UPskill

Interfaces Gráficas

JavaFX

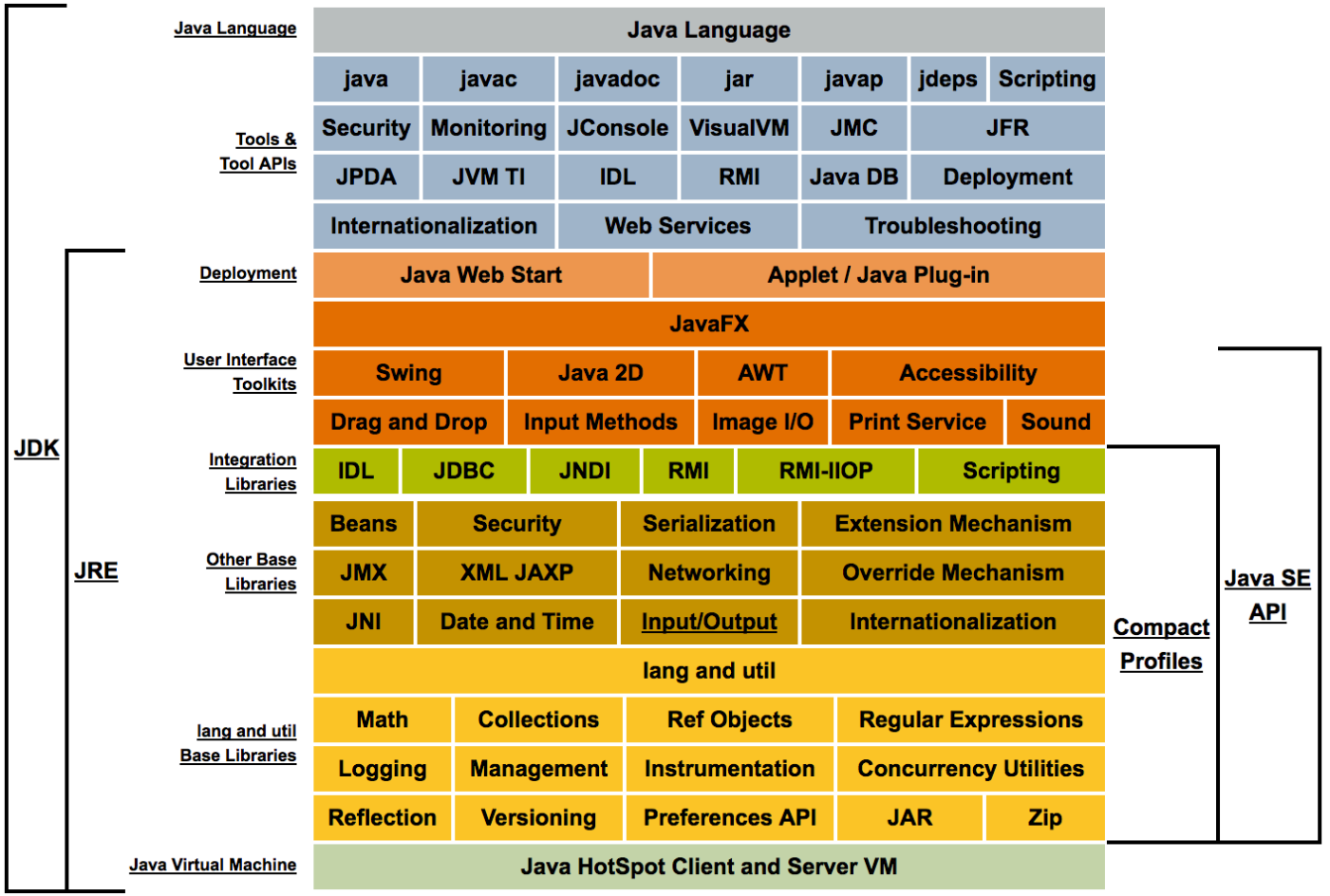
P.PORTO

isep Instituto Superior de Engenharia do Porto

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Instituto Superior de Engenharia do Porto

JavaFX

- An API included in Java 7 for UI development
- The successor of Java Swing



Java Conceptual Diagram
https://docs.oracle.com/javase/8/docs/index.html

Reasons to choose JavaFX

- Rich internet cross platform application
  - Windows, Mac & Linux
- 100% Java APIs with millions of libraries
- Easy to develop even for beginners
- Drag and drop application
- Only uses Java language programming
- Rapid application development using an IDE (*e.g.*, Apache Netbeans, IntelliJ) and SceneBuilder

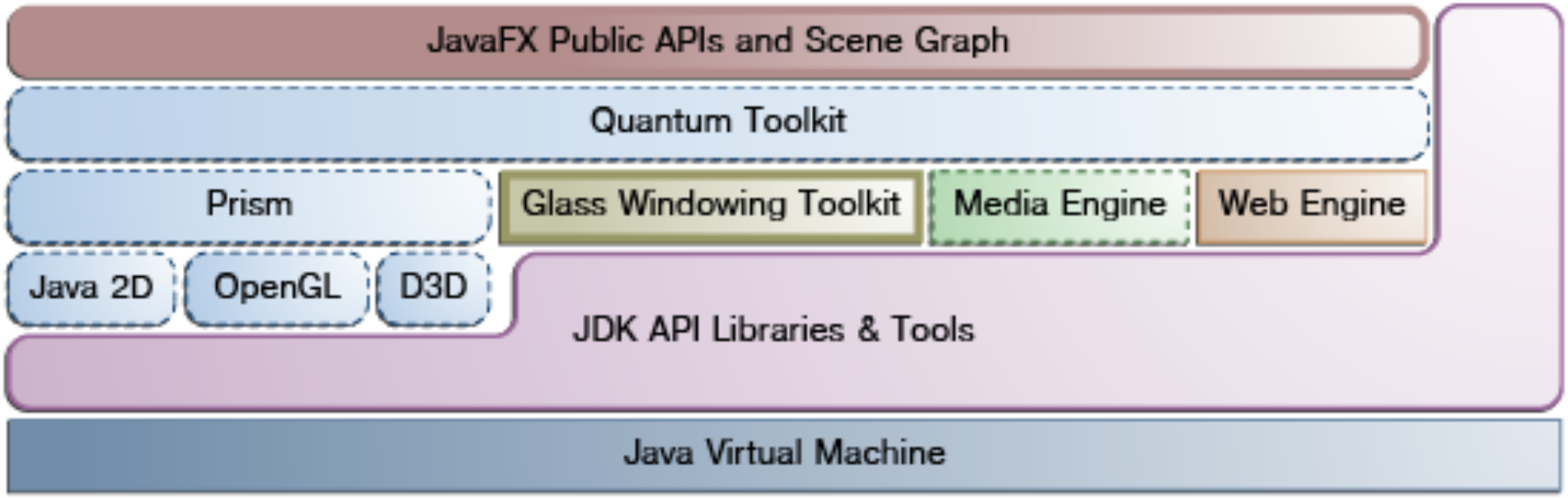
JavaFX Key Features

- FXML → MVC Pattern Support
- WebView (embed web pages within a JavaFX application, including Javascript support)
- Built-in UI controls, CSS and Themes (Modena, Caspian, *etc*.)
- 3D Graphics Features (Shape3D)
- Multi-touch Support, Hi-DPI support, Rich Text Support
- Hardware-accelerated graphics (uses optimally the GPU)
- High-performance media engine (playback of web multimedia content)
- Self-contained application deployment model
- Support FXML: a XML-based declarative language to define the structure of the user interface separated from the application code

Requirements

- IDE: Apache Netbeans, IntelliJ, Eclipse, *etc.*
- JDK 8 has JavaFX 8 already included in installer
- SceneBuilder 2.0
  - Installer for multiple OS: http://www.oracle.com/technetwork/java/javafxscenebuilder-1x-archive-2199384.html

JavaFX Architecture



Anatomy of a JavaFX application

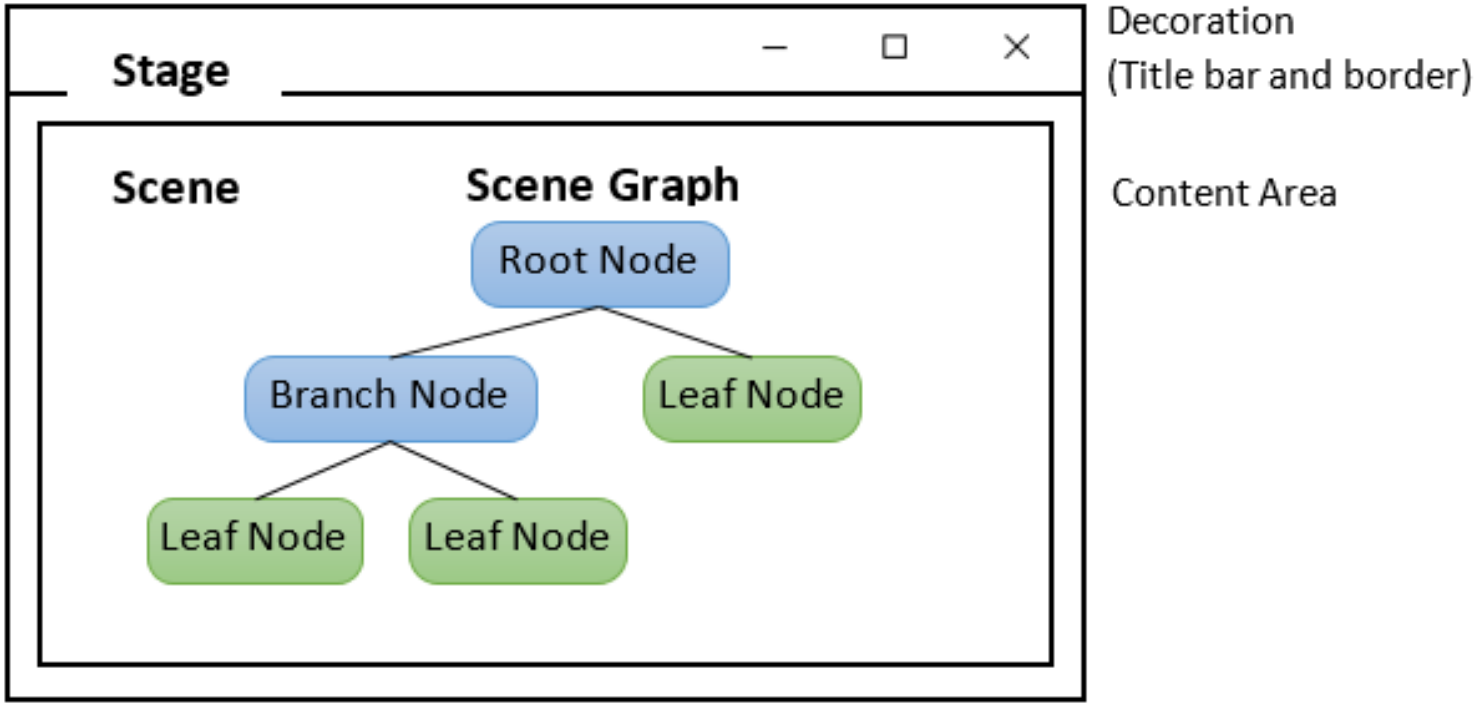
- Each JavaFX program must extend the Application class – the main class of a JavaFX program
- start() method is the main entry point of the application - first method to be called
- A JavaFX application consists of a Stage and a Scene …

Theatre metaphor

- JavaFX uses the metaphor of a theater to model the graphics application
- A *stage* (defined by the javafx.stage.Stage class) represents the top-level container (window)
- Individual controls (or components) are contained in a *scene* (defined by the javafx.scene.Scene class)
- An application can have more than one scenes, but only one of the scenes can be displayed on the stage at any given time

Theatre metaphor (cont.)

- The contents of a scene is represented in a hierarchical *scene graph of nodes* (defined by javafx.scene.Node)



Theatre metaphor (cont.)

- To construct the UI:
  - Prepare a scene graph
  - Construct a scene, with the root node of the scene graph
  - Setup the stage with the constructed scene

JavaFX Application Life Cycle

- A JavaFX application extends from javafx.application.Application
- The JavaFX runtime maintains an Application's life cycle as follows:
  - It constructs an instance of Application
  - It calls the Application's init() method
  - It calls the Application's start(javafx.stage.Stage) method, and passes the primary stage as its argument
  - It waits for the Application to complete (e.g., via Platform.exit(), or closing all the windows)
  - It calls the Application's stop() method
- The start() is an abstract method, that must be overridden; The init() and stop() has default implementation that does nothing
- If you use System.exit(int) to terminate the application, stop() will not be called

Stage

- A *stage* (defined by the javafx.stage.Stage class) represents the top-level container, typically a window
- A stage is divided into decoration (title bar and border) and the content area
- A stage can have one of these styles:
  - StageStyle.DECORATED: solid white background with decorations
  - StageStyle.UNDECORATED: solid white background with no decorations
  - StageStyle.TRANSPARENT: transparent background with no decorations
  - StageStyle.UTILITY: solid white background with minimal decorations
- A primary stage is created by the JavaFX runtime, and passed into the Application as an argument in the Application's start() method

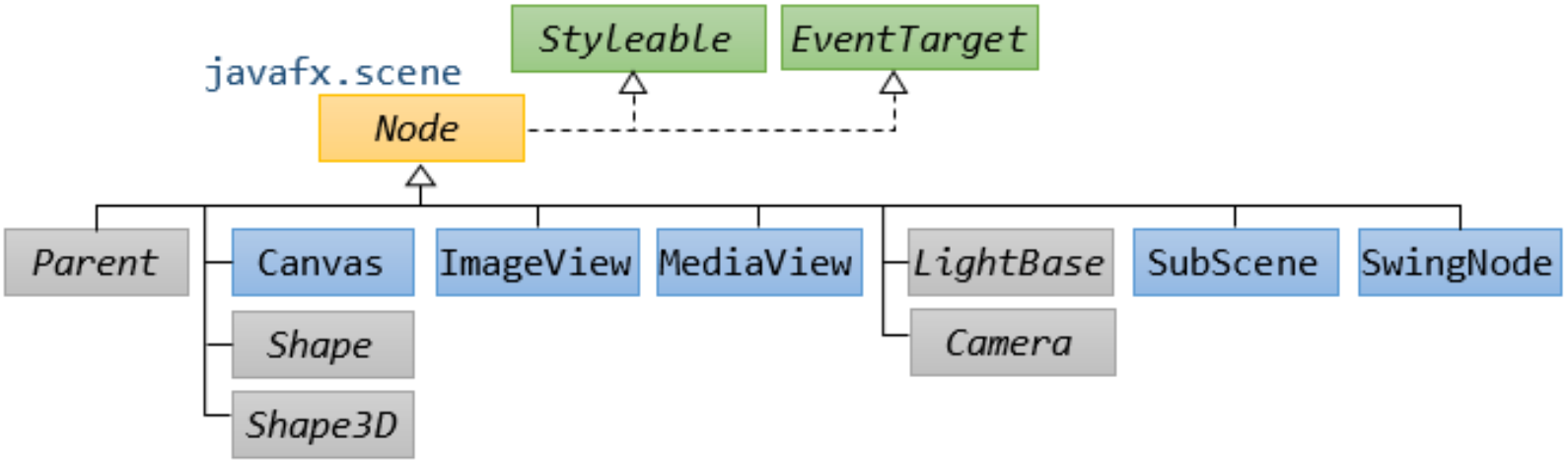
Scene and Scene Graph

- The contents of a scene is represented in a hierarchical (tree-like) *scene graph of nodes*
- To construct a Scene, we need to pass the root of the scene graph and optional width and height

Scene(Parent root, double width, double height)

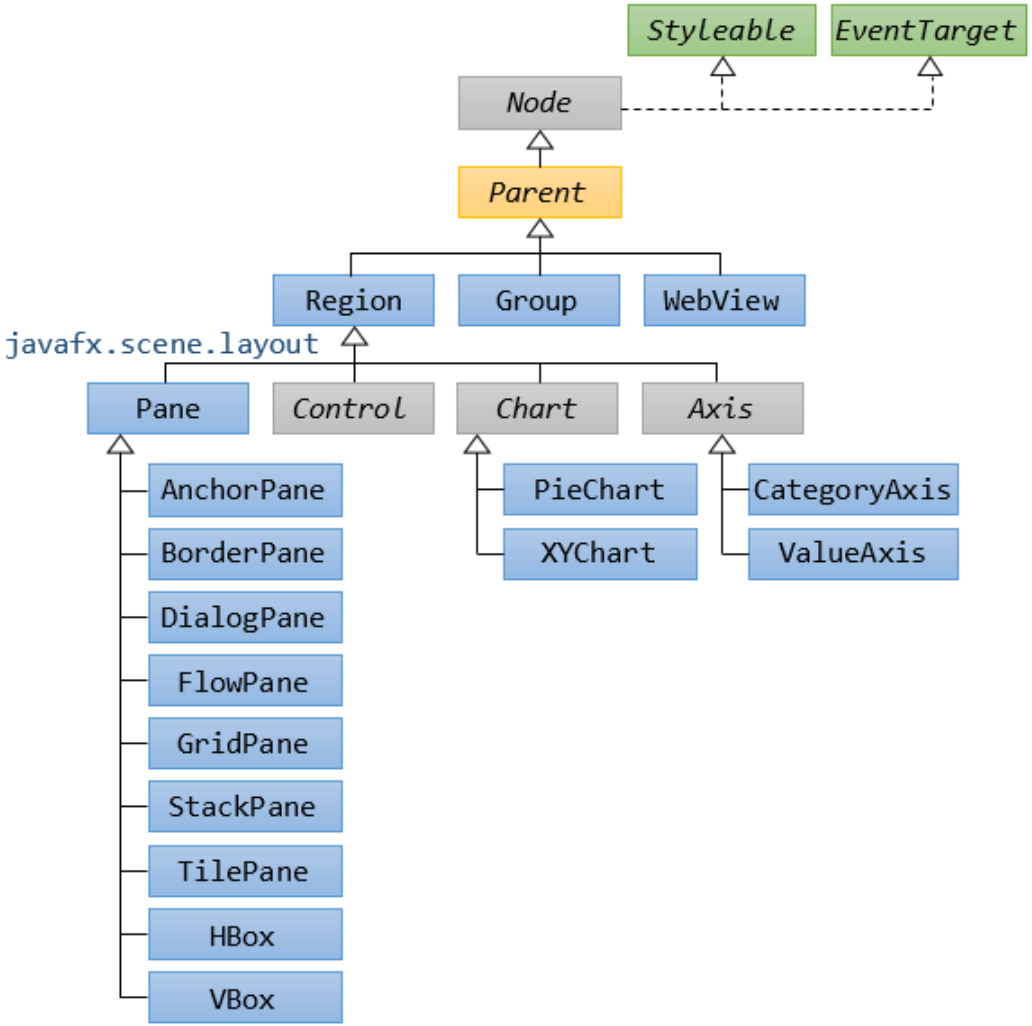
Nodes

- A node is defined by an abstract class javafx.scene.Node, which is the superclass of all the UI elements:
  - Controls (Components): subclasses of Parent in package javafx.scene.control, e.g., Label, TextField, Button
  - Layout Pane (Containers): subclasses of Parent in package javafx.scene.layout, e.g., StackPane, FlowPane, GridPane, BorderPane
  - Geometrical Shapes: subclasses of Shape and Shape3D in package javafx.scene.shape, e.g., Circle, Rectangle, Polygon, sphere, Box
  - Media Elements: e.g., ImageView, MediaView (playable by media player) in packages javafx.scene.image and javafx.scene.media

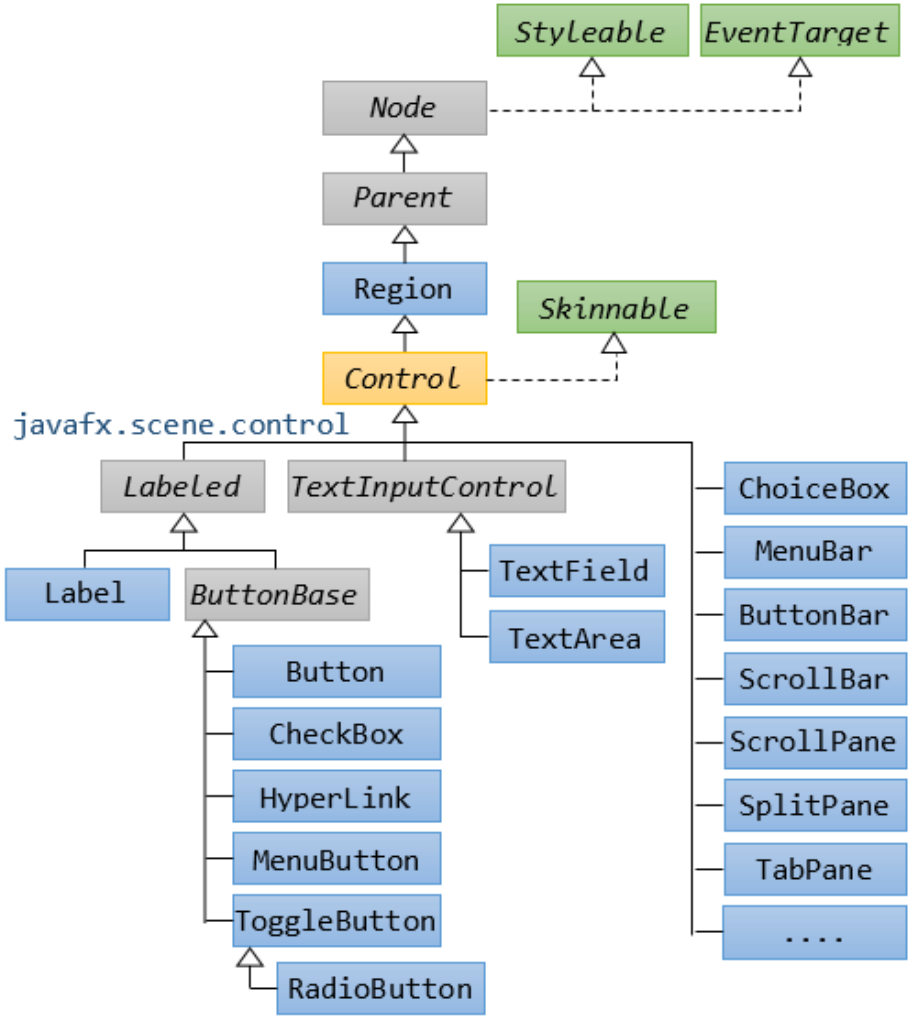


Parents

- Parent (Branch Node): having child nodes, defined by the abstract class javafx.scene.Parent with 3 concrete subclasses:
  - Group: Any transform (*e.g.* rotation), effect, or state applied to a Group will be applied to all children of that group
  - Region: is the base class for all UI Controls and layout containers. Region has 4 subclasses: Control (UI controls), Pane (Container), Chart (PieChart, XYChart) and Axis.
  - WebView: for HTML content



Controls


Styleable
EventTarget
Node
Parent
Region
Skinnable
Control
javafx.scene.control
Labeled
TextInputControl
ChoiceBox
Label
ButtonBase
TextField
MenuBar
TextArea
ButtonBar
Button
ScrollBar
CheckBox
ScrollPane
HyperLink
SplitPane
MenuButton
TabPane
ToggleButton
....
RadioButton

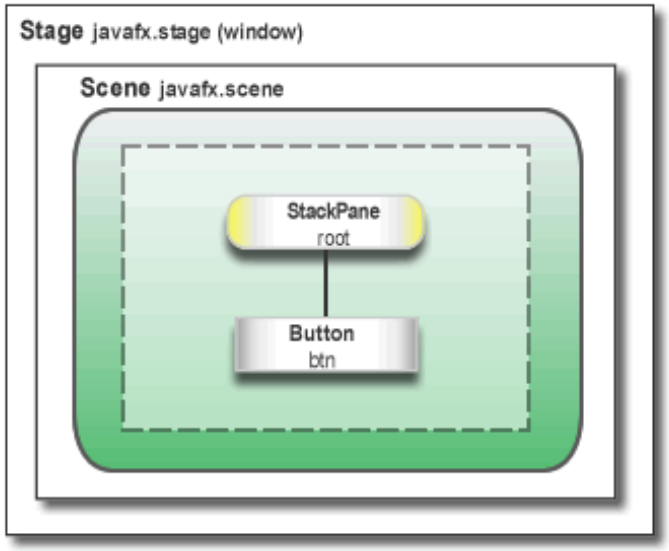
How to develop a JavaFX application

- Programmatically
  - Designing a scene and placing controls using code can become messy as the complexity grows
- FXML and Scene Builder
  - Rapid scene development / mockup using Scene Builder
  - FXML is not a compiled language; you do not need to recompile the code to see the changes; just reload the FXML file
  - It provides a clear separation of GUI from logic/controller
  - So, you can have different versions of a scene/view using the same controller. This is handy for demo's for instance
  - The content of an FXML file can be localized as the file is read
  - But the FXML take some time to be loaded and needs to be parsed

JavaFX Programmatically

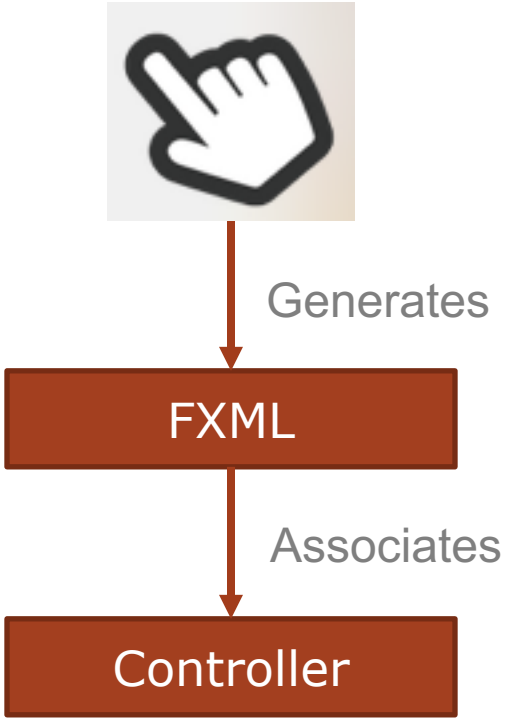
```
import ...

public class HelloWorld extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello World!");
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!"); } });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}
```



FXML and SceneBuilder

- Screen building using SceneBuilder which generates FXML - Drag and Drop application for Layout GUI
  - Layout containers
  - Place controls
  - Associate events
  - Style
- Map to a controller class

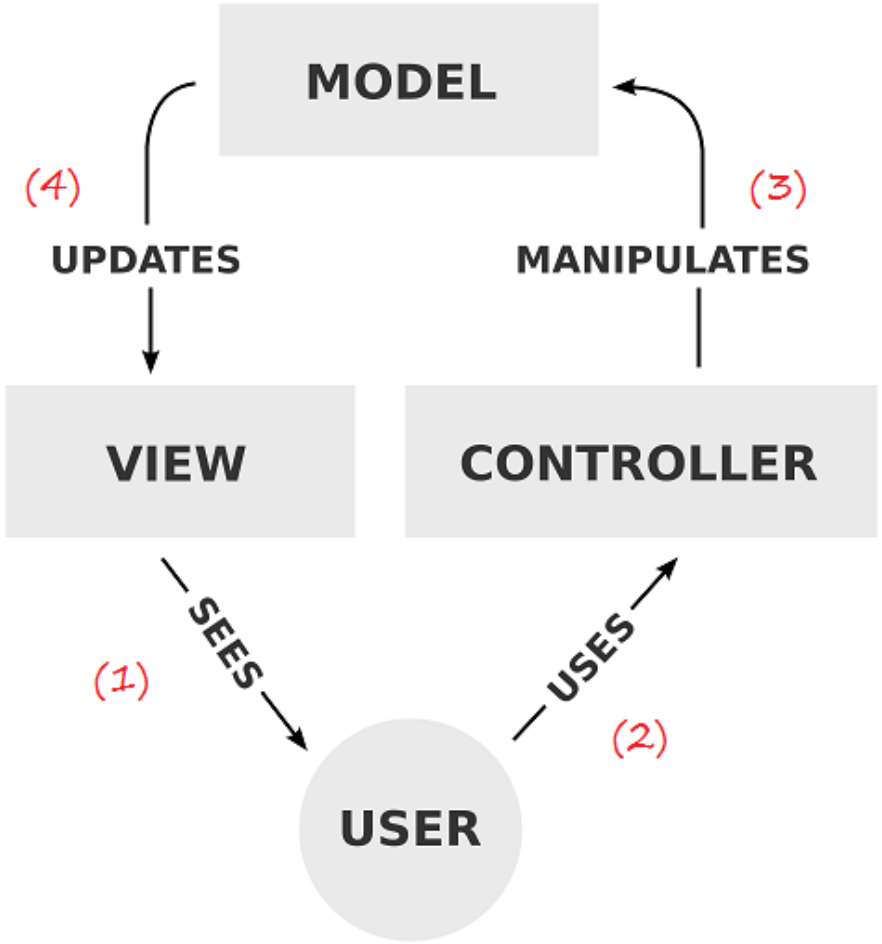


MVC Pattern

- Helps to decouple data access and business logic from the manner in which it is displayed to the user
- Can be broken down into three elements:
  - **Model** - represents data and the rules that govern access to and updates of this data
  - **View** - The view renders the contents of a model. It specifies exactly how the model data should be presented. If the model data changes, the view must update its presentation as needed
  - **Controller** - The controller translates the user's interactions with the view into actions that the model will perform

MVC Pattern (cont.)

1. After seeing it on VIEW
2. Users use CONTROLLER
3. Manipulate data (Update, modify, delete, *etc.*), the data on MODEL has been changed.
4. Displaying the data of MODEL on VIEW.



FXML

- XML-based language that provides the structure for building a user interface separate from the application logic of your code.
- FXML (Declarative)

```
<BorderPane>
    <top>
        <Button text="Sign In"/>
    </top>
    <center>
        <TextField/>
    </center>
</BorderPane>
```

Can be created using Scene Builder

Using FXML to Create UI

- FXML Loader

```
Parent root = FXMLLoader.load(getClass().getResource("Scene.fxml"));
Scene scene = new Scene(root, 300, 275);
```

- Create the link between view and control

```
<BorderPane fx:controller="FXMLController">
        <top>
                <Button text="Sign In" onAction=
                        "#handleSubmitButtonAction"/>
        </top>
        <center>
                <TextField fx:id="actiontarget" />
        </center>
</BorderPane>
```

Using FXML to Create UI (cont.)

- Define the code to handle events in the controller class

```
public class FXMLExampleController implements Initializable {
    @FXML
    private TextField actiontarget;
    @FXML
    protected void handleSubmitButtonAction(ActionEvent event) {
                    actiontarget.setText("Sign in button pressed");
    }
    @Override
    public void initialize(URL location, Resources resources){
    }
}
```

Controller Class

- initialize() method – the controller class implements Initializable interface
  - Called once the contents of its associated FXML document have been completely loaded
- FXML tag
  - These annotation marks a protected or private class member as accessible to FXML
  - In the preview's controller class:
    - First FXML tag: the Text field is annotated to allow the loader to set its value
    - Second FXML tag: allow the handler method handleSubmitButtonAction() be called when a event occurs on the Button

Controller Class (cont.)

- Contains event handlers
- Each FXML can have only one controller class
  - However, a FXML file can include other FXML files (via the <fx:include> element) and each of them can have its own controller
  - Controller instances for nested FXML documents loaded via the <fx:include> element are mapped directly to member fields of the including controller

Components in JavaFX

- Containers
  - Accordion, Anchor Pane, Stack Pane, Tab Pane, Grid Pane, Hbox, Vbox, *etc.*
- Controls
  - Button, Choice box, Combo box, ImageView, WebView, TextField, TextArea, Label, *etc.*
- Shapes
  - Rectangle, Circle, Arc, Ellipse, Line, Polygon, Text, Cubic Curve, *etc.*
- Charts
  - Area chart, Bar chart, Scatter Chart, Pie chart, Bubble chart, *etc.*

Layout Panes - Containers

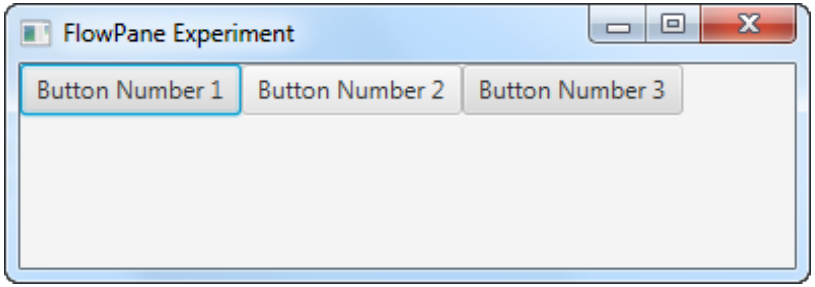
- *Layout panes* are containers which are used for flexible and dynamic arrangements of UI controls within a scene graph of a JavaFX application
- As a window is resized, the layout pane automatically repositions and resizes the nodes it contains

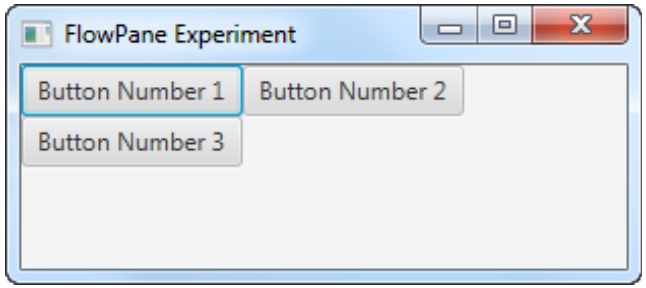
Layout Panes - Containers (cont.)

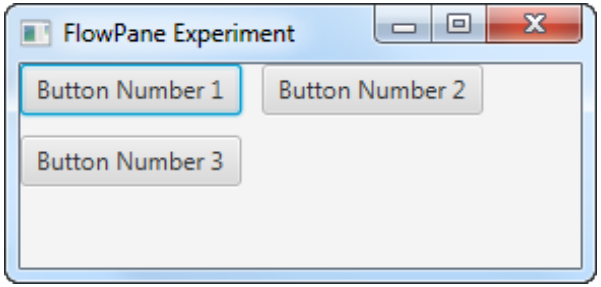
- JavaFX built-in layout panes:
  - FlowPane – lays out its children in a flow that wraps at the flowpane's boundary.
  - HBox – arranges its content nodes horizontally in a single row.
  - VBox – arranges its content nodes vertically in a single column.
  - AnchorPane – anchor nodes to the top, bottom, left side, or center of the pane.
  - BorderPane – lays out its content nodes in the top, bottom, right, left, or center region.
  - StackPane – places its content nodes in a back-to-front single stack.
  - TilePane – places its content nodes in uniformly sized layout cells or tiles.
  - GridPane – places its content nodes in a grid of rows and columns.

Layout Panes - Containers (cont.)

- 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

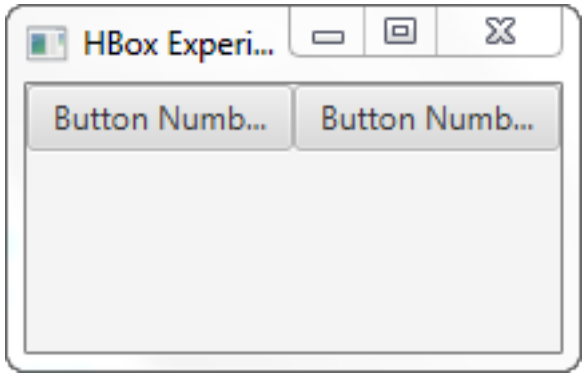






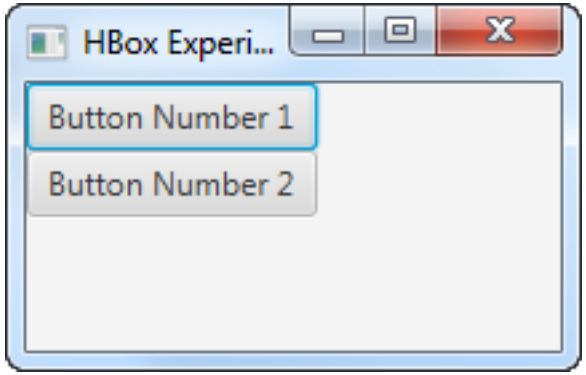
Layout Panes - Containers (cont.)

- HBox – arranges its content nodes horizontally in a single row; controls are kept on the same horizontal row even if there is not enough space to show them in their fully preferred widths



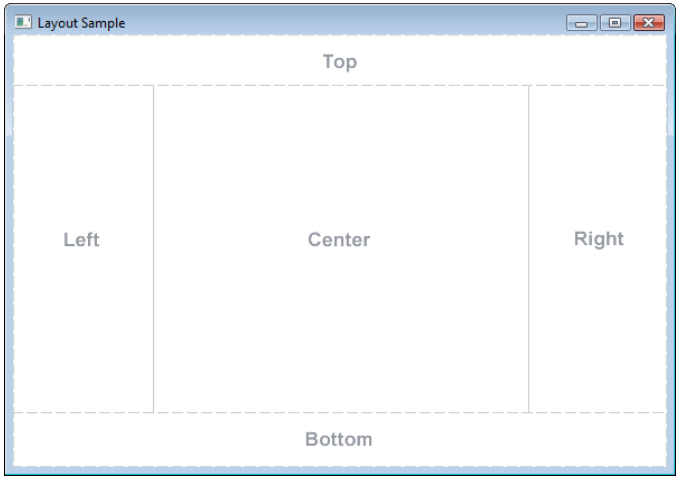
Layout Panes - Containers (cont.)

- VBox – arranges its content nodes vertically in a single column – on top of each other

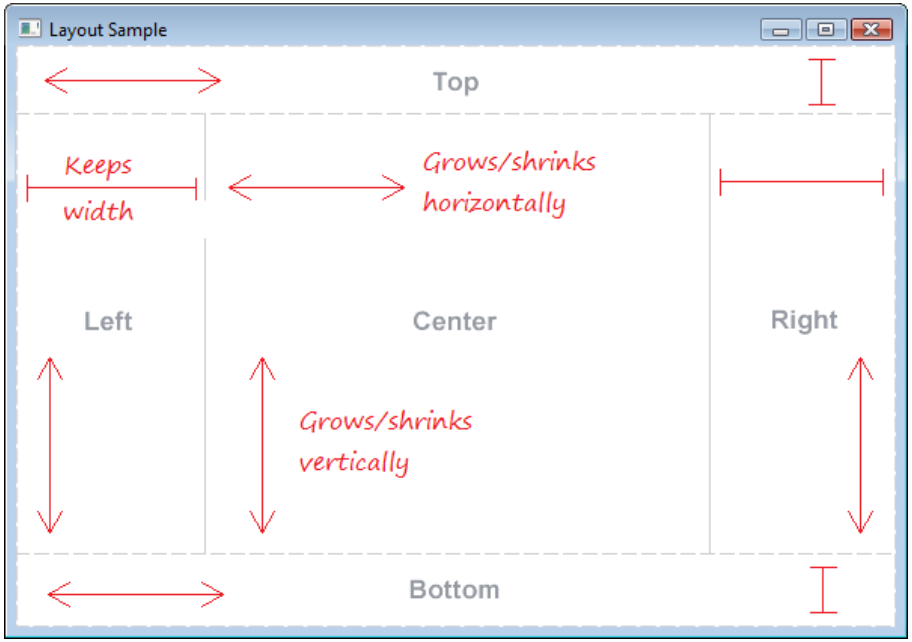


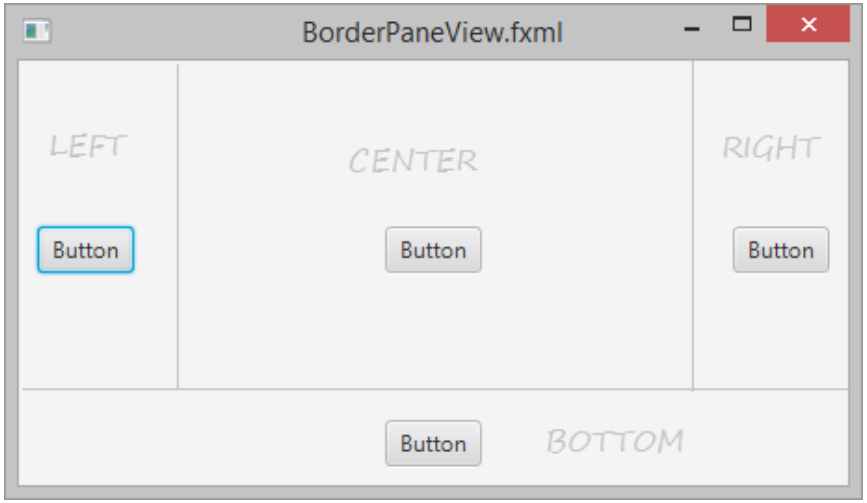
Layout Panes - Containers (cont.)

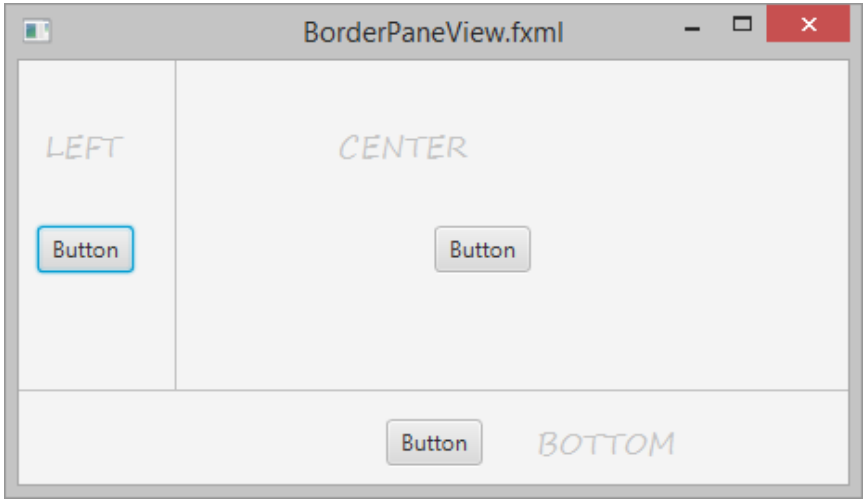
- BorderPane – pane divided into 5 separate areas, each of area can contain a subcomponent
  - Top/Bottom area: Can shrink/expand horizontally and keep the height unchanged.
  - Left/Right area: Can shrink/expand vertically and keep the length unchanged.
  - Center area: Can shrink/expand in both directions



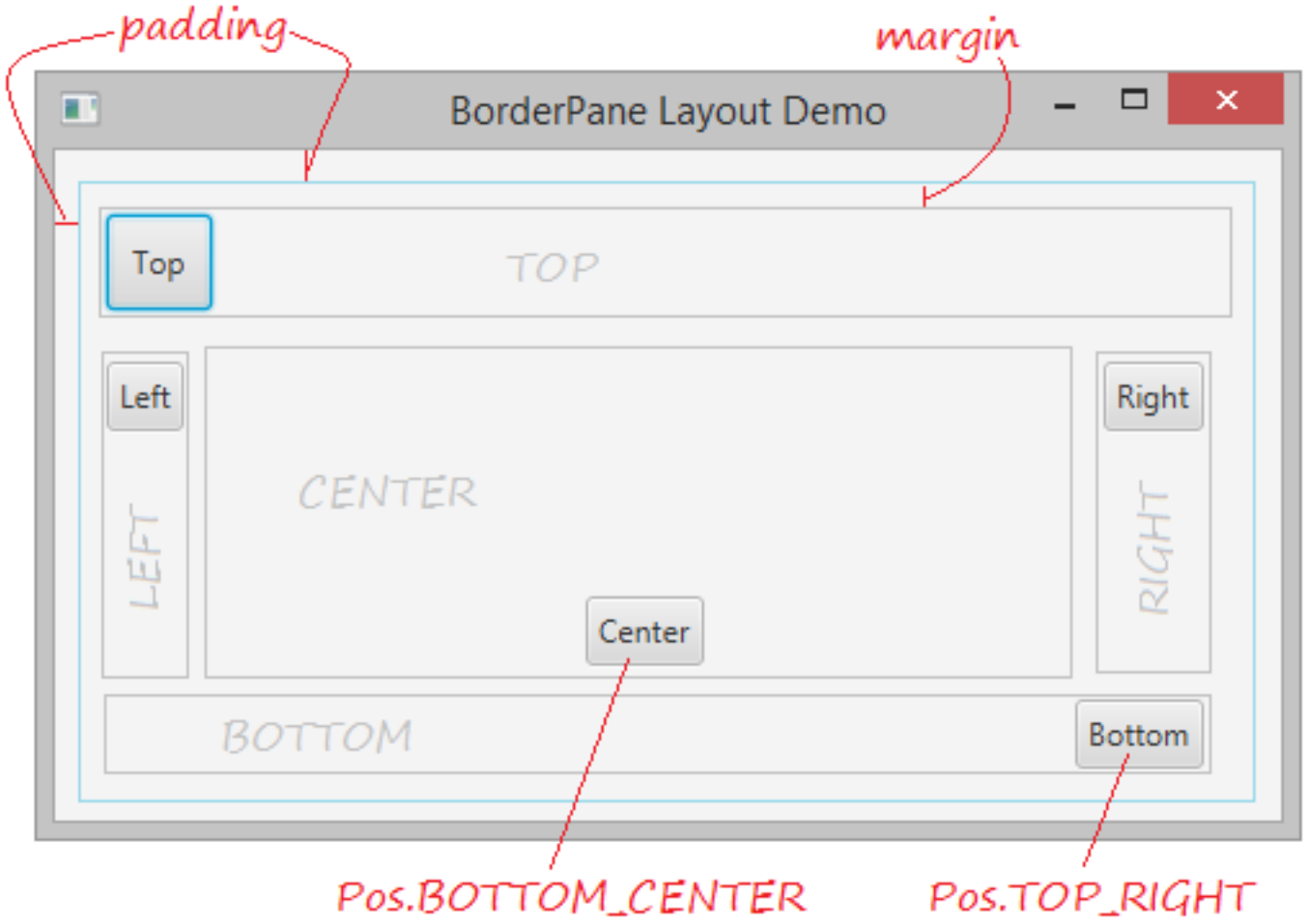
Layout Panes - Containers (cont.)





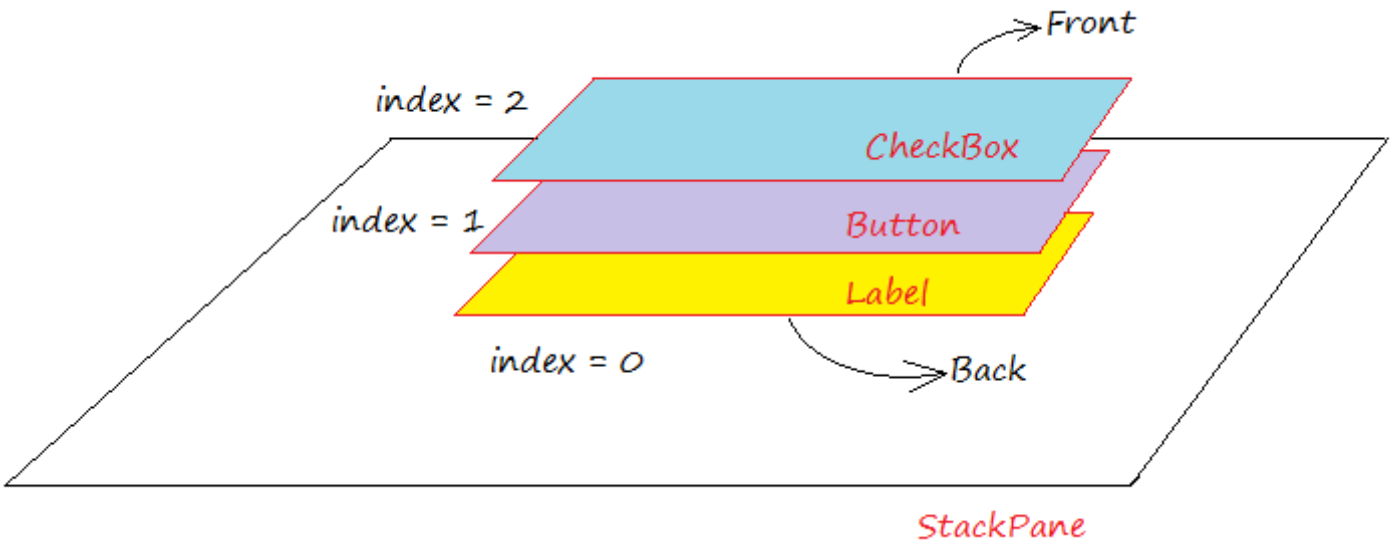


Layout Panes - Containers (cont.)



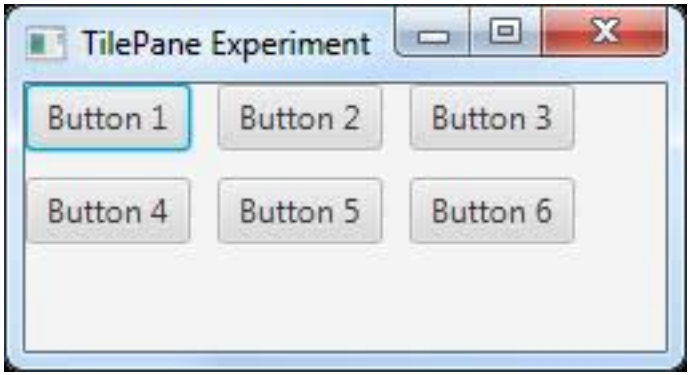
Layout Panes - Containers (cont.)

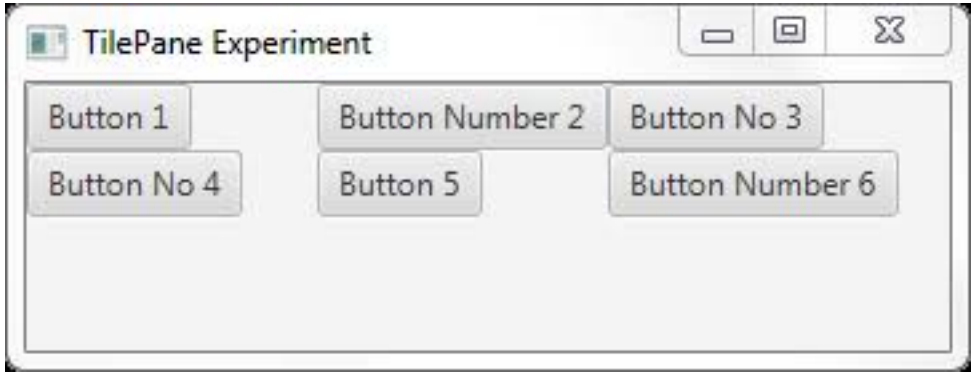
- StackPane – contains different components, subcomponents stacked up to others, and at a certain moment, you can only see the subcomponent lying on the top of Stack



Layout Panes - Containers (cont.)

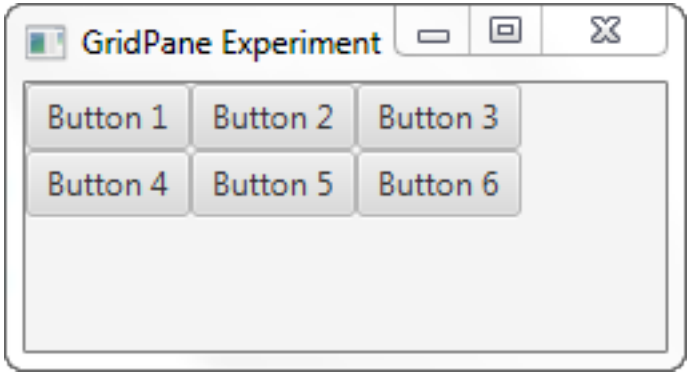
- TilePane – lays out its child components in a grid of equally sized cells

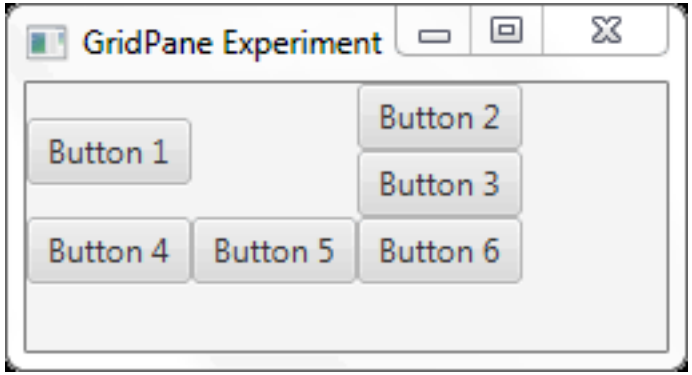




Layout Panes - Containers (cont.)

- GridPane – lays out its child components in a grid; the size of the cells in the grid depends on the components displayed in the GridPane; all cells in the same row will have the same height, and all cells in the same column will have the same width; different rows can have different heights and different columns can have different widths



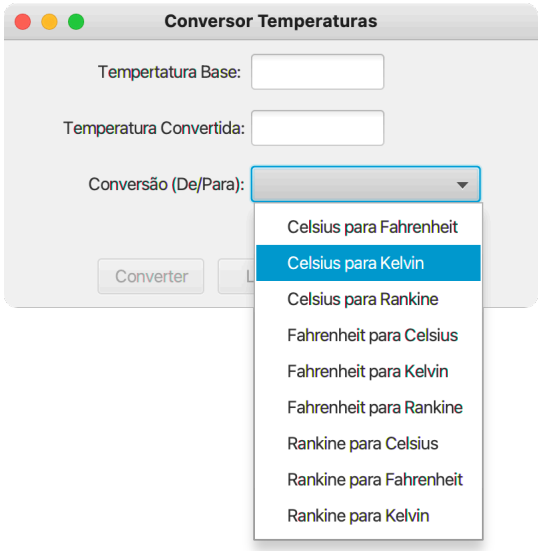


Conclusion

- JavaFX allows the creation of GUI
- Employes the theater metaphor for implementation (Stage andScene)
- With SceneBuilder the implementation of it's MVC model is simple by using FXML and Controllers
- Has a set of componentes (Nodes) that include layout pane to manage how the controls are displayed.

Conclusion (cont.)

- Let's put it all together ...



References

- JavaFX website: http//javafx.com
- JavaFX API https://docs.oracle.com/javase/8/javafx/api/toc.htm
- Open source project http://openjdk.java.net/projects/openjfx/
- Oracle Premier Support for Software http://www.oracle.com/us/support/software/premier/
- Blogs
  - http://fxexperience.com
  - http://blogs.oracle.com/javafx
  - https://o7planning.org/11009/javafx
  - http://tutorials.jenkov.com
  - https://www3.ntu.edu.sg/home/ehchua/programming/index.html