GUIs and Event Handling: Part One

Introducing JavaFX

Text

Buttons

Text Fields

Layout Panes

Online Tutorials/Resources

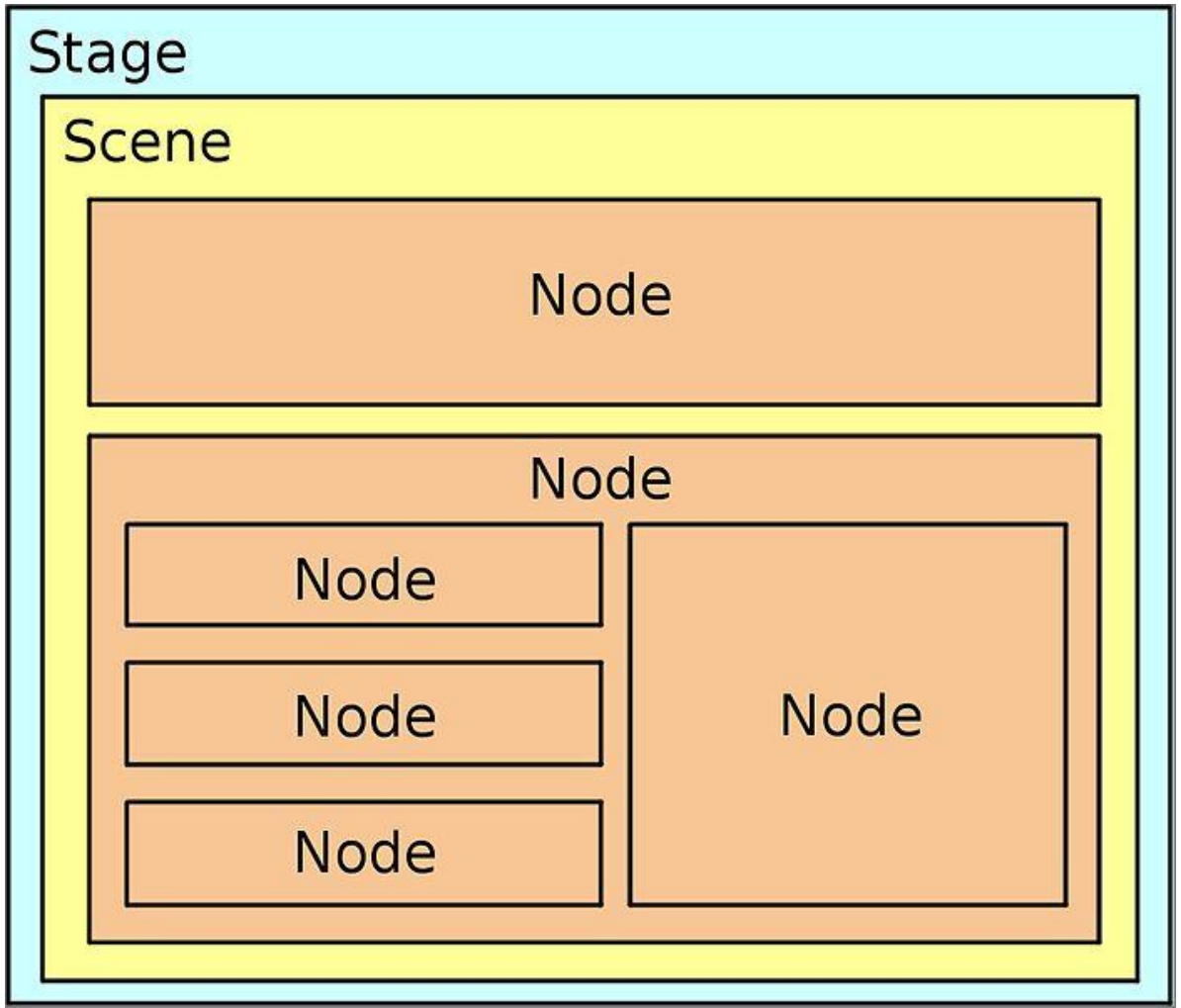
- https://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm
- https://www.tutorialspoint.com/javafx/index.htm
- http://tutorials.jenkov.com/javafx/index.html

INTRODUCTION

Java GUIs and Graphics

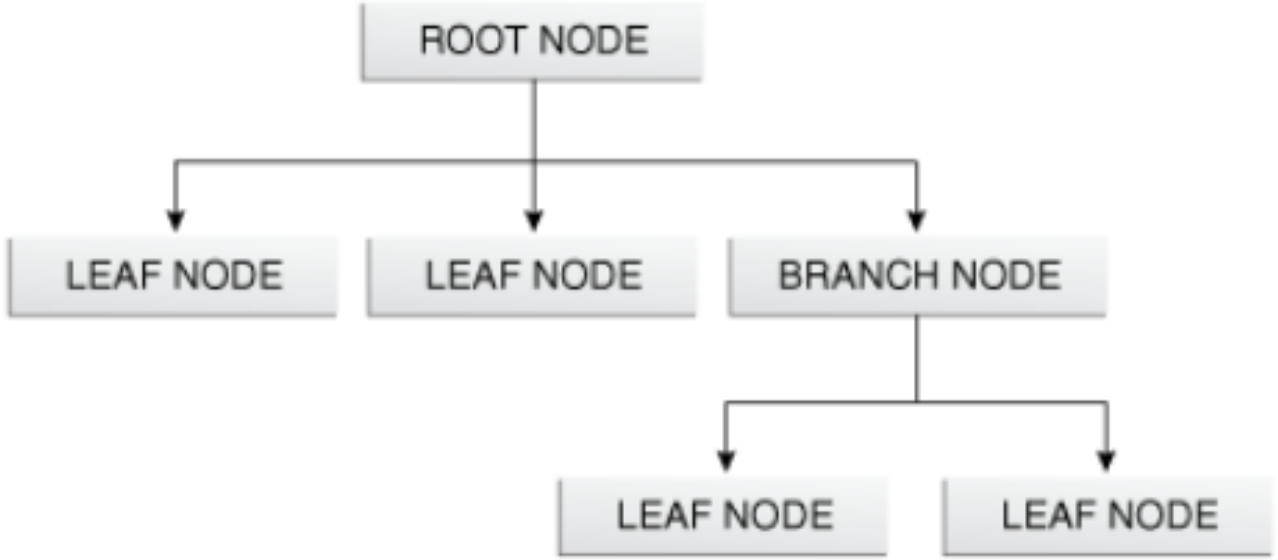
- Began with abstract windowing toolkit (AWT)
- Advanced to the Swing API
- Now JavaFX
  - Graphics API to develop rich desktop and web applications
  - API: https://docs.oracle.com/javafx/2/api/index.html

Structure of JavaFX Application



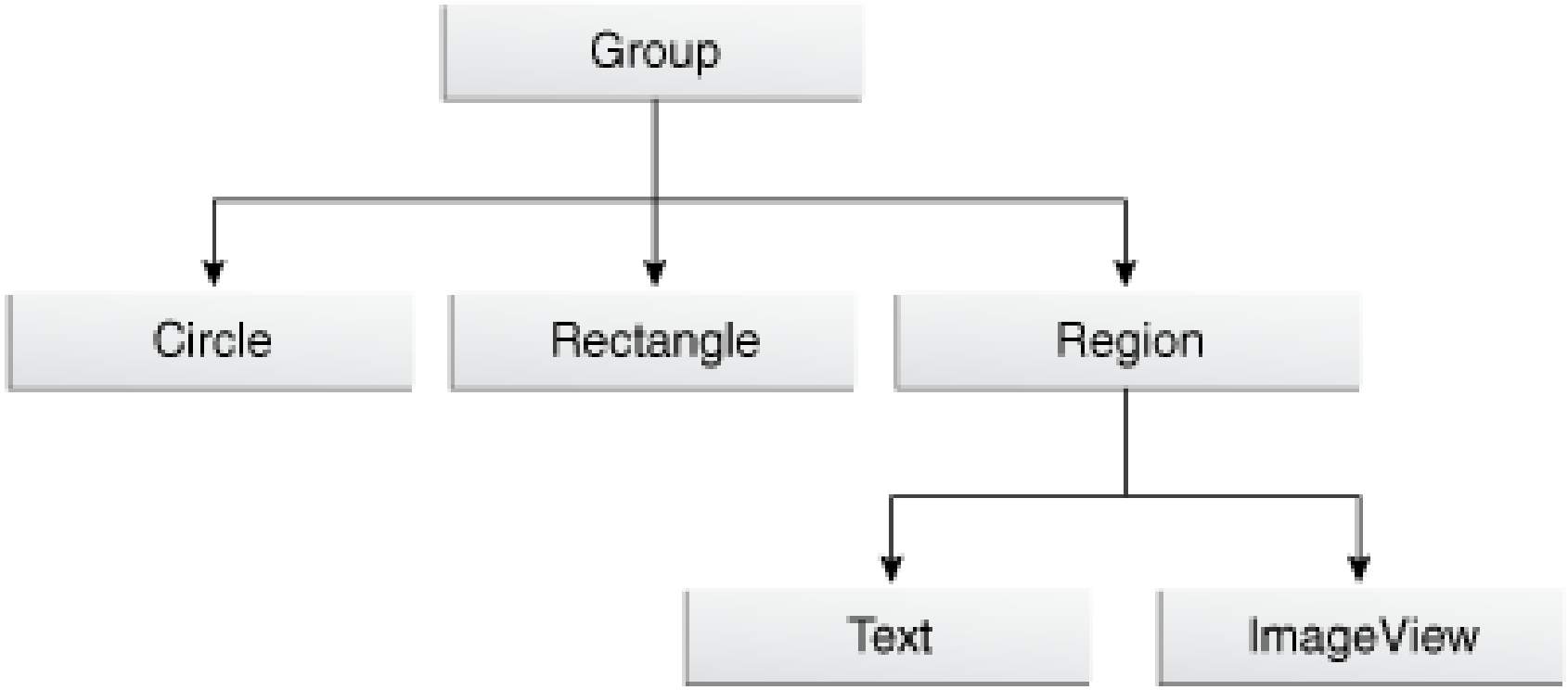
The Scene Graph

- A scene graph is a tree structure
- Each individual element in the scene graph is a *nodes*



images from https://docs.oracle.com/javafx/2/scenegraph/jfxpub-scenegraph.htm

Example Scene Graph



Structure of JavaFX Application

- Each application has a *stage.* The *stage* is the window.
  - Applications can have more than one stage!
- Each stage has a scene.
  - Stages can have more than one scene!
- The scene is represented by the scene graph.
  - Thus, each scene has a *root node*.

Structure of JavaFX Class

- Your class extends Application (in the javafx.application package)
- Your class has at least two methods:
  - `public void start(Stage primaryStage) { … }`
  - ```
    public static void main(String[] args) {
            launch(args);
    }
    ```
    - The main method might not be needed in some IDEs.
    ```

The Stage and the Scene

- Stage (the window)
  - Methods:
    - setTitle(String title)
    - setScene(Scene scene)
    - show()
- Scene
  - Constructors:
    - Scene(Node rootNode)
    - Scene(Node rootNode, double width, double height)
    - Scene(Node rootNode, double width, double height, Color backgroundColor)
  - Method: setFill(Color color)

Nodes

- Each node has zero or more children.
- Each node except the root has exactly one parent.
- Each node has a style class, opacity, effects, transforms, and event handlers.

- The `Node` class is at the top of the hierarchy for all nodes
- The `Parent` class inherits from `Node`.
  - Classes such as `Control`, `Group`, and `Region` inherit from `Parent`.
  - Method: `getChildren()` returns an `ObservableList<Node>` objects

Nodes

- `Group` constructor takes any number of `Node` objects
- `Node` has many useful descendent classes:
  - Controls: Button, Button, ScrollBar, Checkbox, RadioButton
  - Shapes: Text, Circle, Ellipse, Line, Polygon, Rectangle
  - ImageView, MediaView
  - Regions: Pane, BorderPane
  - Many more!

The Text Class

- Constructors:
  - Text(double upperLeftX, double upperLeftY, String text)
  - Text(String text)
  - Note: text can contain \n characters
- Methods:
  - setFont(Font newFont)
  - setFill(Color newColor)

The Font Class

- You can change the fonts of Text or Button objects
- Font constructors:
  - Font(double size)
  - Font(String fontFamilyName, double size)
    - Examples: Arial, Courier New, Helvetica, Garamond, Times New Roman
- Four static Font methods to customize weight (e.g., FontWeight.BOLD) and posture (FontPosture.ITALIC or FontPosture.REGULAR)
  - Font.font(String fontFamily, double size)
  - Font.font(Sttring fontFamily, FontPosture posture, double size)
  - Font.font(String fontFamily, FontWeight weight, double size);
  - Font.font(String fontFamily, FontWeight weight, FontPosture posture, double size)
- See available font families:
  - System.out.println(javafx.scene.text.Font.getFamilies());

Color

- **Many** Color constants are provided
- You can also create a new Color with static methods that specify the red, green, and blue makeup of the color:
  - Color.rgb(int red, int green, int blue); // int between 0 and 255
  - Color.color(double red, double green, double blue); // RGB percentages
- For shapes:
  - *stroke* is the outline
  - *fill* is the interior

- Note: you want the Color class in the javafx.scene.paint package!

Before we begin…

- First, download and run the TestMyFX.java program to make sure that JavaFX works on your machine!
  - If it doesn’t, pause the video, follow the steps on the next two slides, then return to the video.

JavaFX

- First, make sure you have Java 8.
  - Windows: open a command window (type “cmd” into the search box for Windows) or Mac: open a terminal window (on a mac)
    - type: java –version
    - Make sure it’s version 1.8 or higher.
  - If you have a lower version, upgrade. Make sure you get the JDK (not the JRE). http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Follow the next slide to add the JavaFX plugin to eclipse.
- Or use another resource:
  - This website walks through installing Java8 and then setting up both NetBeans and eclipse for JavaFX: https://www.tutorialspoint.com/javafx/javafx_environment.htm
    - Note: for eclipse, there is one mistake after you add the URL- click “Next,” not “Add.”
  - This video also instructions for eclipse without the plugin: https://www.youtube.com/watch?v=ejx3VxuIc8w

The eclipse JavaFX Plugin (e(fx)clipse)

- Open eclipse
- Help > Install New Software
- Click “Add…” in the top right
  - Enter Name: e(fx)clipse
  - Enter Location: http://download.eclipse.org/efxclipse/updates-released/2.3.0/site/
- Select both the install and single components boxes
- Click Next and the follow the on-screen prompts (clicking next, agree, and finish). The install takes a couple minutes.
- You’ll then be prompted to restart eclipse.

How to Begin

- Have your class extend Application
- Implement your start method
  - Create a root node (e.g., Group, Pane, etc.)
  - Create a scene with this root
  - With your stage: set the scene, set the title, and show it!

  - Remember: Stage -> Scene -> Root node

- Launch at this point to make sure you get a blank window before continuing on to create your nodes!

```java
import javafx.application.*;
import javafx.scene.*;
import javafx.scene.paint.*;
import javafx.stage.*;

public class FXShell extends Application {

    // instance data variables (including controls)

    @Override
    public void start(Stage primaryStage) {
        Group root = new Group();

        Scene scene = new Scene(root, 300, 300, Color.BEIGE);

        primaryStage.setTitle("TITLE");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

Important Note!!

- **Be sure to always import the classes/packages from javafx!**
- Many classes have the same name but are in other packages! You do not want these!
  - Example: java.awt.Color and javafx.scene.paint.Color
  - Example: java.awt.event.MouseEvent and javafx.scene.input.MouseEvent

Practice

- Create a HelloWorldFX program to display a basic hello message.

Beyond the Basics

- CSS
  - Cascading style sheets: describes how elements are to be displayed; the “look and feel”
  - You can incorporate CSS into JavaFX programs
    - add a style sheet to a scene
    - set the style of a node
  - https://docs.oracle.com/javafx/2/css_tutorial/jfxpub-css_tutorial.htm
  - https://www.tutorialspoint.com/javafx/javafx_css.htm
- Scene Builder
  - A visual layout tool that lets you design GUIs without coding
  - Supported by both NetBeans and eclipse
  - http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html

EVENT HANDLING

GUIs

- Standalone, text-based programs are run through the main method.
  - They might execute fully without interacting with the user.
  - They might interact with the user through the console.
- GUI programs are *event-driven*.
  - The program is run and then just waits for the user to interact with the user. The program then responds to the user's action.

Event Handling with JavaFX

- Control
  - A screen element that displays information to the user or interacts with the user
  - Examples: button, text field, image
- Event
  - An action taken by the user
  - Examples: the clicking of a button, the movement of a mouse
- Event Handler
  - An object that contains a method that is called when an event occurs
  - Examples: take the user's inputted text and display it back, perform a calculation,

Button Class

- A Button object (the *control*) generates an *event* of type ActionEvent when it is pushed.
- Constructor: takes the text to be displayed on the button.
- Methods:
  - setOnAction- specifies the *event handler*
  - setDisable(boolean)

Specifying the Event Handler

- Preferred method: Java 8's :: operator and a processing method placed inside the class.

```
myButton.setOnAction(this::processButton)
public void processButton (ActionEvent event) { … }
```

- Alternative method: Java 8's lambdas:

```
myButton.setOnAction( (event) -> { … } ) ;
```

- Older method (pre-Java 8):

```
myButton.setOnAction(new ButtonHandler());
private class ButtonHandler implements
                        EventHandler<ActionEvent> {
   public void handle(ActionEvent event) { … }
}
```

Practice

- Modify the HelloWorld program to add a button.
  - When the user clicks the button, change the text displayed on the label.
  - Disable the button after it is clicked.
- Write a program to keep track of and display the number of times a button is clicked.
  - Change the background color of the window a new random color with every fifth click.

TextField Class

- Text fields allow the user to input a line of text
- A TextField object (the *control*) generates an *event* of type ActionEvent when the user clicks “enter” from inside the text field
- Constructor: no parameters, or the initial text to display
- Methods
  - setOnAction method specifies the *event handler*
  - getText() returns the text that is in the text field
    - This can be invoked at any time- not just when the user clicks enter!
    - This always reads in text as a String!
    - To convert to numeric: Integer.parseInt(text);
  - clear() removes the text
- Use the same handling approach as for buttons!

Practice

- Create a HelloWorld program that asks the user to enter their name and displays it as part of the message.
  - The message should be changed when the user clicks “enter.”
  - Then, add a button and allow the user to change the display with either the button or by clicking “enter.”

Basic Approach for Controls

1. Declare the control
   - Inside the start method if only using inside the start method
   - As an instance data variable if using outside of the start method
2. Initialize the control inside of start
3. Customize the control (if necessary)
4. Add the control to the scene graph
5. Recommended: launch to view here before moving on!
6. Set the action of the control (if necessary)
   - Invoke the “setOnAction” method
   - Write the handler method

LAYOUT PANES

Layout Panes

- A container that displays nodes
- Layout pane classes are derived from the Pane class
- Layout panes can be nested together

- Constructors: send all nodes
- Or use the method: `getChildren().add(node);`

FlowPane

- Nodes are added horizontally (by default) or vertically in the order they were added
- Nodes wrap around when no more room (additional rows/columns created as needed)
- Methods:
  - setStyle(“-fx-background-color: color”); // css styling
  - setHgap(int)
  - setVgap(int)
  - setAlignment(Pos.CONSTANT); // e.g., Pos.BOTTOM_LEFT, Pos.CENTER, etc.

- Review the LayoutPane example

TilePane

- Similar to FlowPane except that the nodes are displayed in fixed-sized tiles (or cells)
  - The size is set to accommodate the largest node in the pane
- Methods:
  - setStyle(cssBackgroundColorString)
  - setHgap(int)
  - setVgap(int)
  - setAlignment(Pos.CONSTANT)

StackPane

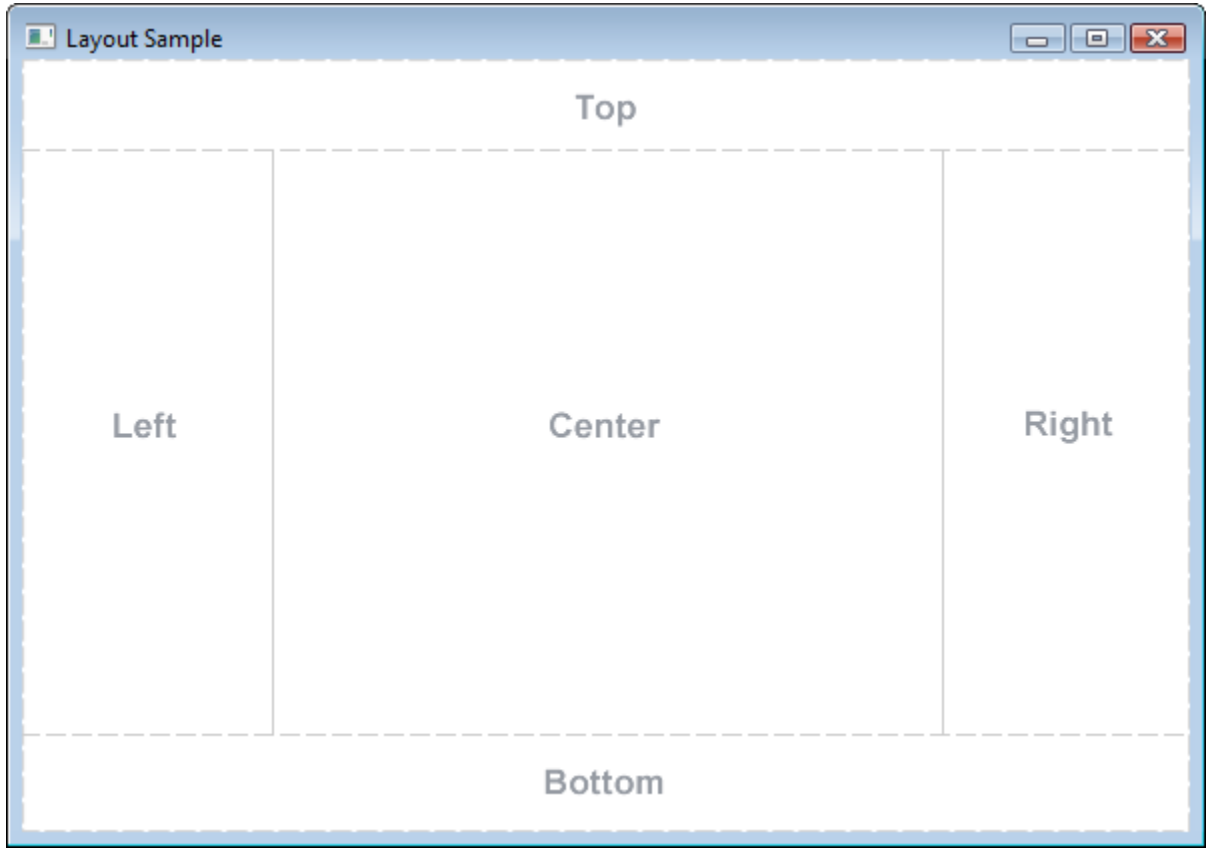
- Stacks nodes on top of each other in the order added
- Default alignment is center/center, but can be changed
- Often used for:
  - Overlaying shapes, pictures, text
  - Keeping nodes centered in their display area

HBox and VBox

- HBox displays nodes horizontally in one row
- VBox displays nodes vertically in one column
- There is no wrapping (this is different from FlowPane)
- Methods:
  - setPadding(Inset); // Inset constructor takes 4 numbers
  - setSpacing(double);
  - setAlignment(Pos.CONSTANT)

BorderPane

- Displays nodes in five areas
- Each area grows and shrinks as needed
- Methods: setTop, setLeft, setCenter, setBototm, setRight



GridPane

- Displays nodes into a flexible grid of rows and columns
- Add each node to a particular cell
  - Nodes can also span multiple cells
  - Row and column numbers begin at 0
- Methods:
  - add(node, column, row)
  - add(node, column, row, numColsSpan, numRowsSpan)
  - setGridLinesVisible(boolean)

  - setStyle(cssBackgroundColorString)
  - setHgap(int)
  - setVgap(int)
  - setAlignment(Pos.CONSTANT)

  - GridPane.setHalignment(node, HPos)
  - GridPane.setValignment(node, VPos)

Practice- Use Layout Panes!

- Write a GUI to allow the user to click buttons to increment or decrement a counter by 1.
  - Modify the program to allow the user to enter a new interval so that increment and decrement will use that interval (instead of 1).
- Write a GUI to allow the user to enter in a series of integer numbers.
  - A “calculate” button will display the total, min, max, and average of the numbers.
  - A “clear” button will allow the user to start over.