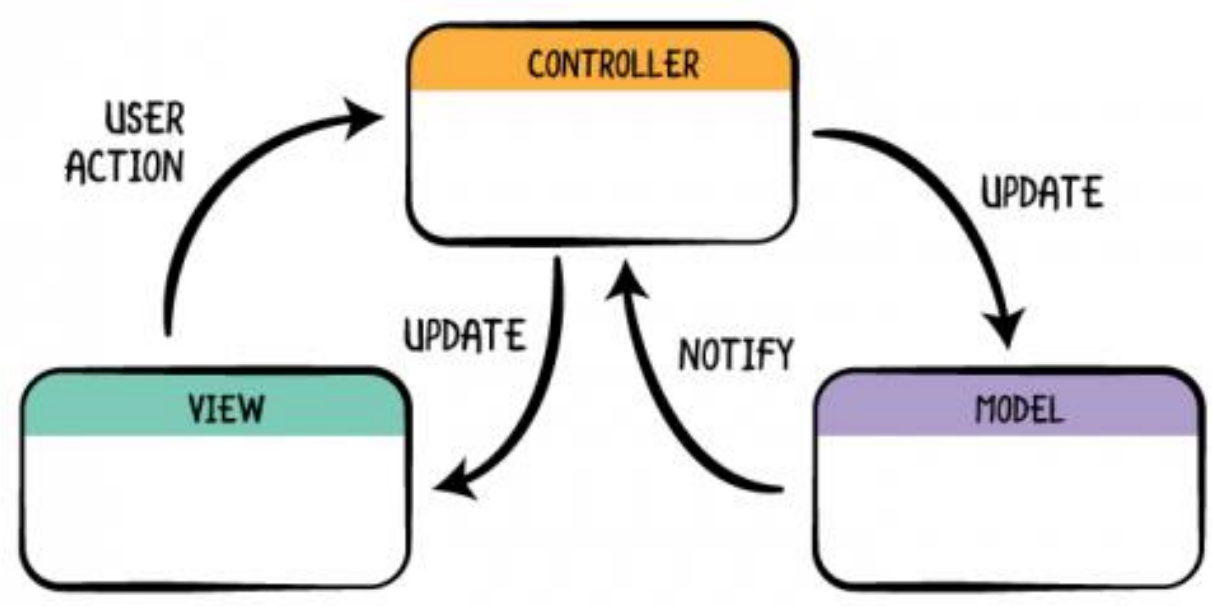
What is Event Driven Programming?

- GUI programming model is based on **event driven programming**
- An **event** is a signal to the program that some external action has occurred
  - Keyboard (key press, key release)
  - Pointer Events (button press, button release)
  - Mouse Events (mouse enters, leaves)
  - Input focus (gained, lost)
  - Window events (closing, maximize, minimize)
  - Timer events
- 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

Handling Events using Lambdas


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

// Or use method reference
btn.setOnMouseClicked(this::handleMou
seEvent);

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

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



Back

MVC-based Application

View

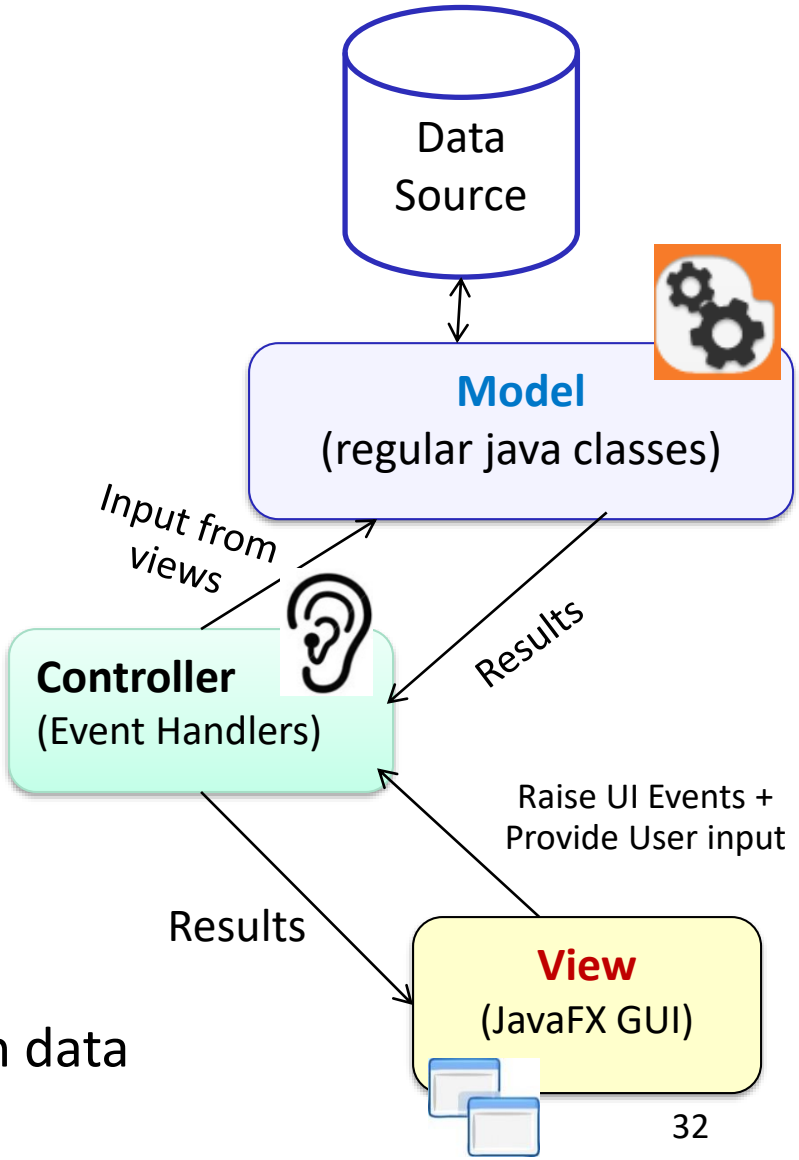
- Gets input from the user
- Notifies the controller about UI events
- Displays output to the user

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. 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

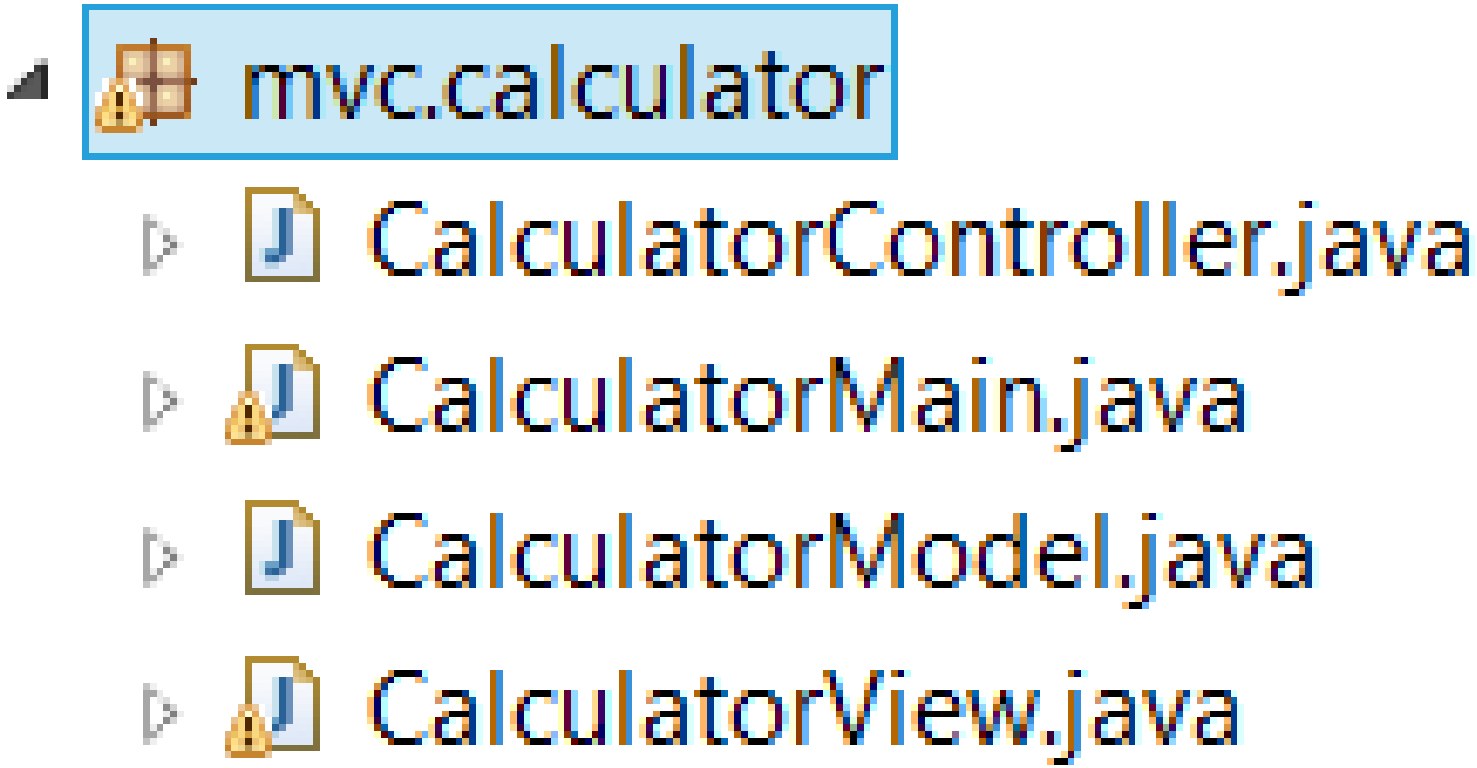
3. Build the **view** using *JavaFX Components* to collect input from the user and displays the results received from the controller

Advantages of MVC

- ***Separation of concerns***
  - Views, controller, and model are **separate components**. This allows modification and change in each component without significantly disturbing the other.
    - Computation is not intermixed with Presentation. Consequently, code is cleaner and easier to understand and change.
- **Flexibility**
  - The UI can be completely changed without touching the model
- **Reusability**
  - The same model can 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

Example


mvc.calculator
CalculatorController.java
CalculatorMain.java
CalculatorModel.java
CalculatorView.java

Commonly used JavaFX UI Components

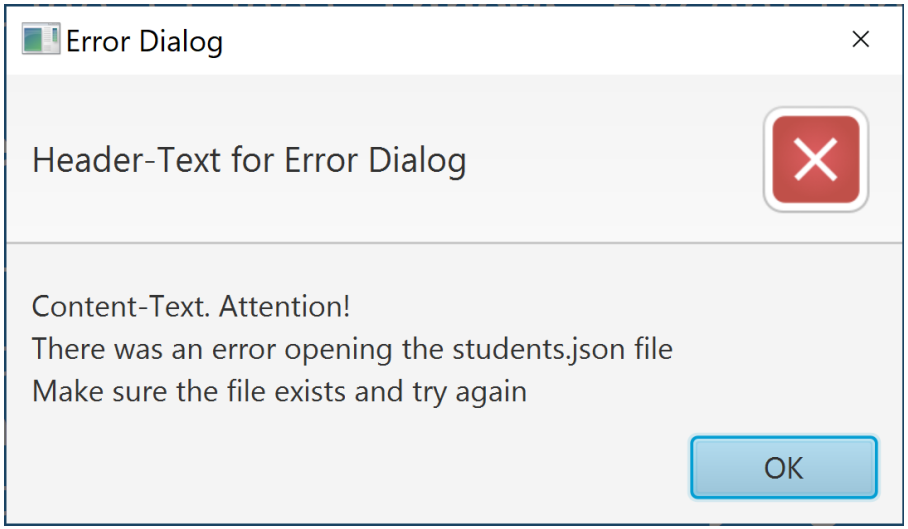
Back

Commonly used JavaFX UI Components

- Label, Button, RadioButton, ToggleButton
- CheckBox, ChoiceBox
- TextField, PasswordField, TextArea, Hyperlink
- ListView, TableView
- MenuBar, MenuButton, ContextMenu, ToolBar
- Tooltip, Separator
- ImageView, Audio Player, Video Player

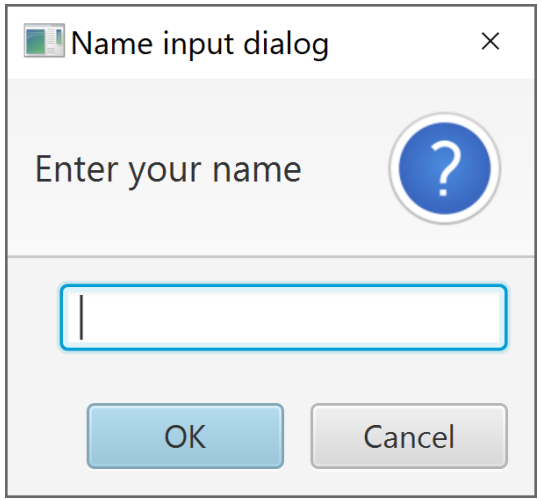
Info/Warn/Error Dialog

```
public void start(Stage stage) throws Exception
{
 Alert alert = new Alert(AlertType.ERROR);
 alert.setTitle("Error Dialog");
 alert.setHeaderText("Header-Text for Error Dialog");
 alert.setContentText("Content-Text. Attention!\n" +
  "There was an error opening the students.json   file\n" +
  "Make sure the file exists and try again");
 alert.showAndWait();
}
```



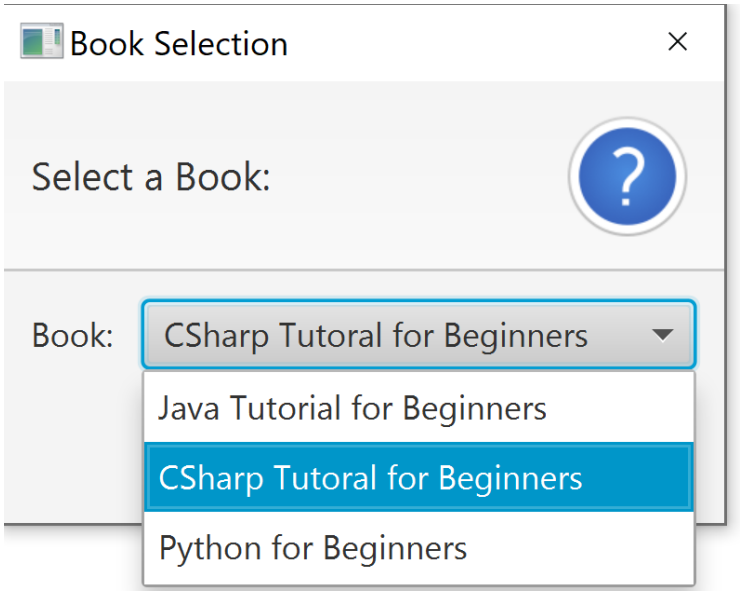
Input Dialog

```
public void start(Stage stage) throws Exception
{
 TextInputDialog dialog = new TextInputDialog();
 dialog.setTitle("Name input dialog");
 dialog.setHeaderText("Enter your name");
 Optional<String> result = dialog.showAndWait();
 result.ifPresent(name ->
      System.out.println("Your name: " + name));
}
```



Choice Dialog

```
List<Book> books = List.of(java, csharp, python);
Book defaultBook = csharp;
ChoiceDialog<Book> dialog = new ChoiceDialog<Book>(defaultBook, books);
dialog.setTitle("Book Selection");
dialog.setHeaderText("Select a Book:");
dialog.setContentText("Book:");
Optional<Book> result = dialog.showAndWait();
result.ifPresent(book -> System.out.println(book.getName()) );
```



Properties and Bindings

Back

Property Binding

- Property binding enables propagating changes
- The target listens for changes in the source and updates itself when the source changes
- Binding syntax: `target.bind(source);`

```
IntegerProperty num1 = new SimpleIntegerProperty(1);
IntegerProperty num2 = new SimpleIntegerProperty(2);
num1.bind(num2);
System.out.println("num1 is " + num1.getValue()
            + " and num2 is " + num2.getValue());

num2.setValue(15); //Changing num2 will auto-update num1

System.out.println("num1 is " + num1.getValue()
            + " and num2 is " + num2.getValue());
```

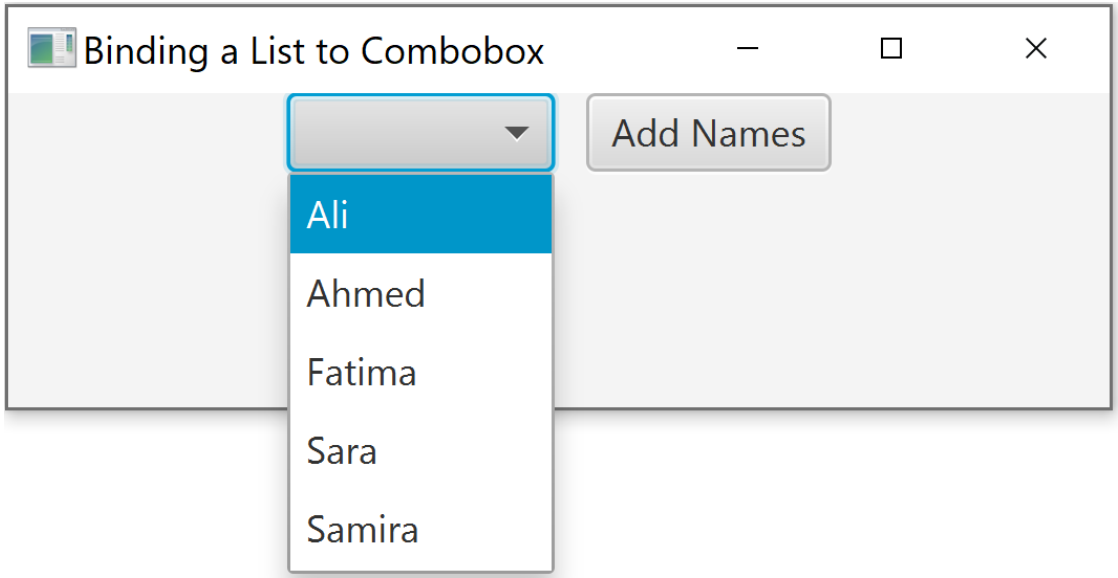
```
num1 is 2 and num2 is 2
num1 is 15 and num2 is 15
```

Property Binding

- A pair of simple integer property objects (not int values) are created with different values. Then one is **bound** to the other
- Their values are printed out (showing that they are different). If the value of one is changed then the other will also be changed.

ObservableList + ComboBox

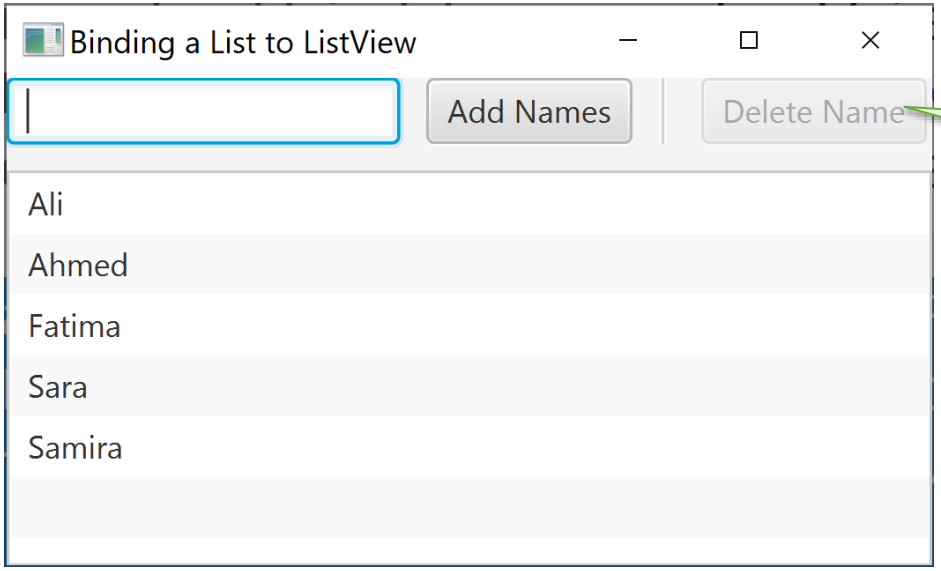
```
final String[] names = { "Ali", "Ahmed", "Fatima", "Sara", "Samira" };
final ObservableList<String> entries =
      FXCollections.observableArrayList(names);
final ComboBox<String> namesCombo = new ComboBox<>(entries);
final Button addNameBtn = new Button("Add Names");

addNameBtn.setOnAction(event -> {
      entries.addAll("Abbas", "Farid");
      namesCombo.show();
});
```



Observable = receive notification of changes

ObservableList + EditableListView

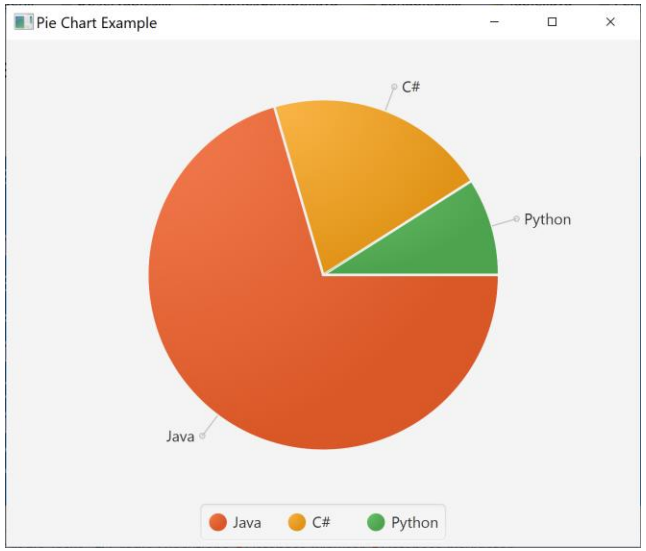
```
String[] names = { "Ali", "Ahmed", "Fatima", "Sara", "Samira" };
ObservableList<String> entries =
        FXCollections.observableArrayList(names);
ListView<String> listView = new ListView<>(entries);

SelectionModel<String> selectionModel = listView.getSelectionModel();
deleteBtn.disableProperty().bind(Bindings.isNull(
                    selectionModel.selectedItemProperty()));
```



When no name is selected the delete button is disabled

Pie Chart

```
public void start(Stage stage) throws Exception
{
 PieChart pieChart = new PieChart();
 pieChart.setData(createChartData());
 VBox root = new VBox();
 root.getChildren().add(pieChart);
 stage.setScene(new Scene(root));
 stage.setTitle("Pie Chart Example");
 stage.show();
}
```




```
private ObservableList<Data> createChartData() {
 ObservableList<Data> data =FXCollections.observableArrayList();
 data.add(new Data("Java", 70.5));
 data.add(new Data("C#", 20.5));
 data.add(new Data("Python", 9));
 return data;
}
```

Summary

- JavaFX provides a set of UI components to ease building GUI applications.
- The key expected learning outcome is gaining a good understanding and some hands on experience with:
  - UI components
  - Layout panes
  - UI event handlers
  - Building GUI Applications using the Model-view-controller (MVC) Pattern
  - Properties and Bindings