CMPS 251

Read Chapter 12 and 13

Graphical User Interfaces (GUI)

Dr. Abdelkarim Erradi

CSE@QU

Outline

- GUI Programming Model
- JavaFX Basics: Containers, Components
- Layout Managers
- Event Driven Programming
- Building GUI Applications using MVC
- Commonly used UI Components

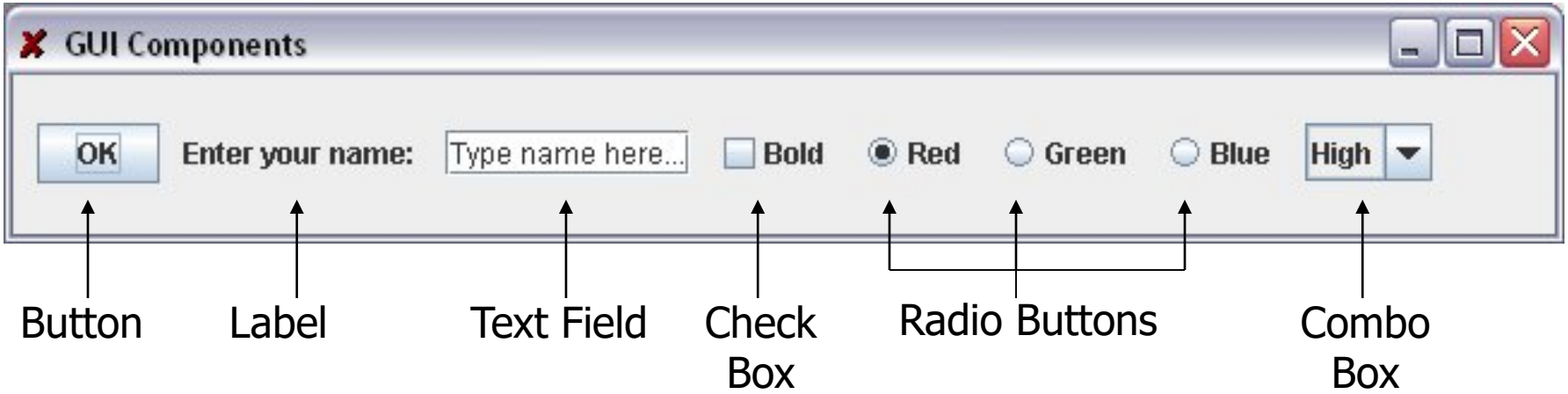
GUI Programming Model

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

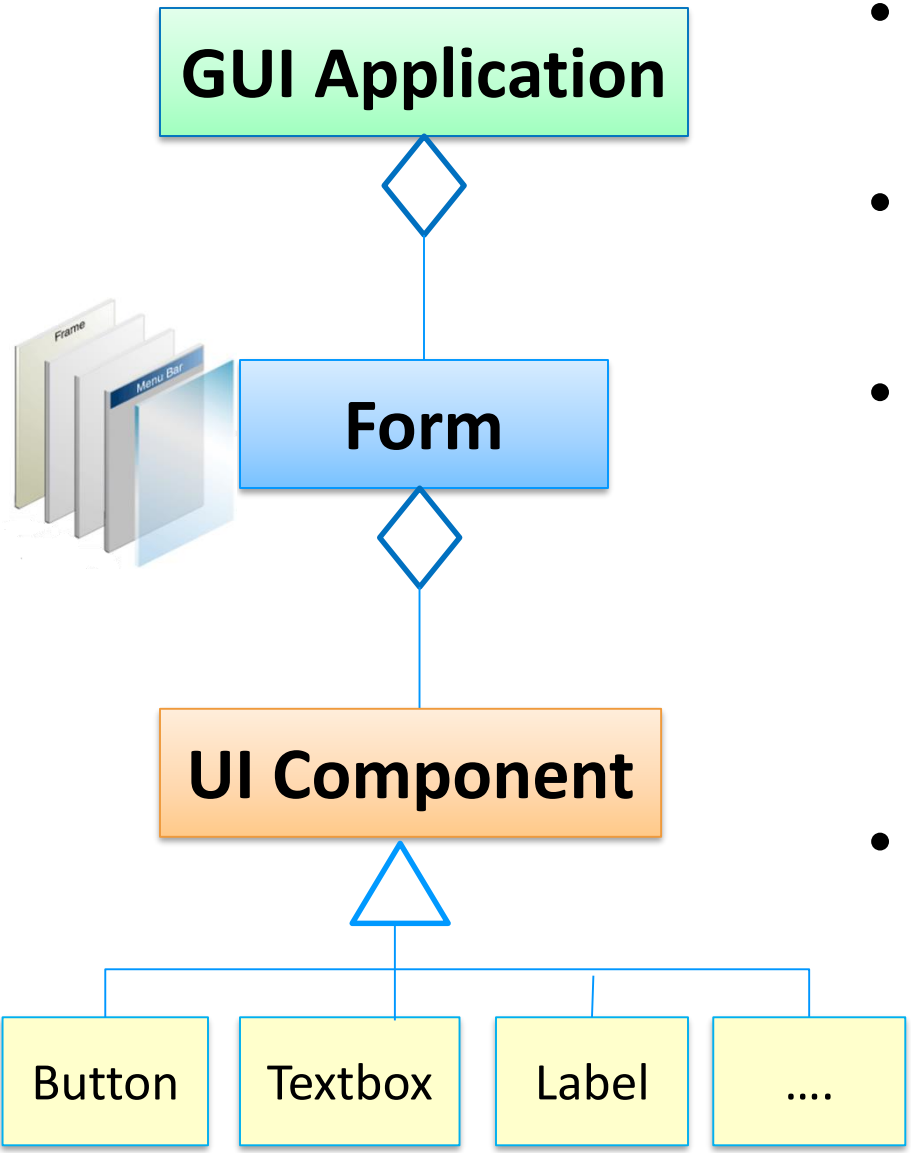
Back

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 'scary black screen' 
- Java has **standard packages** (called **JavaFX**) for creating GUI
- Some of the fundamental GUI components:

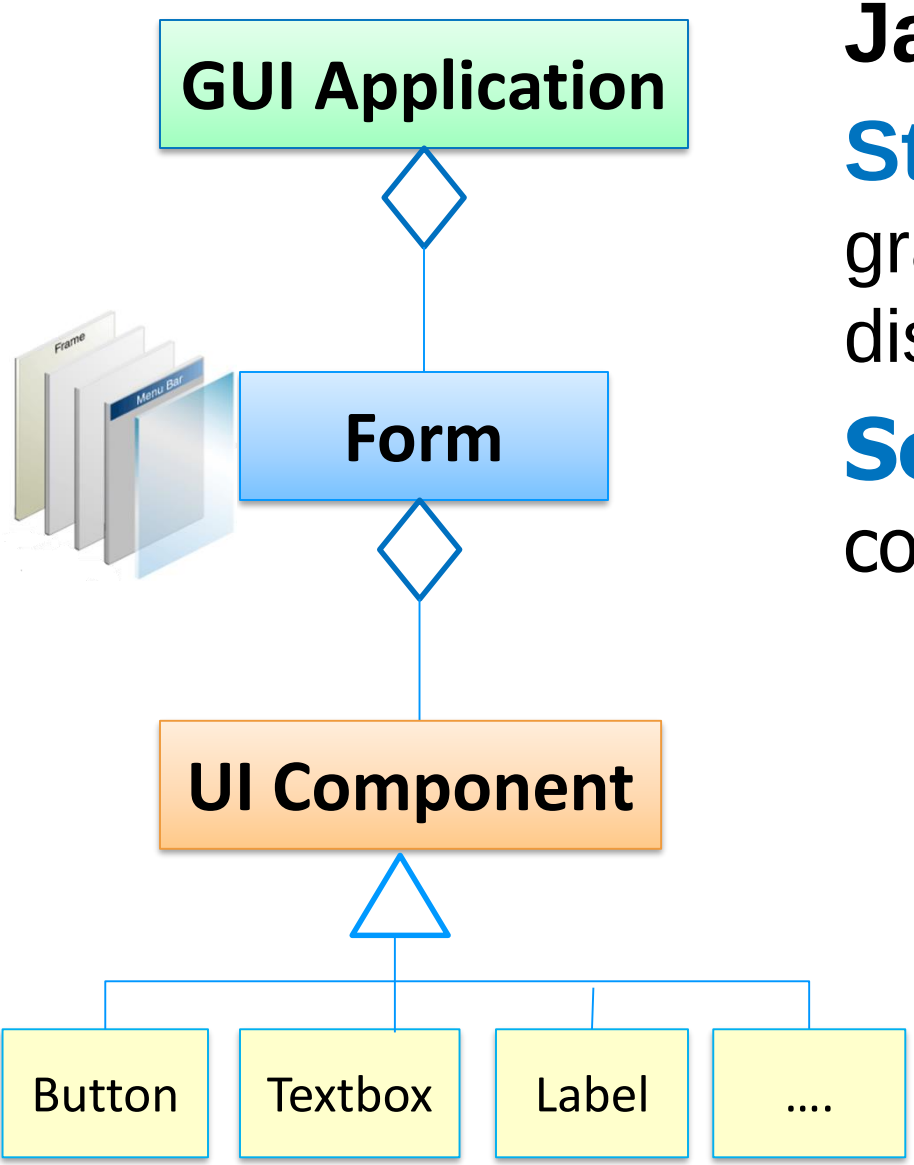


GUI Programming Model



- GUI of an application is made up of **Forms** (Stage)
- Each form has **container** (Scene) to place UI **Components**
- UI Components **raise Events** when the user interacts with them (such as a mouseClickEvent is raised when a button is clicked).
- Programmers write **Event Handlers** to respond to the UI events

Structure of JavaFX application

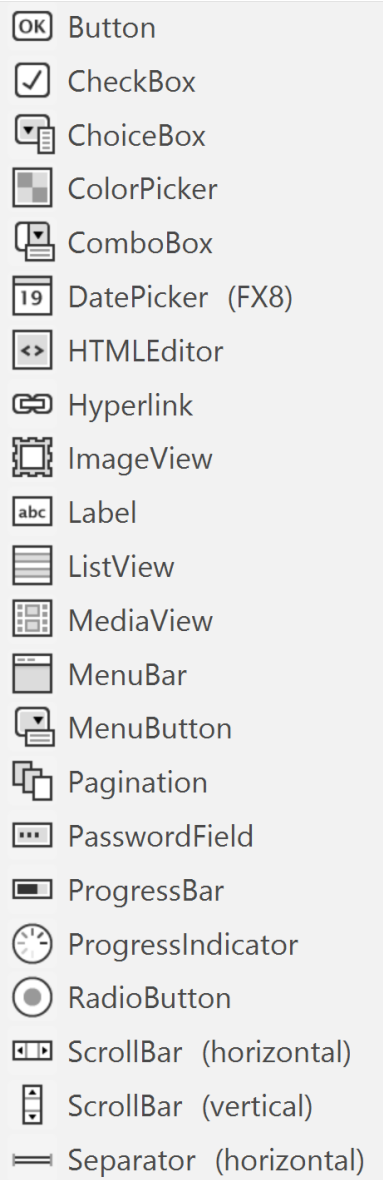


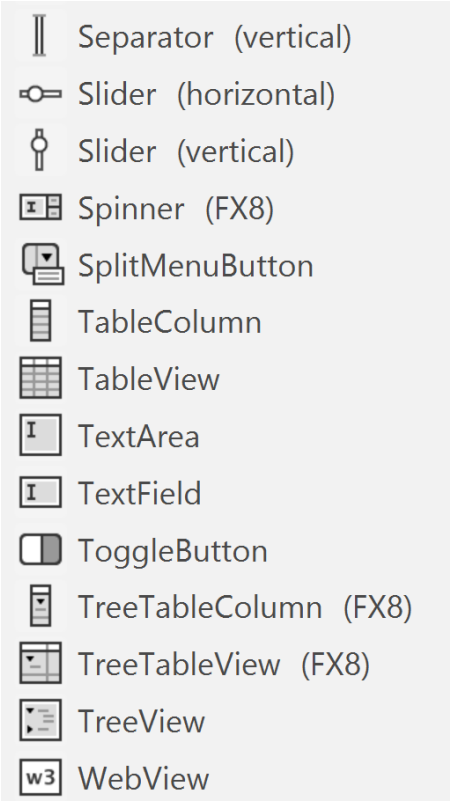
JavaFX terminology:

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

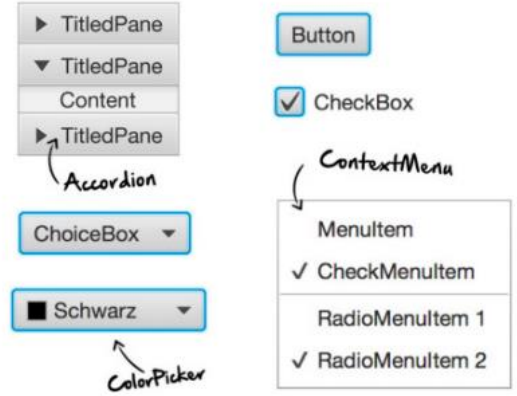
Scene = **container** for all UI components to be displayed

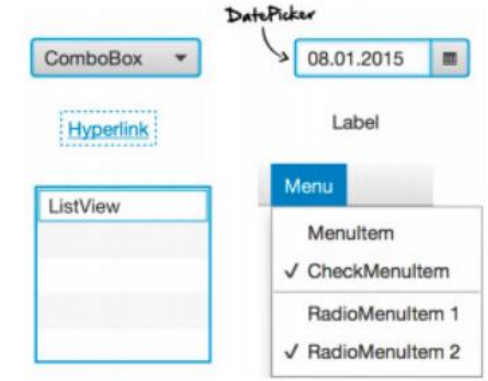
What is JavaFX?

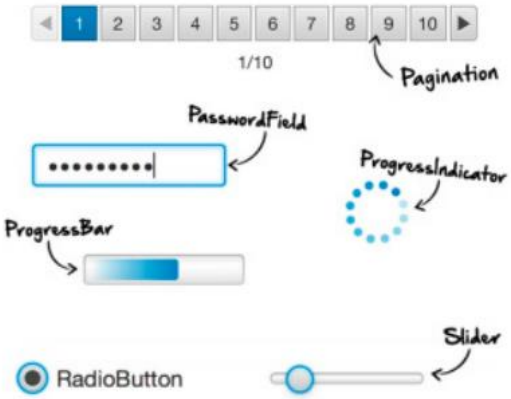




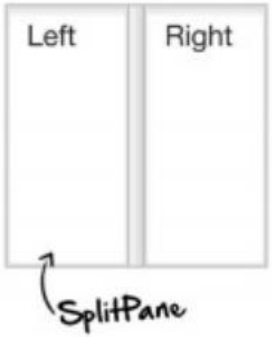
- JavaFX is a Java library for creating GUIs
- Has a set of pre-built **UI components** that can be composed to create a GUI application

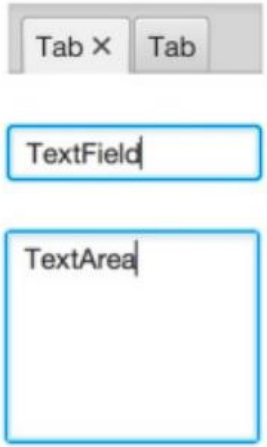

TitledPane
TitledPane
Content
TitledPane
Accordion
Button
CheckBox
ContextMenu
MenuItem
CheckMenuItem
RadioMenuItem 1
RadioMenuItem 2
ChoiceBox
Schwarz
ColorPicker

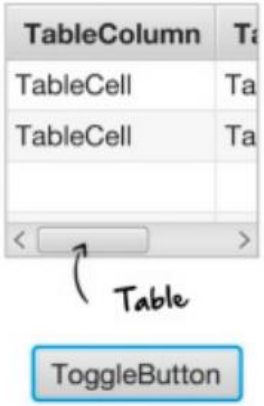

DatePicker
ComboBox
08.01.2015
Hyperlink
Label
ListView
Menu
MenuItem
CheckMenuItem
RadioMenuItem 1
RadioMenuItem 2

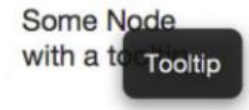

1 2 3 4 5 6 7 8 9 10
1/10
Pagination
PasswordField
ProgressIndicator
ProgressBar
Slider
RadioButton

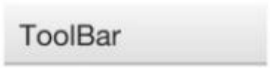

ScrollPane

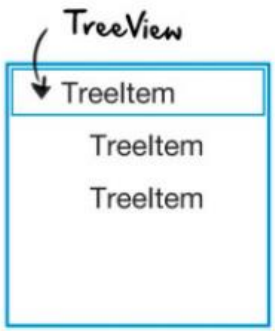

Left
Right
SplitPane

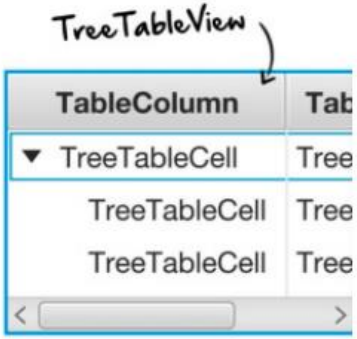

Tab
Tab
TextField
TextArea


TableColumn
TableCell
TableCell
Table
ToggleButton


Some Node
with a
Tooltip


ToolBar

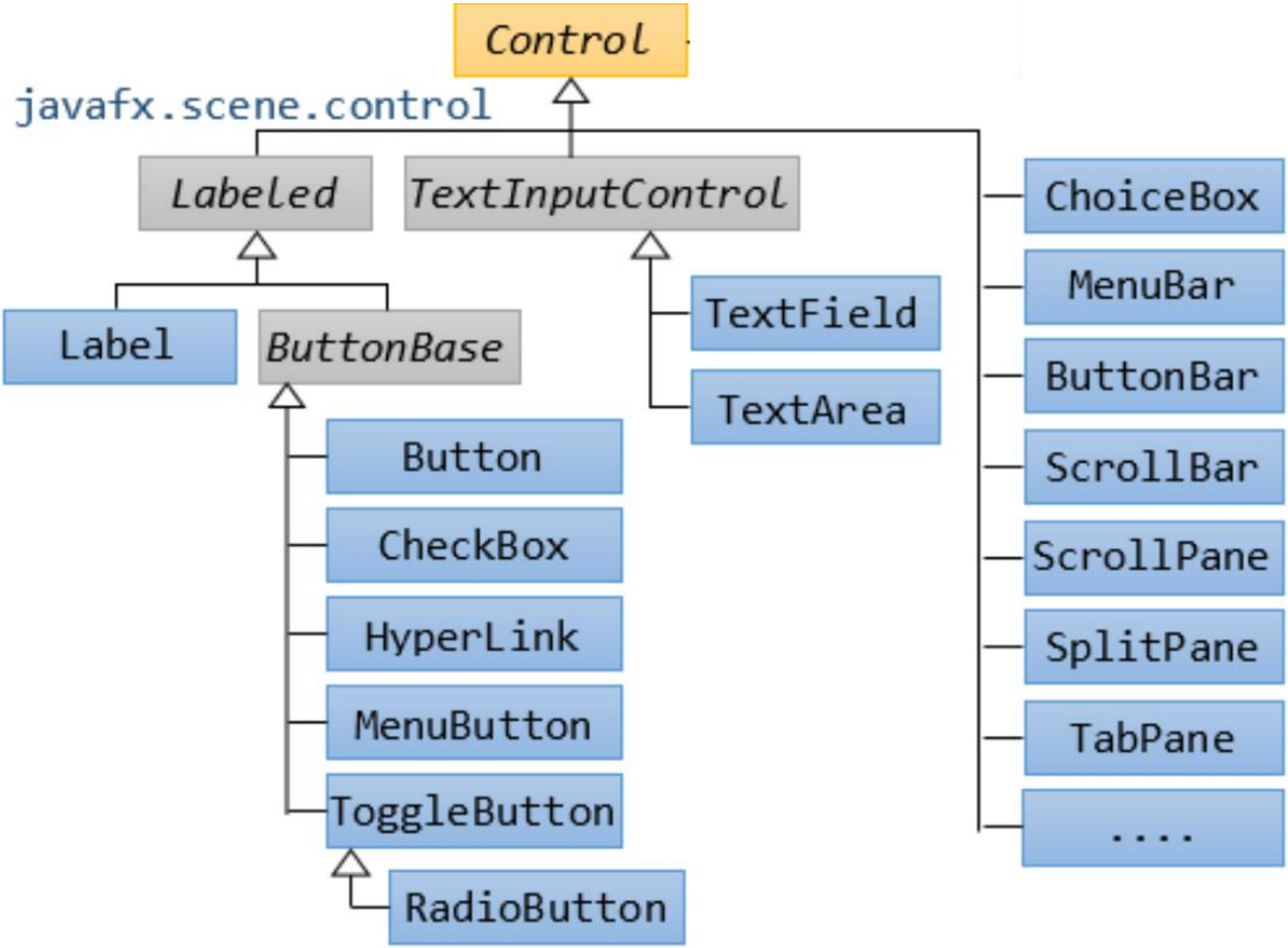

TreeView
TreeItem
TreeItem
TreeItem


TreeTableView
TableColumn
TreeTableCell
TreeTableCell
TreeTableCell

What Makes up JavaFX?

- **UI components**
  - e.g. buttons, text-fields, menus, tables, lists, etc.
- **Containers**
  - A set of UI components are contained and managed by a container. E.g., frames, panels, etc.
- **Layout managers**
  - Dictates how the UI components are arranged
- **UI event handlers**
  - When the user clicks a button, an event handler is called to handle it

JavaFX UI Components



Steps to creating a GUI Interface

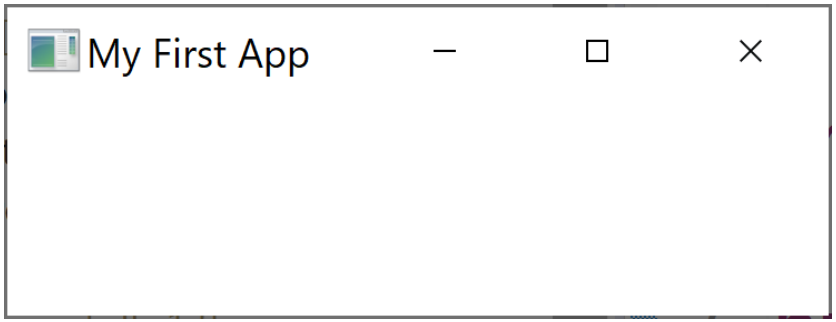
1. Design it on paper:
   - Decide what information to present to user and what input the should supply.
2. Choose components and containers
   - Decide the components and layout on paper
3. Create a form and add components to it
   - Use many panels to group components
4. Add event handlers to respond to the user actions (event driven programming)
   - Do something when the user presses a button, moves the mouse, change text of input field, etc.

Creating GUIs With JavaFX: Stage (1/2)

- Create a class that extends `javafx.application.Application`
- Implement the `start()`
  - `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);
 }
}
```



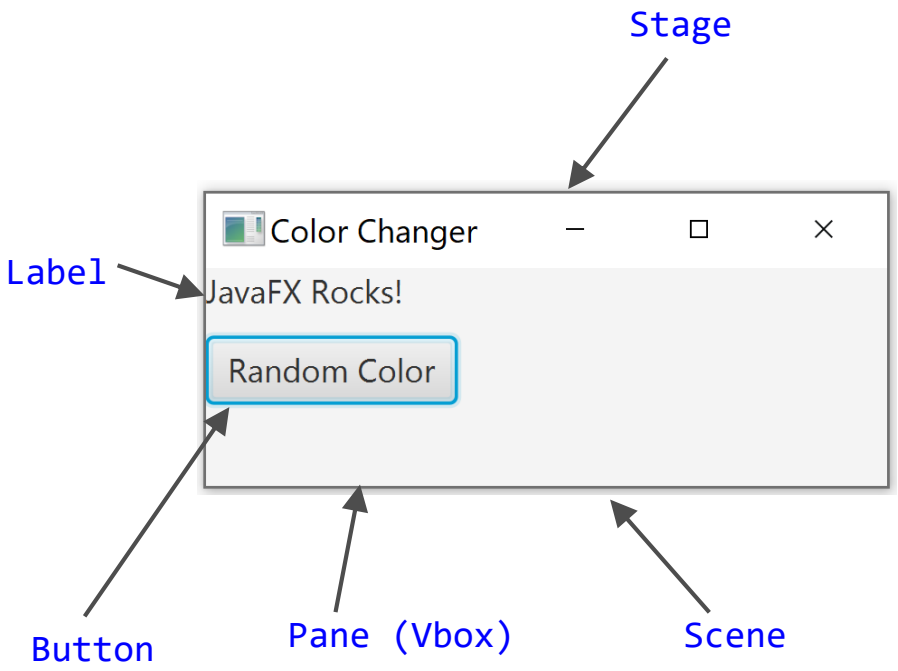
Creating GUIs With JavaFX: Scene (2/2)

- Create a **scene** (instance of `javafx.scene.Scene`) as the top-level container for all UI elements
  - first instantiate `Scene` within `App` class' `start` method
  - then pass that `Scene` into `Stage` using the `setScene` method to *set the scene!*

- UI elements can be added to the `Scene`, such as a `Button` or a `Label`... To get displayed

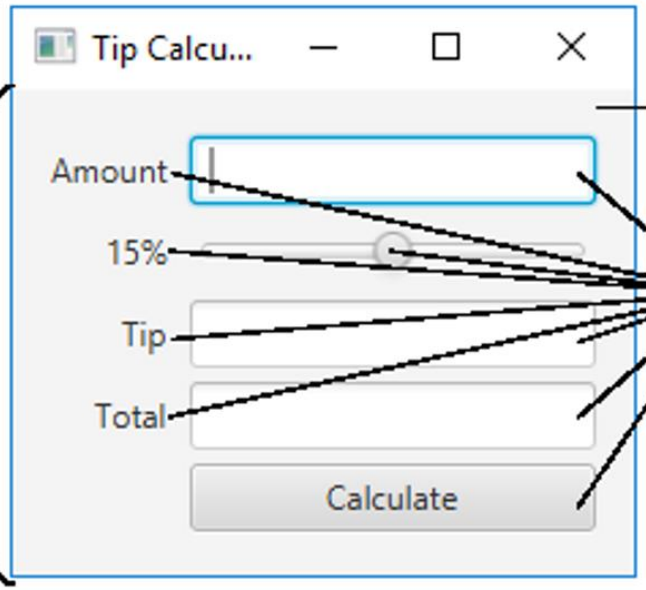
JavaFX Application: ColorChanger

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



JavaFX app window parts

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

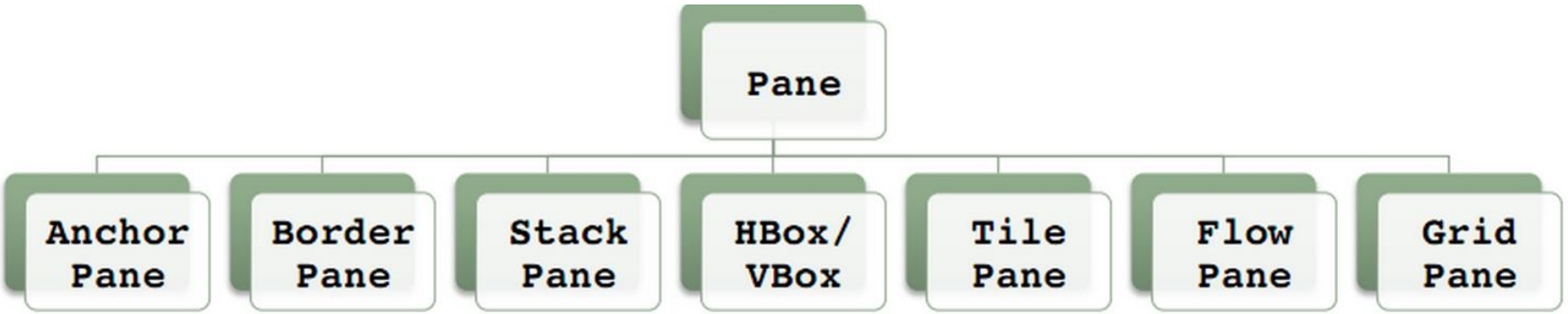
Each of the JavaFX components in this GUI is a node in the scene graph

Label component

ImageView component

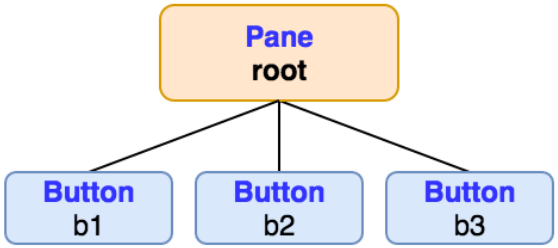
The root of the Scene

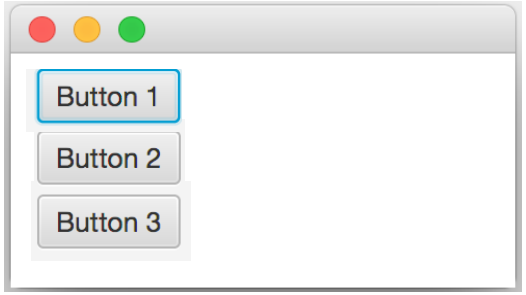
- Root `Node` should be an instance of `javafx.scene.layout.Pane` or one of its subclasses
- Different `Panes` have different built-in layout capabilities to easy positioning of UI elements



Adding UI Elements to the Scene

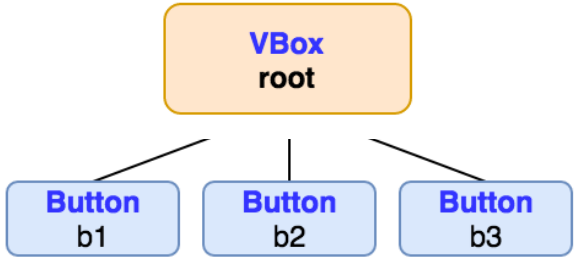
- `getChildren()` returns a `List` of child `Nodes` of the root container
  - in example on right, `root.getChildren()` returns a `List` holding three `Button`s
- To add a `Node` to this list of children, call `add(Node node)` on that returned `List`!
  - can also use `addAll(Nodes… node1, node2, …)` which takes in *any number of* *`Node`s*
- To remove a `Node` from this list of children, call `remove(Node node)` on that returned `List`

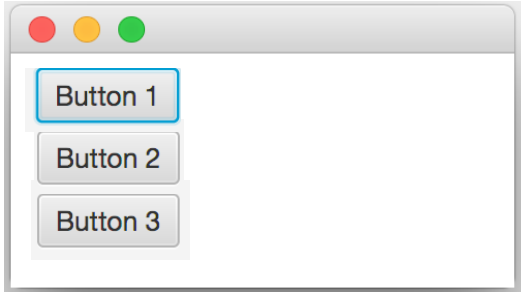




root.getChildren().add(…) in action

- Add 3 Buttons to the Scene by adding them as children of the root Node
- First create buttons
- Then add buttons to root






```
/* 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);
    }
```

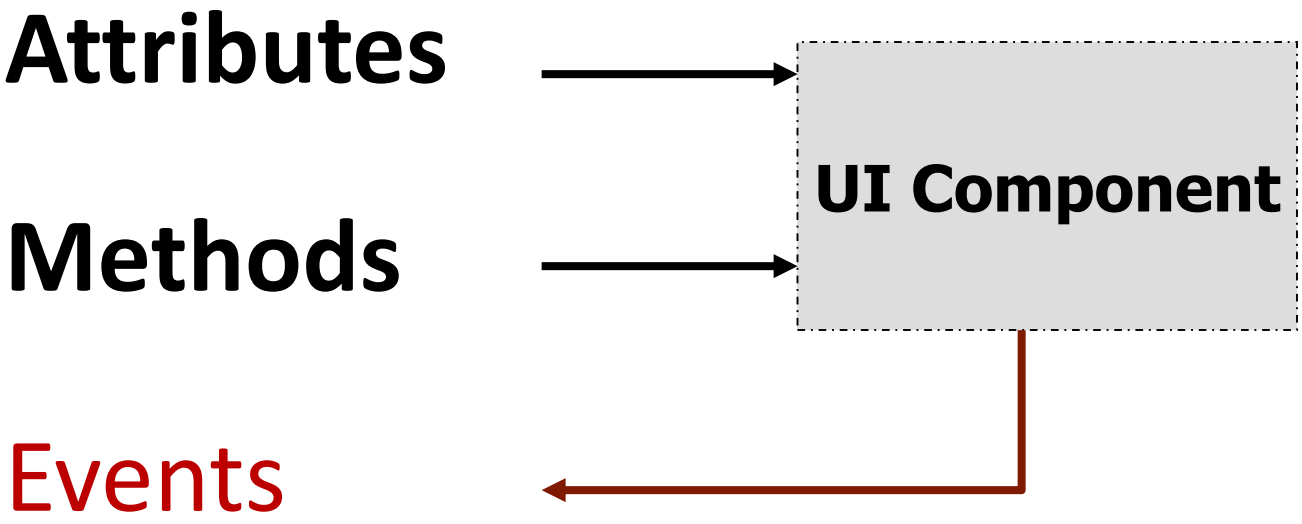
Order matters - order buttons added effects order displayed
(b1, b2, b3) vs. (b2, b1, b3)

VBox layout pane

- VBox layout Pane creates an easy way for arranging a series of children in a *single vertical column*
- We can customize vertical spacing *between* children using VBox's `setSpacing(double)` method
- Can also set positioning of entire vertical column of children
- Default positioning for the vertical column is in TOP_LEFT of VBox (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

UI Component

- UI component is a class that has:



Using a UI Component

1. Create it

- Instantiate object:

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

Button

2. Configure it

- Methods: `button.setText("Press me");`

3. Add it

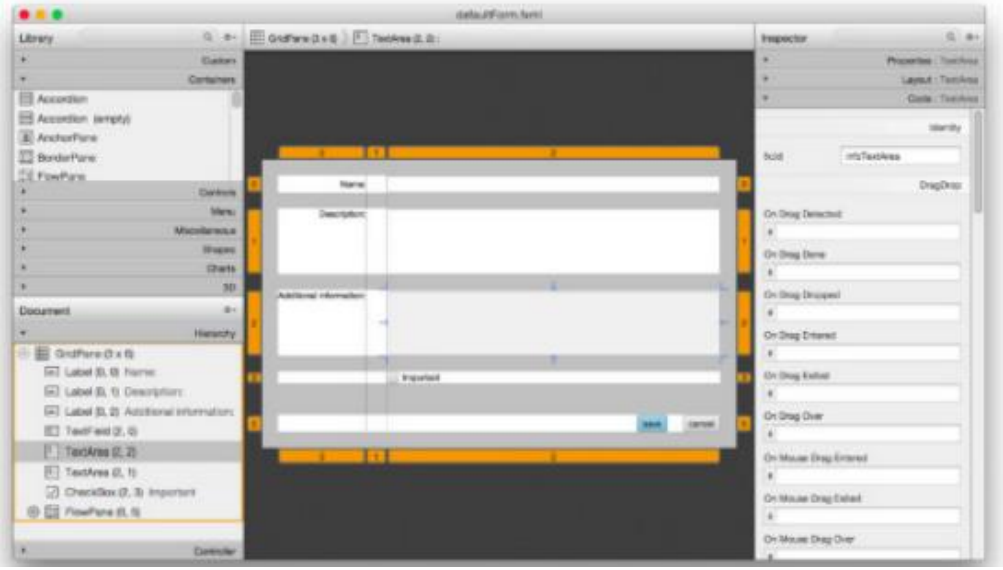
```
panel.add(button); //add it to a
container
```

4. Listen to its events

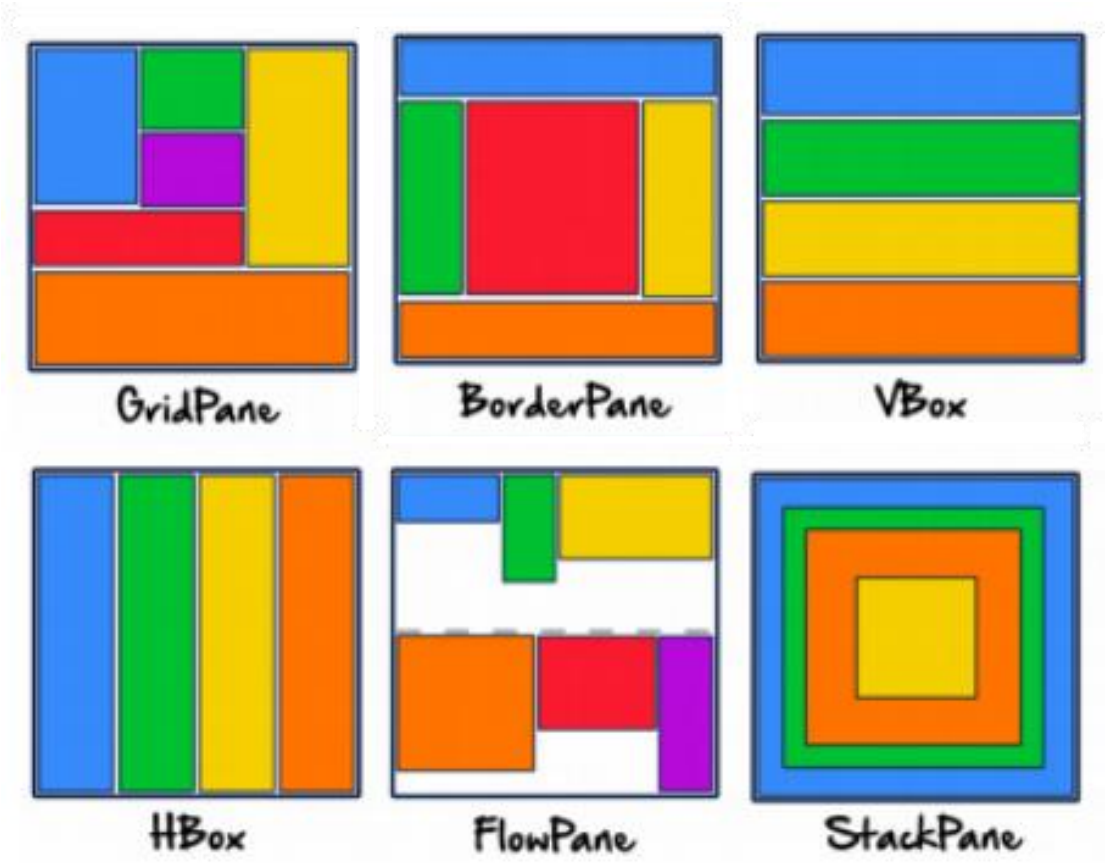
```
// Register an event handler
button.setOnMouseClicked( event ->
System.out.println(event) );
```

FXML

- FXML is an XML-based language that defines the structure and layout of JavaFX UIs.
- FXML allows a clear separation between the view of an app and the logic.
- SceneBuilder is a WYSIWYG editor for FXML



Layouts

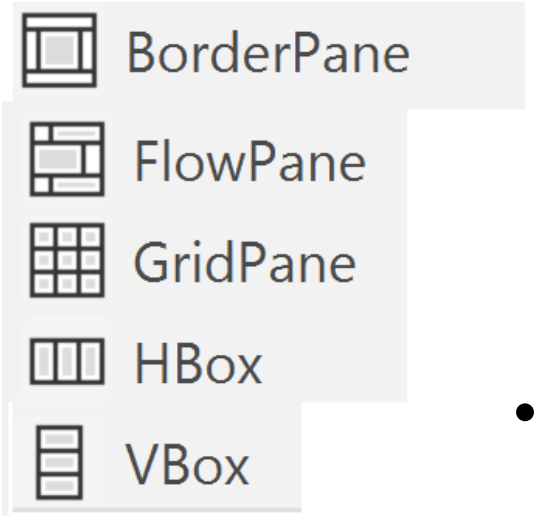


Back

Layouts

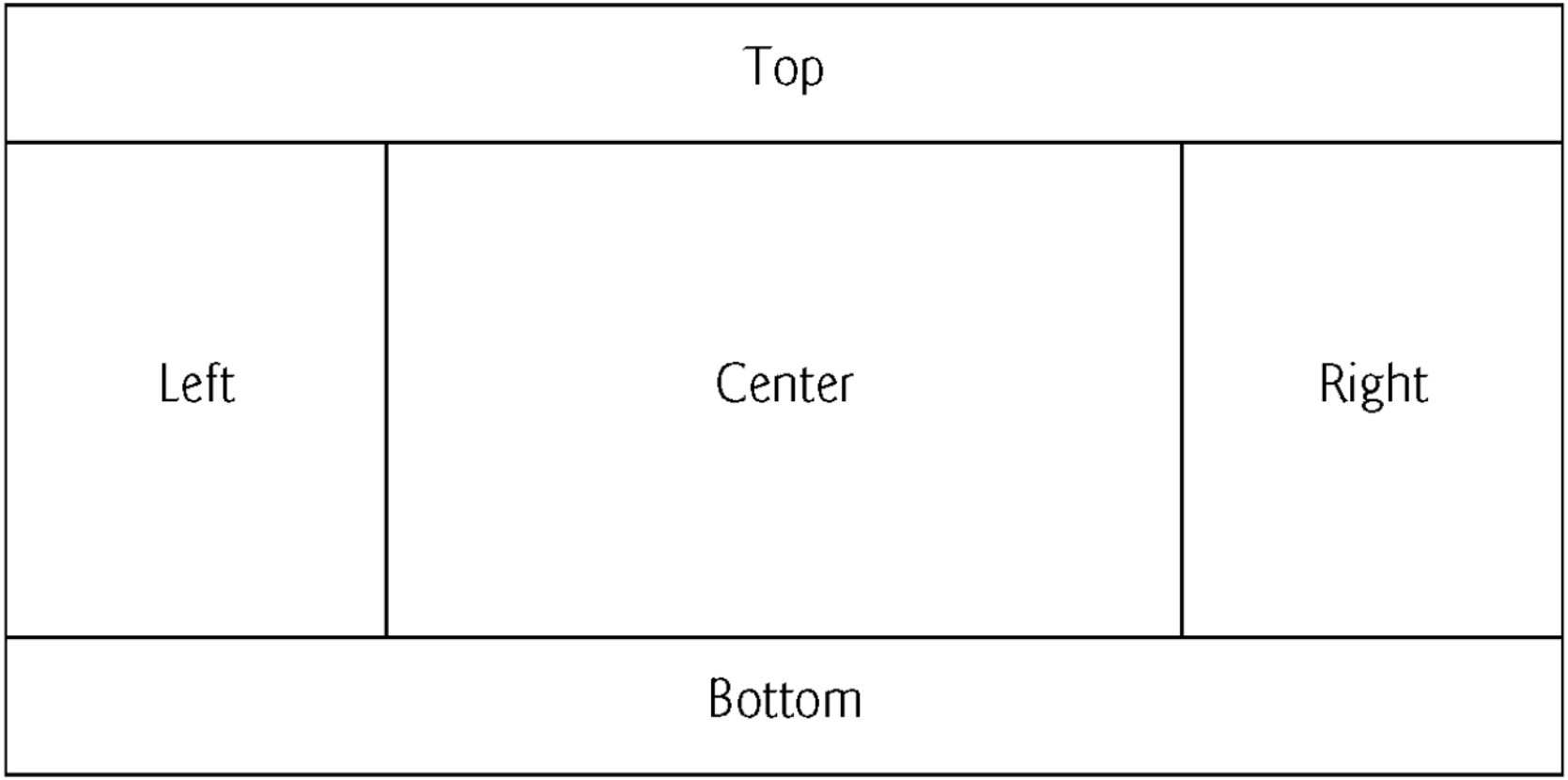
- Automatically control placement of components in a container
  - Frees programmer from handling positioning details
- There are many layout which control how UI components are organized on the container.
- As the window is resized, the UI components reorganize themselves based on the rules of the layout

Common Layouts



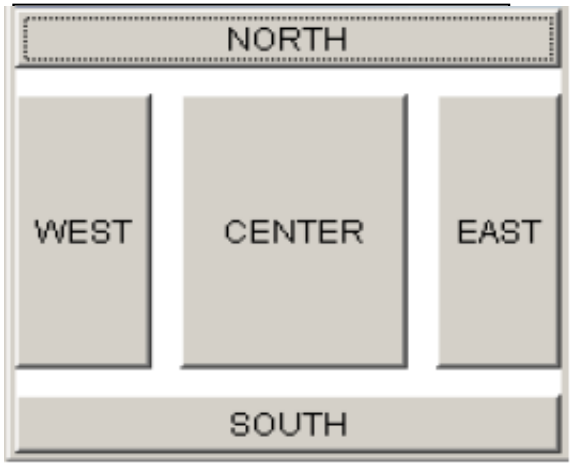
- **BorderPane** - provides five areas: top, left, right, bottom, and center.
- **FlowPane** - lays out its child components either vertically or horizontally, and which can wrap the components onto the next row or column if there is not enough space in one row/column.
- **GridPane** - displays UI elements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

BorderPane five areas

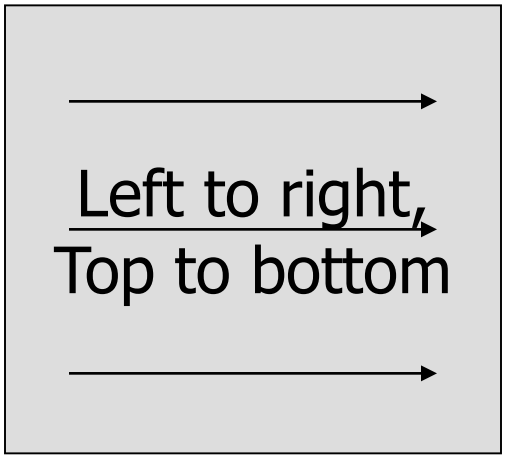


Important Layout Managers

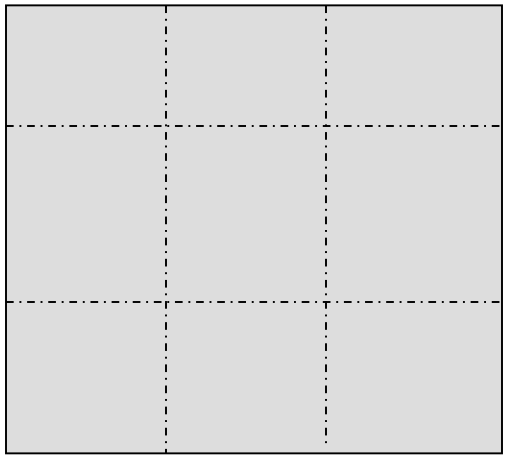
BorderPane



FlowPane



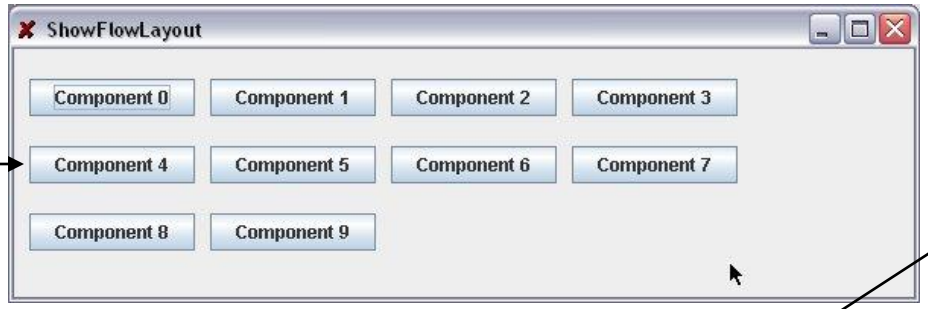
GridPane



FlowPane

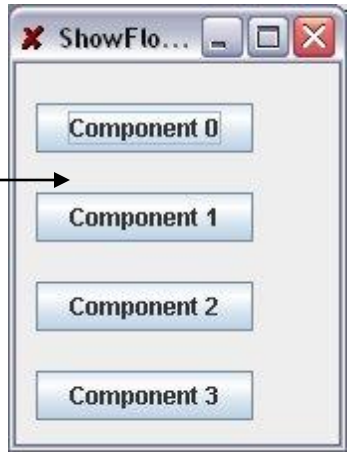
- With **flow pane**, the components arrange themselves from left to right and top to bottom manner in the order they were added

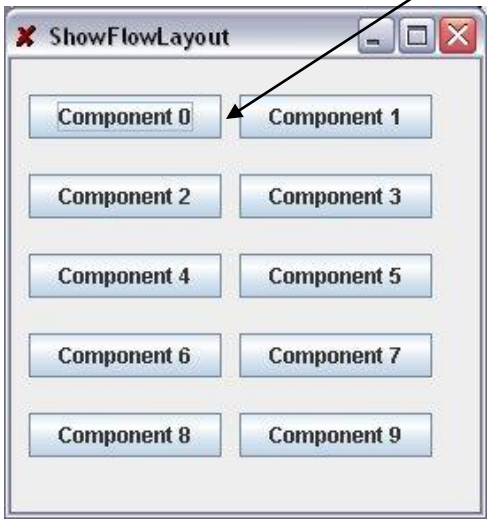
Rows/buttons are left aligned using `FlowPane.LEFT`



Horizontal gap of 10 pixels

Vertical gap of 20 pixels



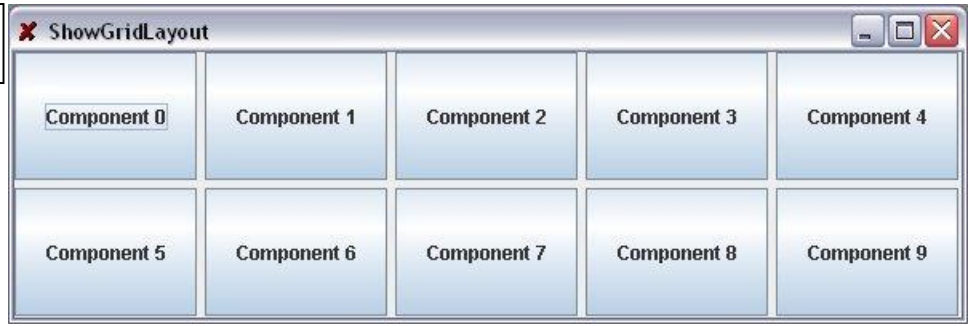


GridPane

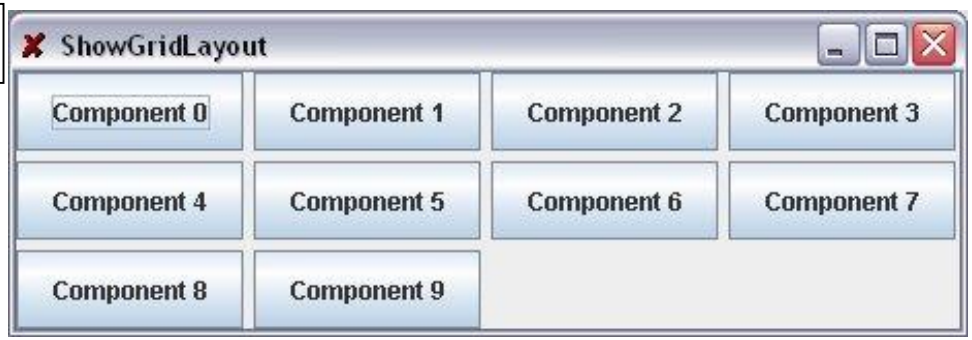
- With **grid pane**, the components arrange themselves in a matrix formation (rows, columns)
  - Divides the container into a number of rows and columns
  - Regions are equally sized
  - Positions components from left to right and top to bottom
- Either the row or column must be non-zero
- The non-zero dimension is fixed and the zero dimension is determined dynamically

GridPane

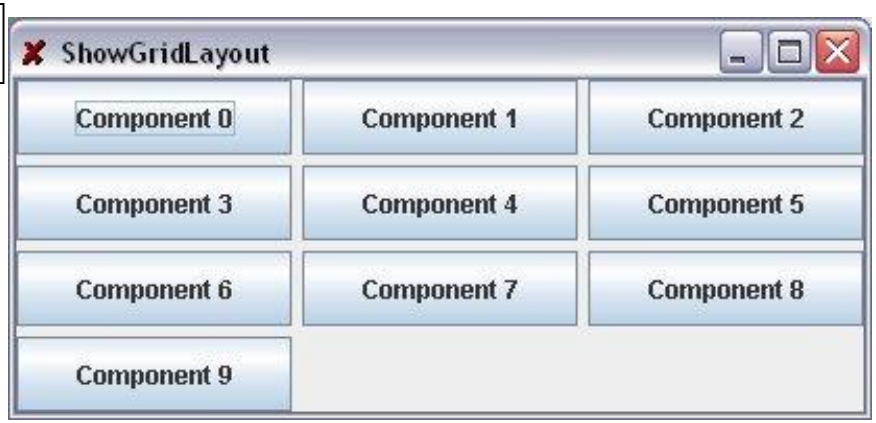
2,4



0,4



4,4



10, 10

BorderPane

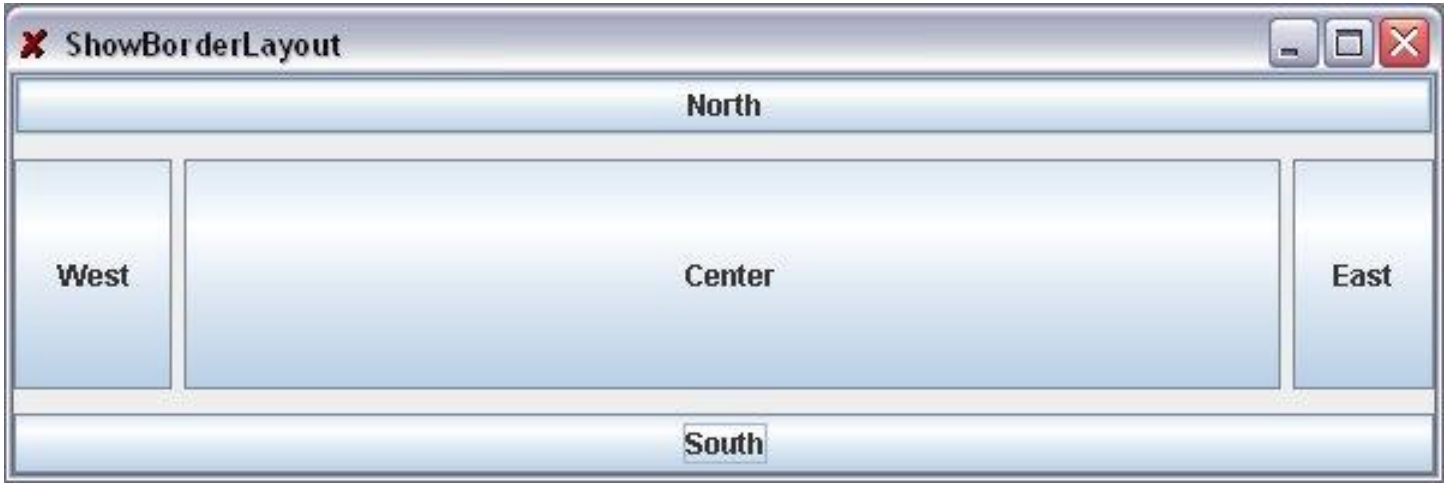
- With **border layout**, the window is divided into five areas:

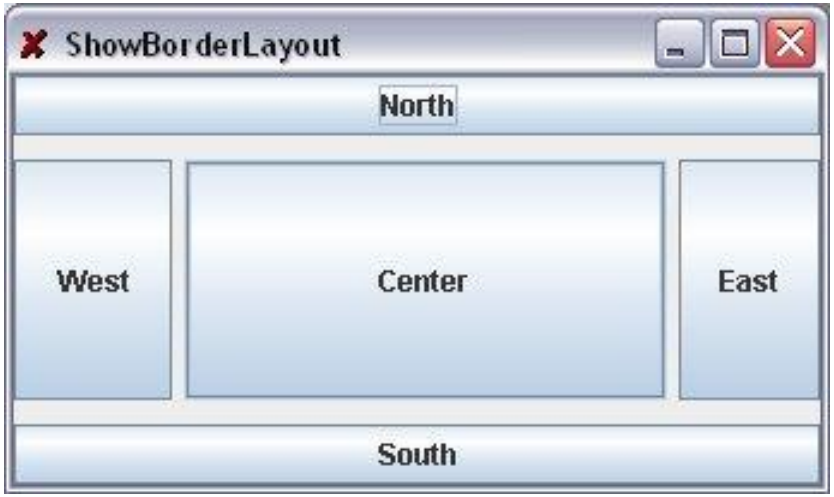
	BorderPane.NORTH	
BorderPane.WEST	**BorderPane.CENTER**	**BorderPane.EAST**
	BorderPane.SOUTH	

- Components are added to the container using a specified index:

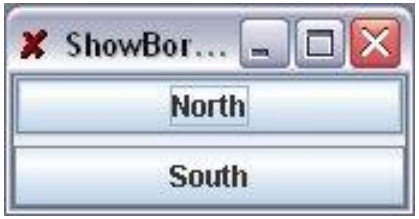
```
container.add(new Button("East"), BorderPane.EAST);
```

BorderPane


ShowBorderLayout
North
West
Center
East
South


ShowBorderLayout
North
West
Center
East
South


North
West
East
South


ShowBor...
North
South

BorderPane

- The components stretch in this manner:
  - North and South stretch horizontally
  - East and West stretch vertically
  - Center can stretch in both directions to fill space

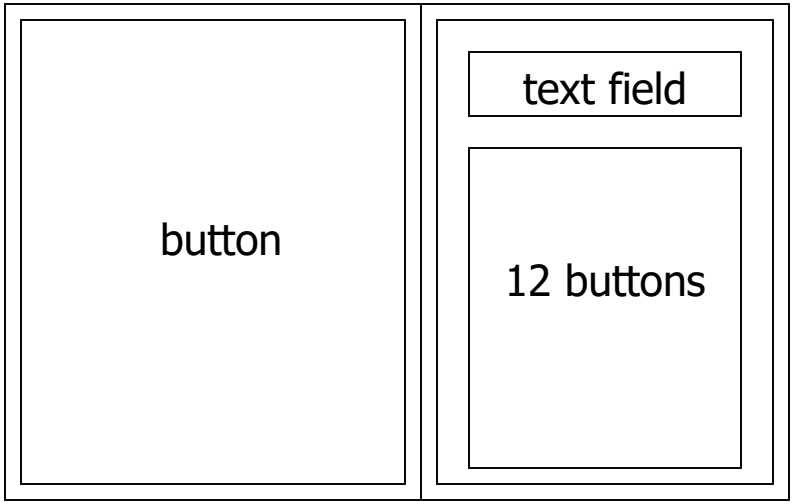
- The default location for a component is `BorderPane.CENTER`

- If you add two components to the same location, only the last one will be displayed

- It is unnecessary to place components to occupy all areas

Panels and Complex Layouts

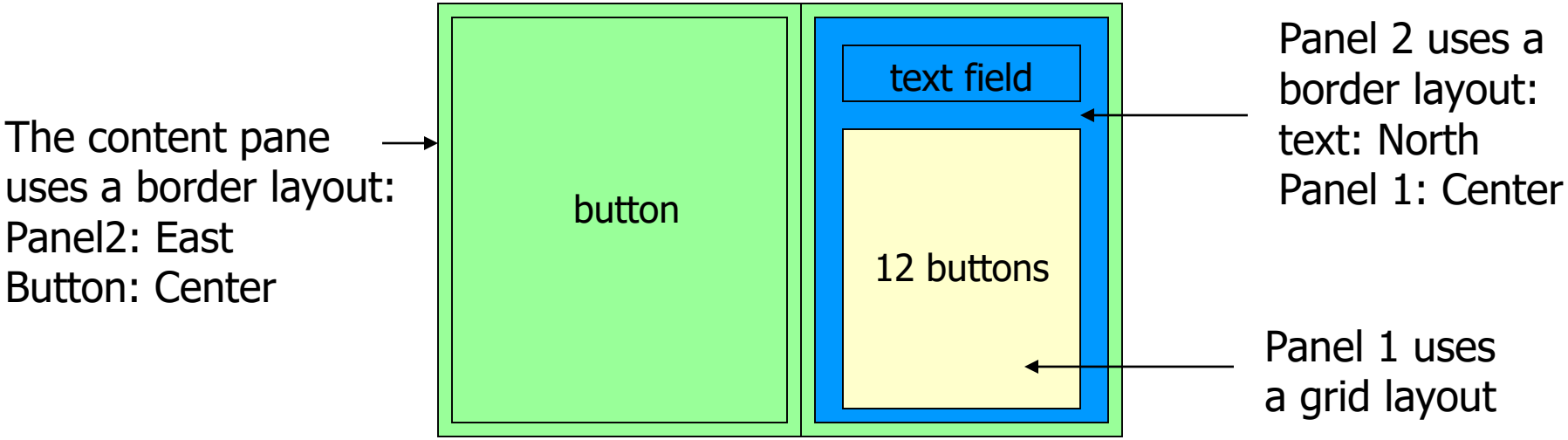
- For more complex layouts
  - Can combine the different layout managers
  - Use of panels to group components
- Example of organizing the UI components for a microwave oven:



- Each panel can use different layout

Panels

- The window can be subdivided into different panels
- The panels act as sub-containers for grouping UI components



Configure Components

Back

Methods of all Swing components

- `public int getWidth()`
  `public int getHeight()`
  Allow access to the component's current width and height in pixels.

- `public boolean isEnabled()`
  Returns whether the component is enabled (can be interacted with).

- `public boolean isVisible()`
  Returns whether the component is visible (can be seen on the screen).

More `JComponent` methods

- `public void setBackground(Color c)`
  Sets the background color of the component to be the given color.
- `public void setFont(Font f)`
  Sets the font of the text on the given component to be the given font.
- `public void setEnabled(boolean b)`
  Sets whether the component is enabled (can be interacted with).
- `public void setVisible(boolean b)`
  Sets whether the component is visible (can be seen on the screen). Set to `true` to show the component, or set to `false` to hide the component.

Fonts

- You can create a font using the `Font` class

```
public Font(String name, int style, int size);
```

- The standard fonts are "`SansSerif`", "`Serif`", "`Monospaced`", "`Dialog`", or "`DialogInput`"

- The styles are `Font.PLAIN`, `Font.BOLD`, `Font.ITALIC`, and `Font.BOLD + Font.ITALIC`

```
Font font1 = new Font("SansSerif", Font.BOLD, 16);
Font font2 = new Font("Serif", Font.ITALIC, 12);
JButton button = new JButton("OK");
button.setFont(font1);
```

Color

- The color of GUI components can be set using the `java.awt.Color` class

- Colors are made of red, green and blue components which range from `0` (darkest shade) to `255` (lightest shade)

- Each UI component has a background and foreground:

```
Color color = new Color(128, 0, 0);
JButton button = new JButton();
button.setBackground(color);     // reddish
button.setForeground(new Color(0, 0, 128));  // blueish
```

Color

- There are 13 constant colors defined in `Color`:
  - `BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW`

Event Driven Programming

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 left mouse 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

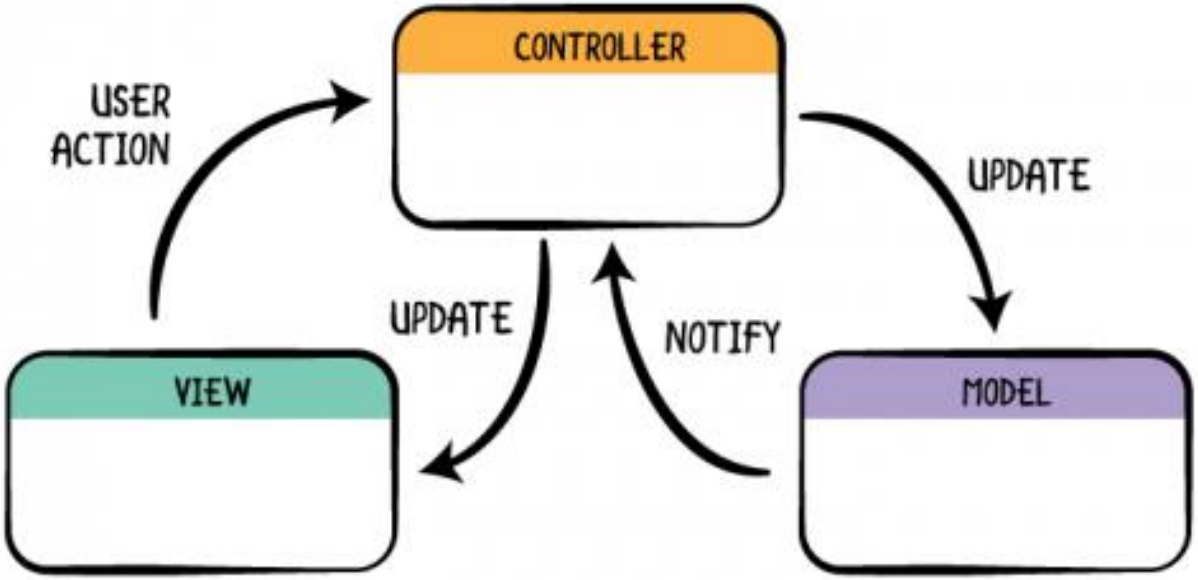

```
btn.setOnMouseClicked(event ->
handleMouseEvent(event));

// Or use method reference
btn.setOnMouseClicked(this::handleMous
eEvent);

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

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

Back


CONTROLLER
USER ACTION
UPDATE
UPDATE
NOTIFY
VIEW
MODEL

MVC-based Application with GUI =
Swing for the view + EventListeners for Controller

View

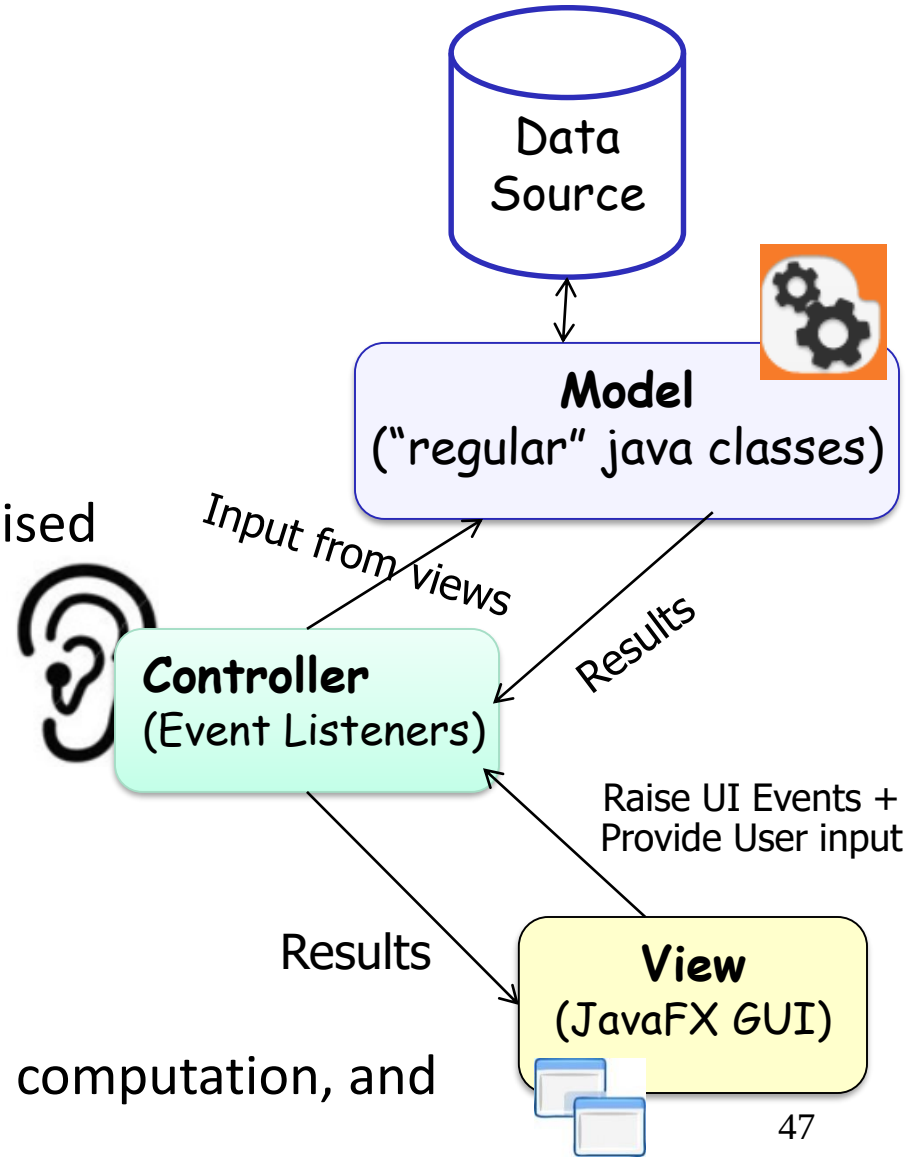
- Gets input from the user
- Notifies the controller about UI events
- Displays output to the user

Controller

- Listens to events raised by the raised
- Gets user input
- Instructs the model to perform actions based on that 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

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 view component, which often needs changes and updates to keep the users continued interests, is separate
    - The UI can be completely changed without touching the model in any way
- **Reusability**
  - The same model can used by different views (e.g., Swing view, Web view and Mobile view)

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

Implementing MVC with Java Swing

1. Define the model (*ordinary Java classes*) to represent data and encapsulate computation

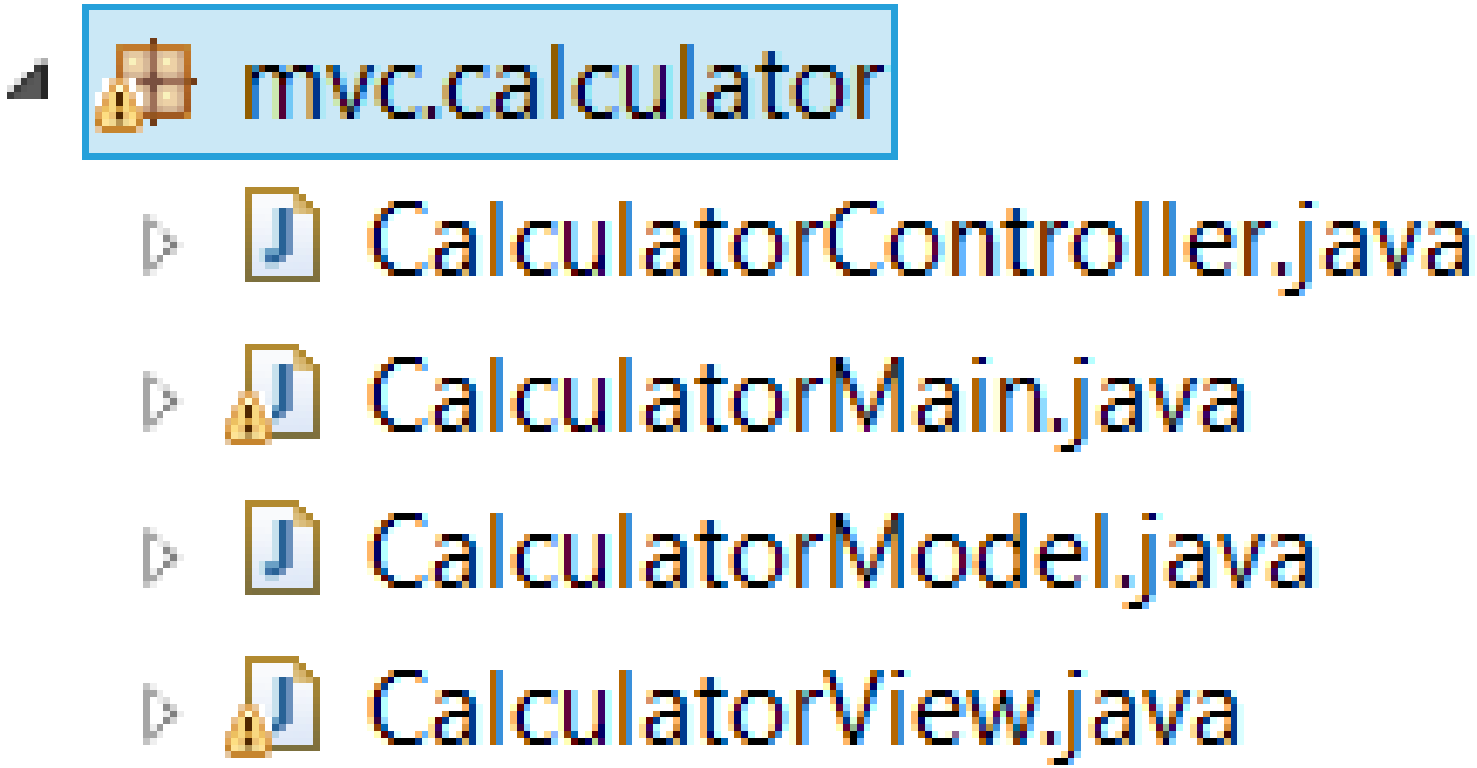
2. Use a controller (i.e., *an EventListener*) to listen to and to **handle events** raised by the view

   - 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

Example


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

- 08-Swing
  - src
    - fileio
    - mvc.calculator
    - mvc.simplecalculator
    - swing.basics
    - swing.events
    - swing.jtable
    - swing.layoutmanager
    - swing.uicomponents

Commonly used UI Components

Back

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
  - Containers
  - Layout managers
  - UI event handlers
  - Building GUI Applications using the Model-view-controller (MVC) Pattern