Objective

Layout:

Layout (Parent) elements that arrange elements inside them

Pane, Vbox, Hbox, BorderPane, GridPane, Gtroup, Region, …

JavaFX So Far

So far we have used setLayoutX() and setLayoutY() to manage elements on a Pane.

Positioning elements manually is slow and difficult

Here we learn layout Panes provided by JavaFX that make it much easier to organize and align the GUI elements

Layout

Layout classes arranges GUI components in a functional and attractive way.

Each type of Layout implements a specific arrangement that's useful in different use cases. We'll talk about 4 types of layouts.

1. VBox and HBox (these 2 are similar)
2. BorderPane
3. GridPane
4. Group

HBox and VBox

HBox and VBox are two simple layout panes that arrange their children in a vertical or horizontal row.

Three of their most useful constructors

```
HBox(Node... children)
```

```
HBox(double spacing, Node... children)
```

```
HBox(double spacing)
```

The children are the elements arranged inside the HBox/VBox and can also be added using getChildren/add

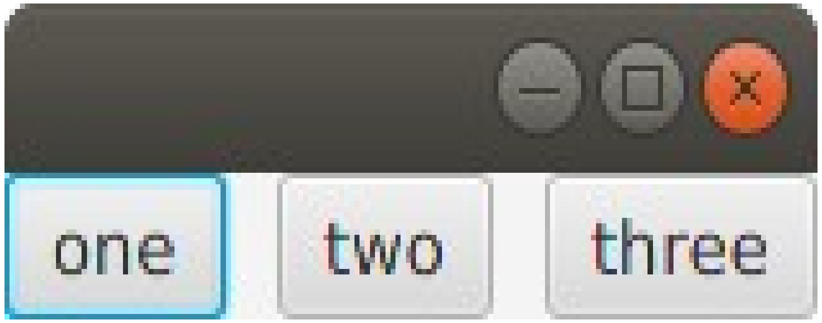
The spacing determines the space in between the child elements

HBox

HBox arrange elements in a single horizontal row.

```
Button b1 = new Button("one");
Button b2 = new Button("two");
Button b3 = new Button("three");
HBox hbox = new HBox(10); // margin between elements
hbox.getChildren().addAll(b1,b2,b3);
```

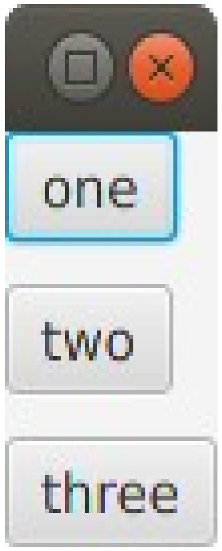
Documentation



VBox

VBox arrange elements in a single column.

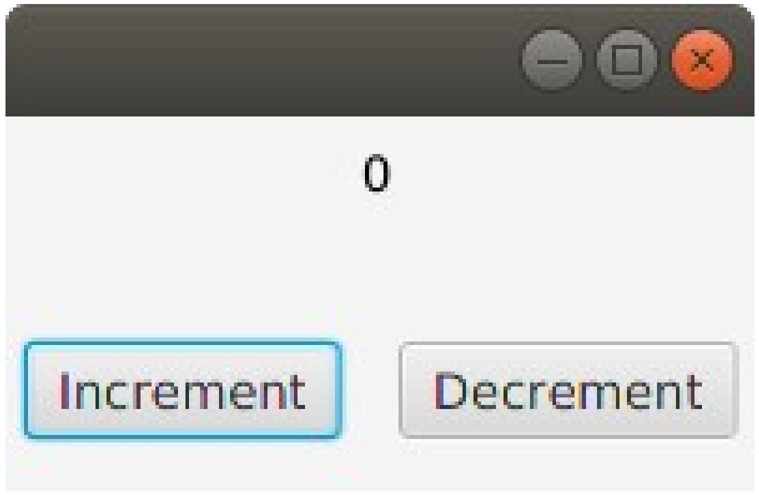
```
Button b1 = new Button("one");
Button b2 = new Button("two");
Button b3 = new Button("three");
VBox hbox = new VBox(10); // margin between elements
hbox.getChildren().addAll(b1,b2,b3);
```



Documentation

Using Layouts

- Let’s use the JavaFX’s layout tools to improve a couple examples from last lecture.
- We’ll start with the increment/decrement application.



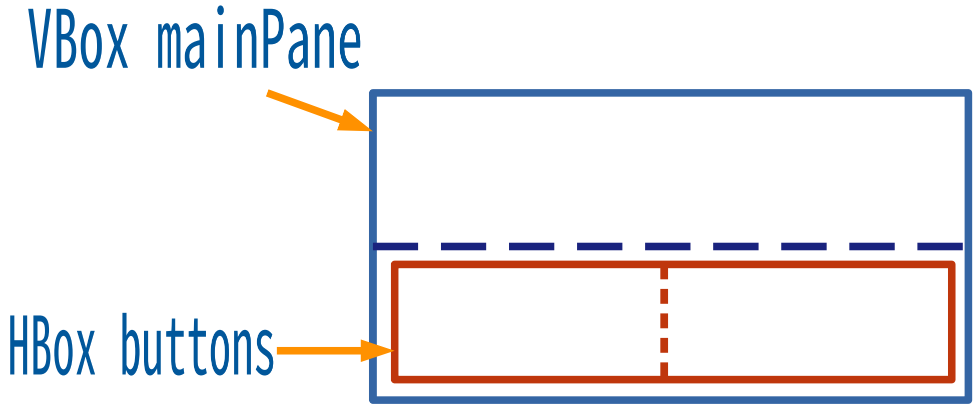
Combined Layout

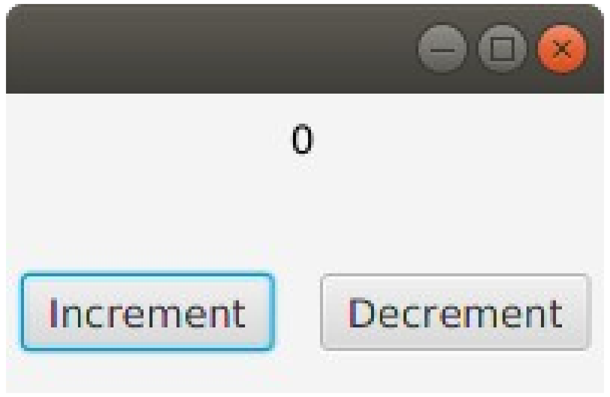
Each of the layout elements can be inserted into another layout.

Lets use our previous example inc/dec that we manually placed elements inside Pane, and arrange elements using layout.

Combined Layout

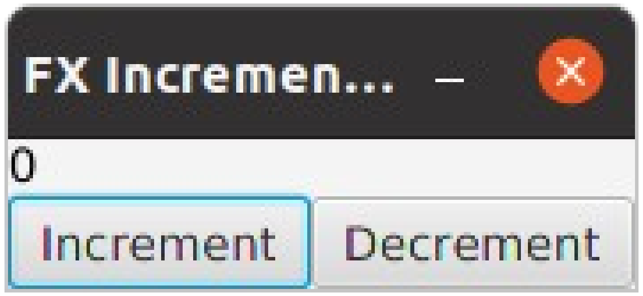
We can combine VBox and HBox to arrange the elements:

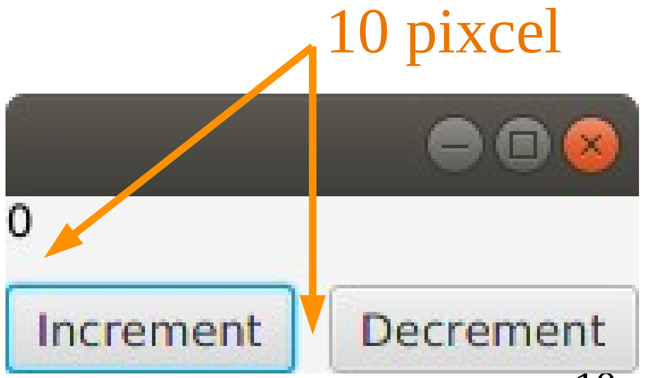




Increment/Decrement Example

```
public void start(Stage primaryStage) {

valueText = new Text(""+val); // do not need x, y coordinates
incButton = new Button("Increment");
decButton = new Button("Decrement");

IncListener listener = new IncListener();
incButton.setOnAction(listener);
decButton.setOnAction(listener);

// create HBox holding buttons
HBox buttons = new HBox(10,incButton,decButton);
// adding text and the buttons (HBox) to the mainPane(VBox)
VBox mainPane = new VBox(10,valueText,buttons);
// add vbox to the scene
Scene scene = new Scene(mainPane);

primaryStage.setScene(scene);
primaryStage.show();

}
```





Scene Size

Notice that we didn’t set the size of the scene

```
Scene scene = new Scene(mainPane);
```

The scene’s size is set based on the root element which sizes itself based on its child nodes and layout

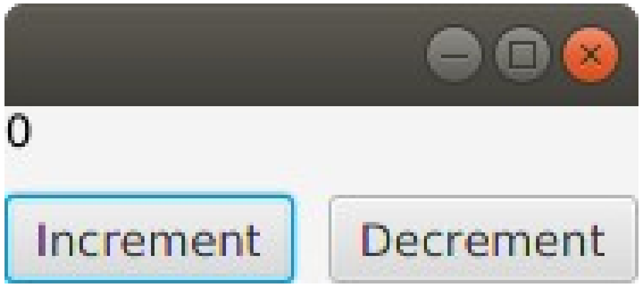
Increment/Decrement Example

The buttons are now in the right position, but

- The value in the top would look better centered
- The space in between the buttons is good, but this would look better with space around the outside of the buttons and the value.

We will set the alignment to control whether elements are positioned horizontally/vertically in their areas

We will use the Insets class to specify margins/padding around elements



Increment/Decrement Example

The setAlignment method of HBox/VBox controls the alignment of the contained elements

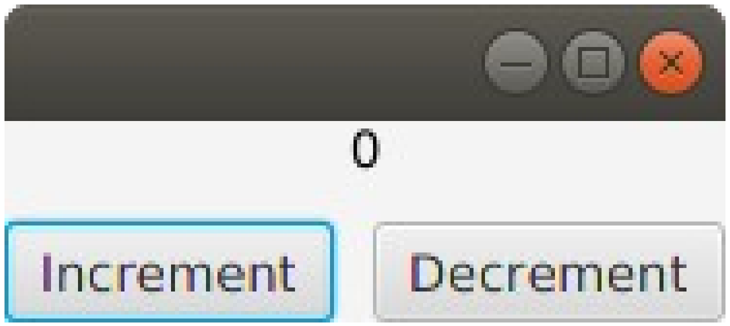
```
setAlignment(Pos value)
```

Pos is an enum containing alignment values

TOP_LEFT, TOP_CENTER, TOP_RIGHT, CENTER_LEFT, CENTER, etc

If we call this for the VBox

```
mainPane.setAlignment(Pos.CENTER);
```



Insets

The insets class allows us to specify the amount of space to add on the different sides of an element

There are two constructors, one where you set the size for each side individually, and one where you set all with one value

```
Insets(double top, double right,
       double bottom, double left)
Insets(double allSides)
```

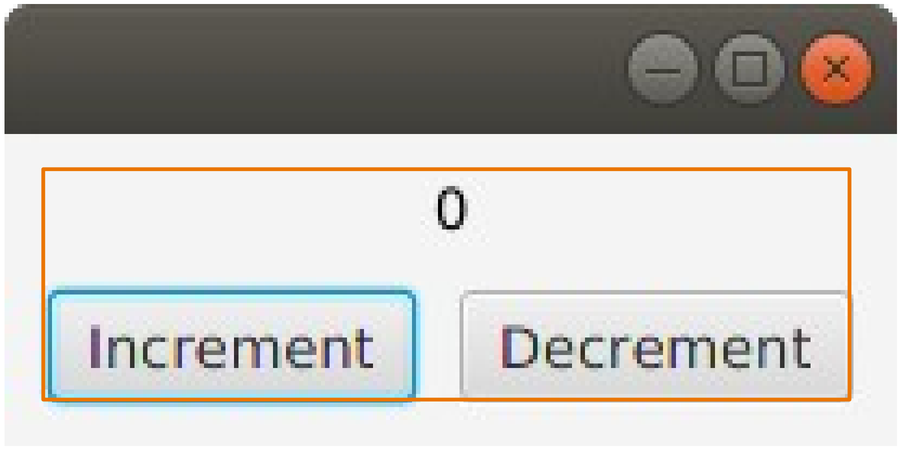
Increment/Decrement Example

Use setPadding to set the space around the edge of the layout pane

`setPadding(Insets value)`

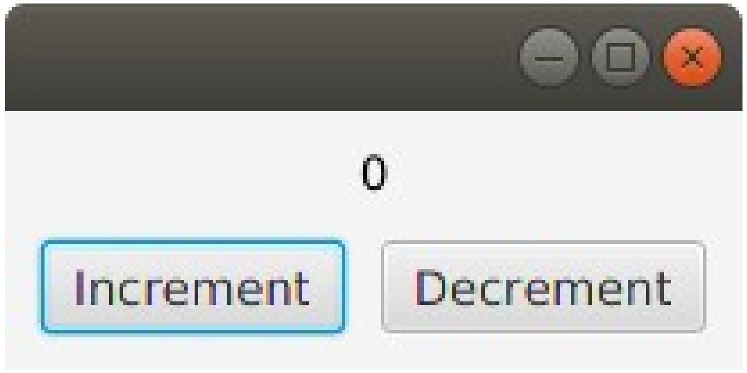
If we call this for the VBox

`mainPane.setPadding(new Insets(10));`



Final Version

```
....
// create HBox holding buttons
HBox buttons = new HBox(10,incButton,decButton);
// adding text and the buttons (HBox) to the
mainPane(VBox)
VBox mainPane = new VBox(10,valueText,buttons);

mainPane.setAlignment(Pos.CENTER);
mainPane.setPadding(new Insets(10));

// add vbox to the scene
Scene scene = new Scene(mainPane);
.....
```



Refer to IncrementDecrementLayoutFX.java

Application Resizing

One issue that we are going to avoid is the resizing of applications.

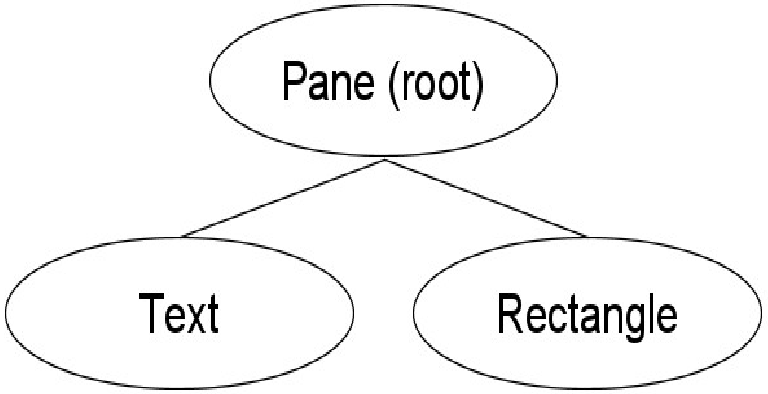
For this course, we will try to create a quality layout for a specific size and stop there.

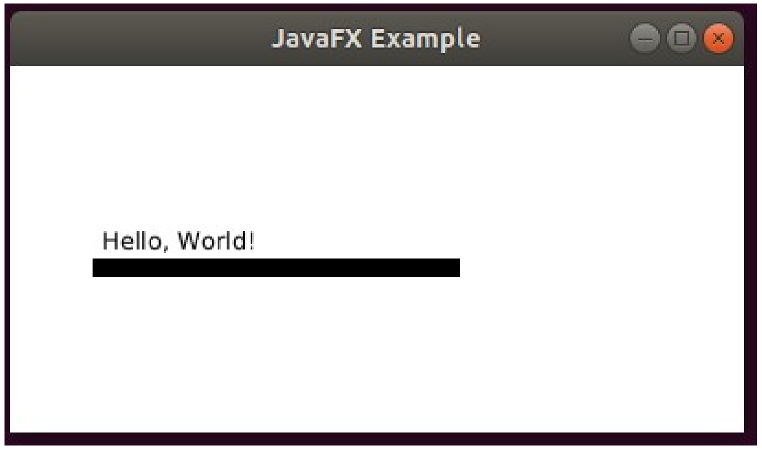
To prevent the application from being resized, we can call setResizable(boolean) for the stage

```
primaryStage.setResizable(false);
```

Scene Graph (review)

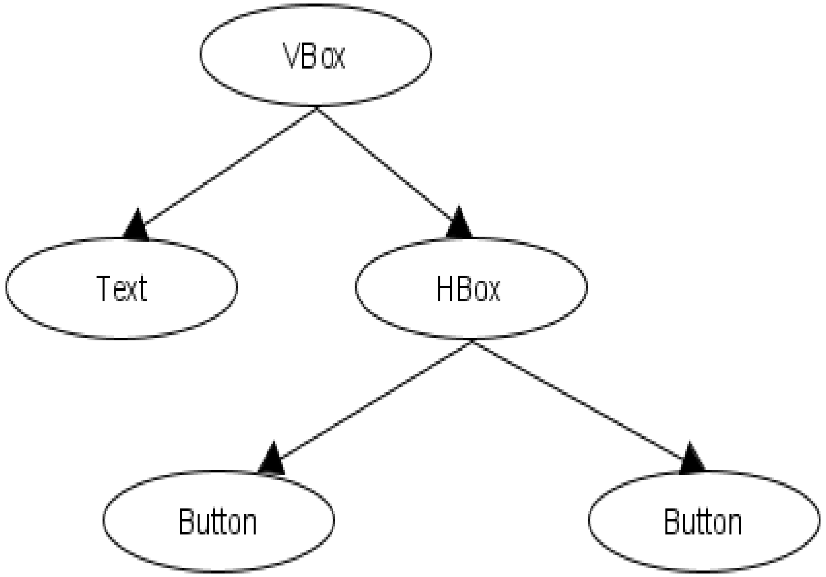
- Each element is a node in the tree.
- The root has no parents, all other nodes have one parent.
- Nodes that have children are parents.
- For now we will just add more children to the root element, but more complex layouts will lead to a more complex tree.

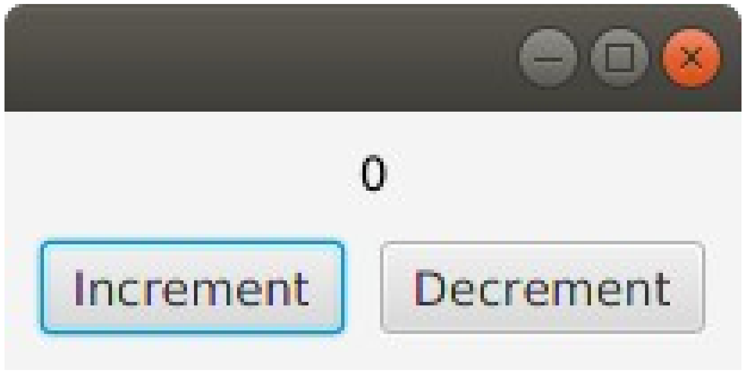




Scene Graph

- We haven’t looked at a scene graph in awhile.
- All of the scene graphs before now just have a bunch of children of the root node.
- Now that we are using layout panes, the graph will get more interesting
- The tree (because it really is a tree) will get deeper.

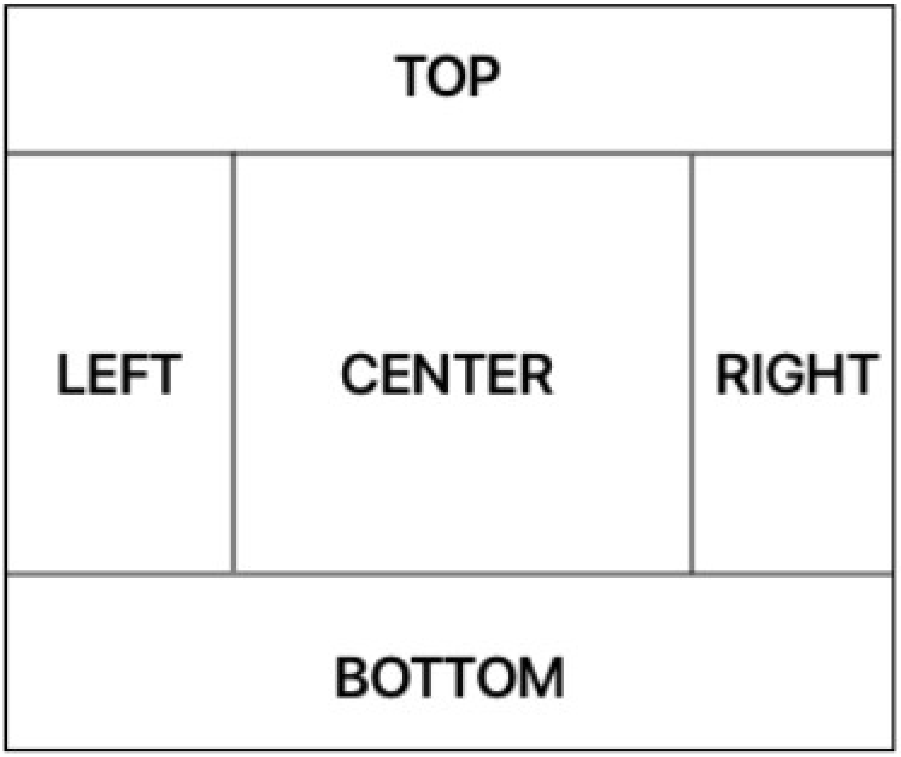




BorderPane

BorderPane breaks itself up into 5 areas. Components are placed toward areas of a container TOP, RIGHT, BOTTOM, LEFT, or CENTER.

Not all areas need to be given a node. They can be left empty.

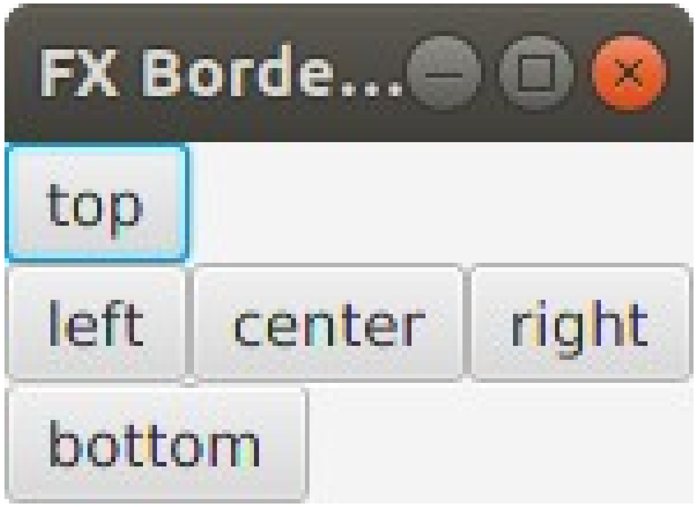


Example: BorderPane

```
Button top = new Button("top");
Button bottom = new Button("bottom");
Button left = new Button("left");
Button center = new Button("center");
Button right = new Button("right");
```

```
BorderPane mainPane = new BorderPane();
mainPane.setTop(top);
mainPane.setBottom(bottom);
mainPane.setLeft(left);
mainPane.setCenter(center);
mainPane.setRight(right);
```

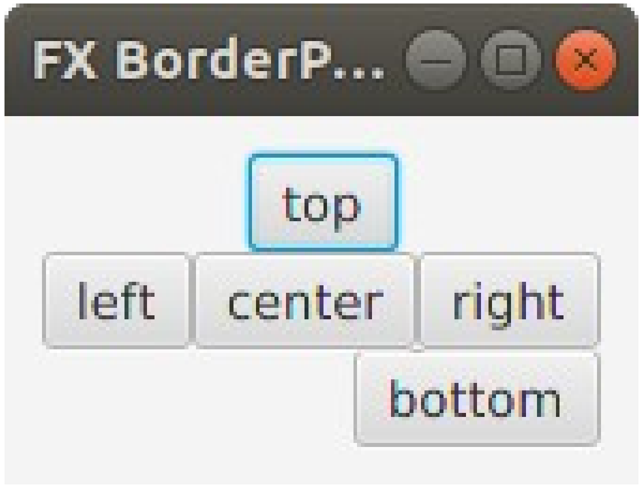
```
Scene scene = new Scene(pane);
```



Example: BorderPane

We can also align the elements in BorderPane:

```
mainPane.setAlignment(top, Pos.TOP_CENTER);
mainPane.setAlignment(bottom, Pos.BOTTOM_RIGHT);
mainPane.setPadding(new Insets(10));
```



Refer to BorderPaneFX.java

Self-Check

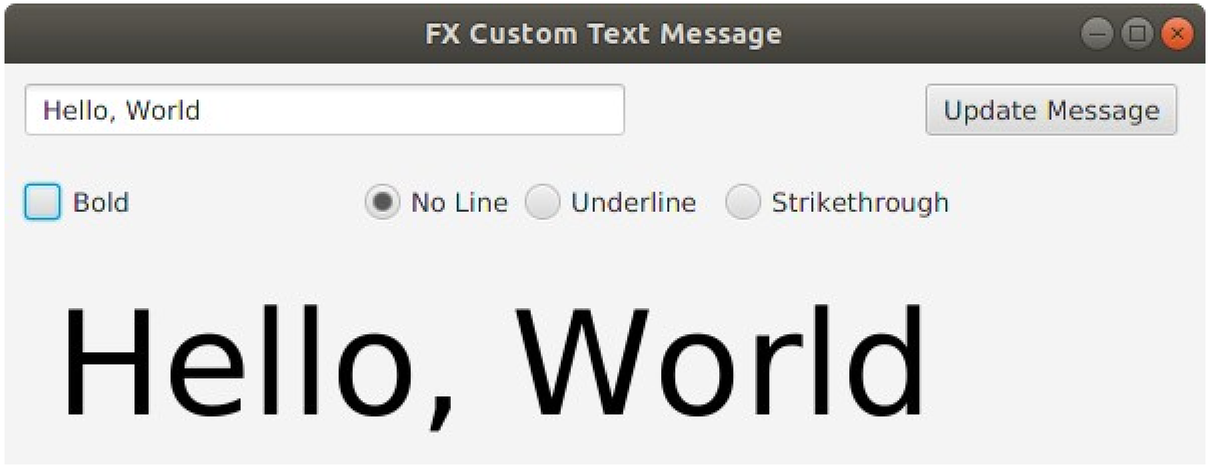
What do you think would happen if try to add 2 buttons to the same region in the BorderPane ?

```
pane.setTop(top);
pane.setTop(top2);
```

Using Layouts

Next, we’ll work with the message example from last time. Refer to : E7_MessageFX3.java

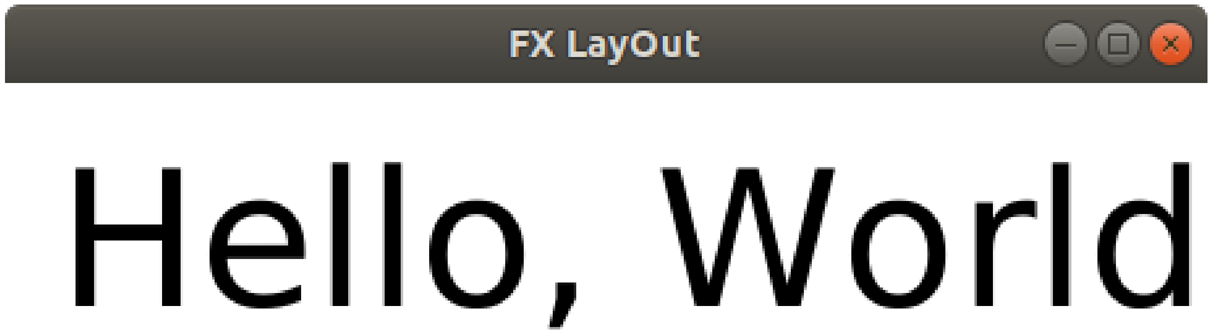
We will reorganize it with the message at the center and other components around it using BorderPane



Hello, World

Message Example

```
public void start(Stage primaryStage) {
  message = new Text(20,85,initMsg);
  message.setFont(Font.font("Arial",fontSize));
  Pane messagePane = new Pane();
  messagePane.getChildren().add(message);
  BorderPane root = new BorderPane();
  root.setCenter(messagePane);

  Scene scene = new Scene(root);
  ...
```


FX LayOut

Hello, World

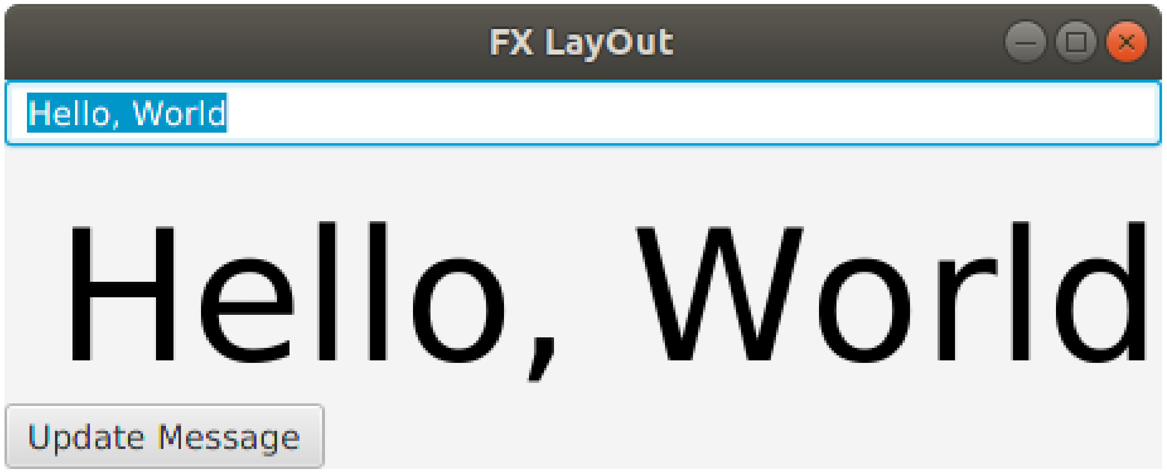
Message Example

We will start by placing the Text into its own Pane.

Then we will set the size of that pane, so we don’t need to worry if the Text element grows too large and stretches the other areas of the BorderPane

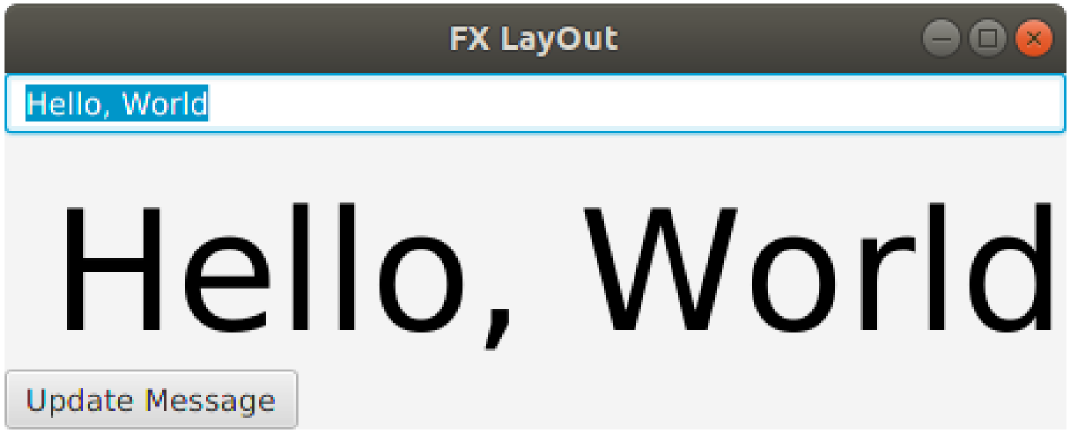
Message Example

- Let’s put the textbox in the top area and the button in the bottom area.



Message Example

```
public void start(Stage primaryStage) {
  ...
  BorderPane root = new BorderPane();
  root.setCenter(messagePane);
  textField = new TextField(initMsg);
  Button updateButton = new Button("Update Message");
  root.setTop(textField);
  root.setBottom(updateButton);

  Scene scene = new Scene(root);
  ...
```



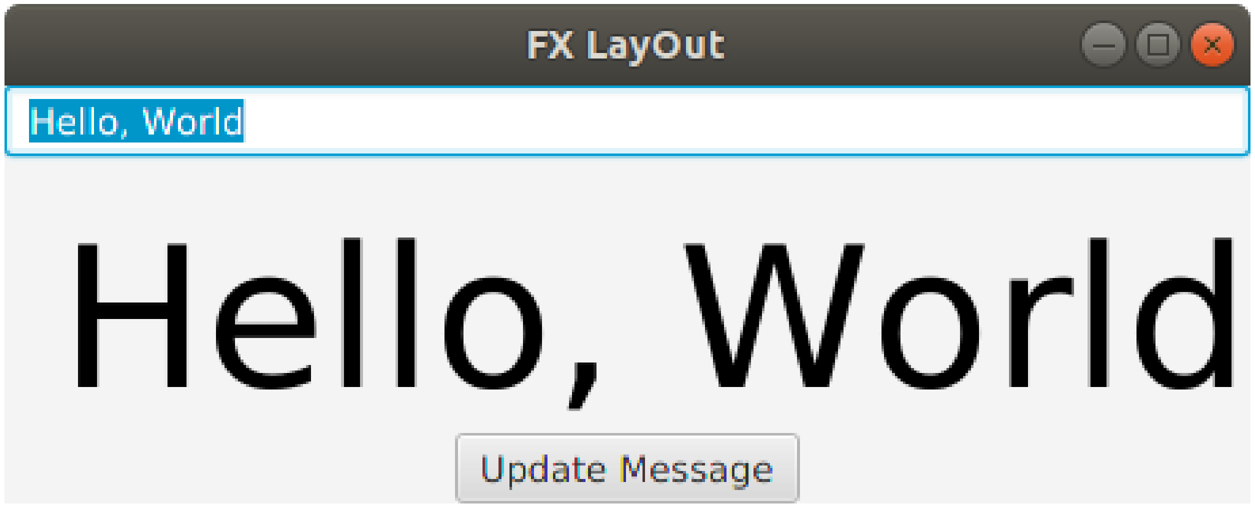
Message Example

The textbox looks fairly good, but let’s work on the button.

Center it

```
root.setAlignment(updateButton, Pos.CENTER);
```

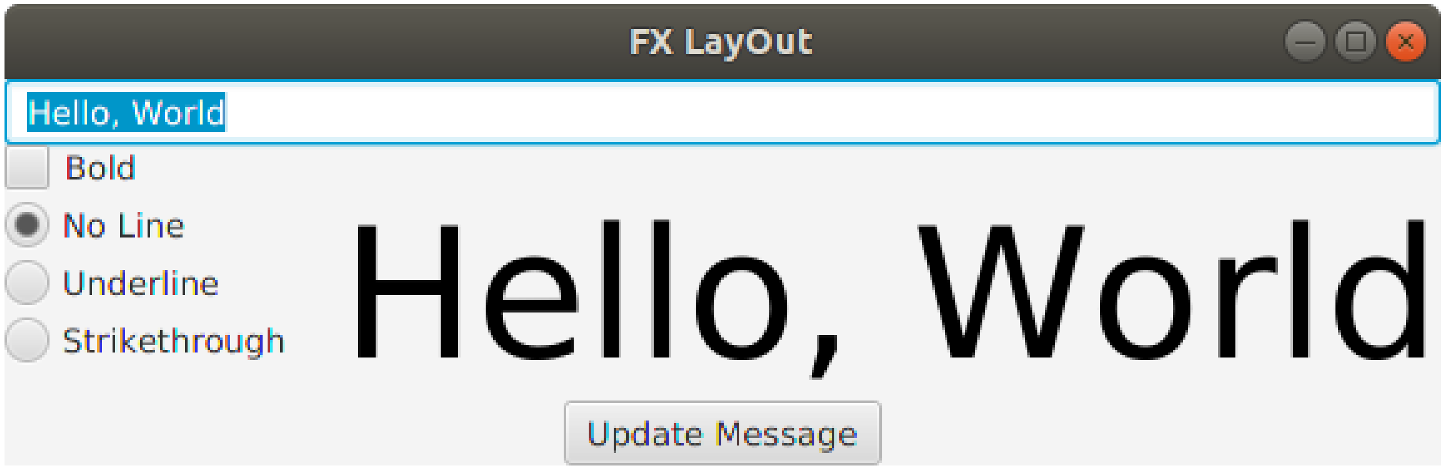
In this case the BorderPane controls how the elements in it are positioned, so we use it to set the child node’s alignment



Message Example

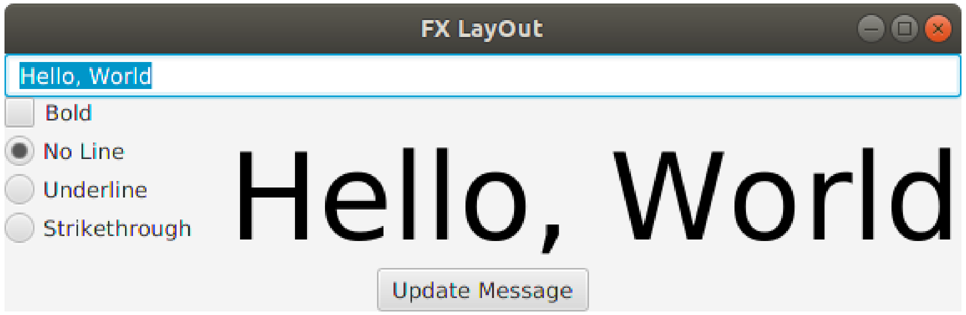
Next, let's make a left panel with the checkbox/radio buttons stacked

- Use a Vbox



Message Example

```
public void start(Stage primaryStage) {
  ...
  boldBox = new CheckBox("Bold");
  noLineRB = new RadioButton("No Line");
  underlineRB = new RadioButton("Underline");
  strikeThroughRB = new RadioButton("Strikethrough");

  buttonGroup = new ToggleGroup();
  noLineRB.setToggleGroup(buttonGroup);
  underlineRB.setToggleGroup(buttonGroup);
  strikeThroughRB.setToggleGroup(buttonGroup);
  buttonGroup.selectToggle(noLineRB);

  VBox leftSide = new VBox(5,boldBox,noLineRB,underlineRB,strikeThroughRB);
  root.setLeft(leftSide);

  Scene scene = new Scene(root);
  ...
```



Message Example

Now that the elements are in their basic location. Let’s improve things a little

`root.setPadding(new Insets(5));`

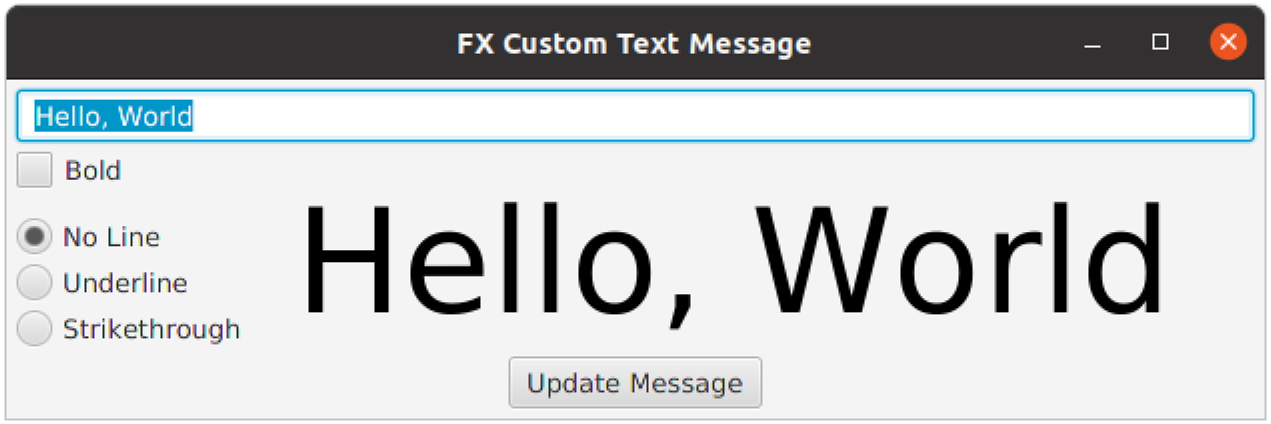
- Adds space around the entire BorderPane

`leftSide.setPadding(new Insets(5,5,5,0));`

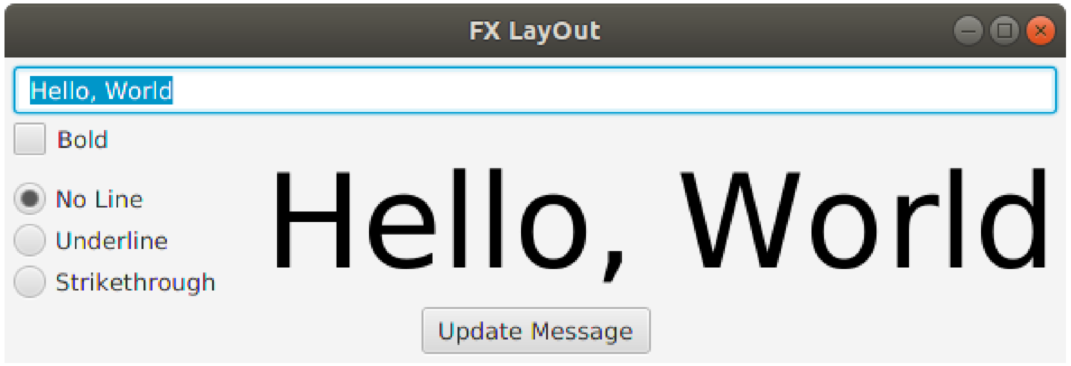
- Adds space around the VBox, The left side is 0 because the BorderPane padding is there.

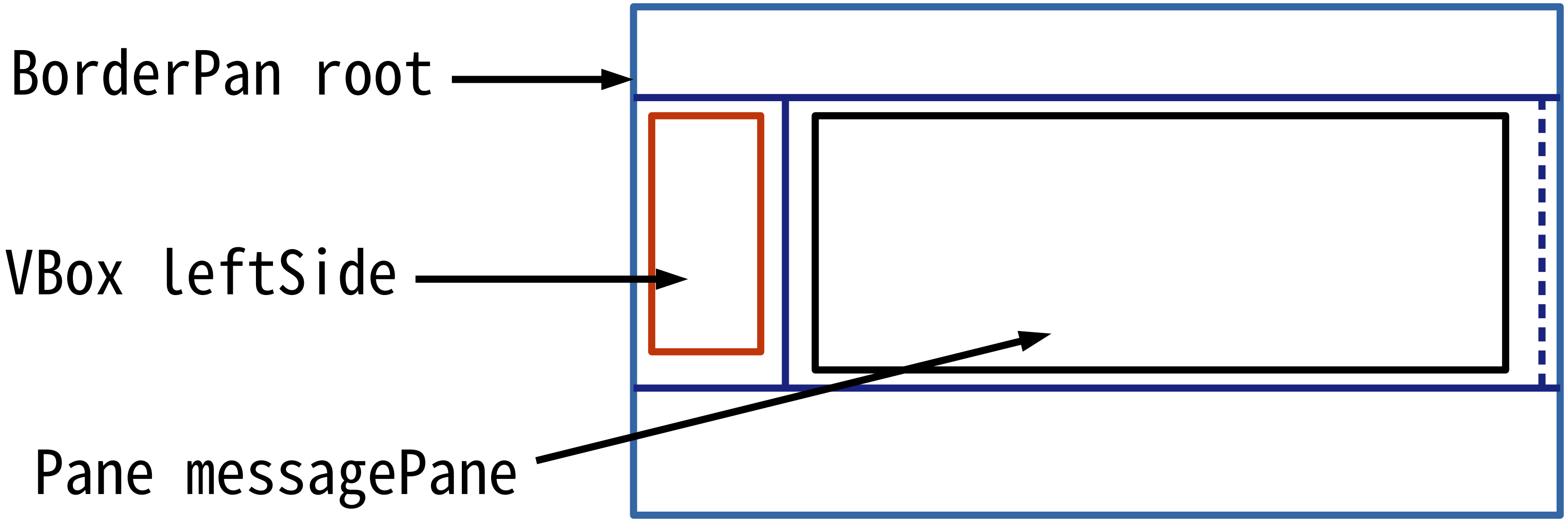
`VBox.setMargin(boldCheckBox, new Insets(0,0,10,0));`

- Adds space below the checkbox

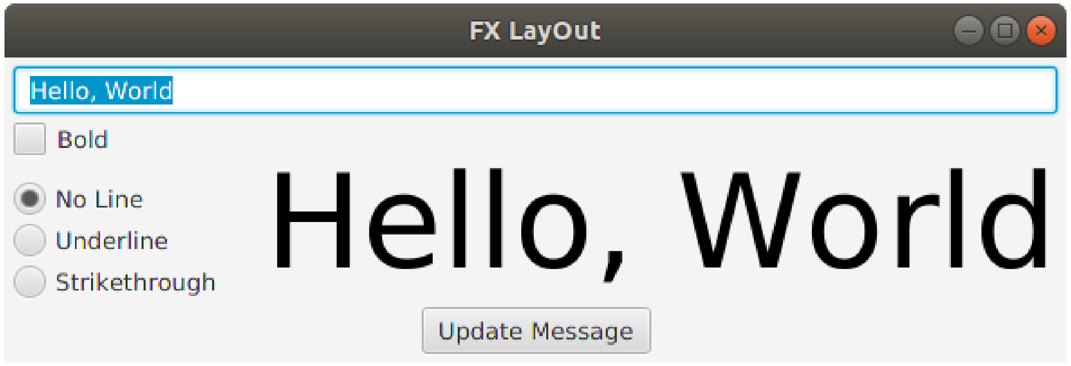


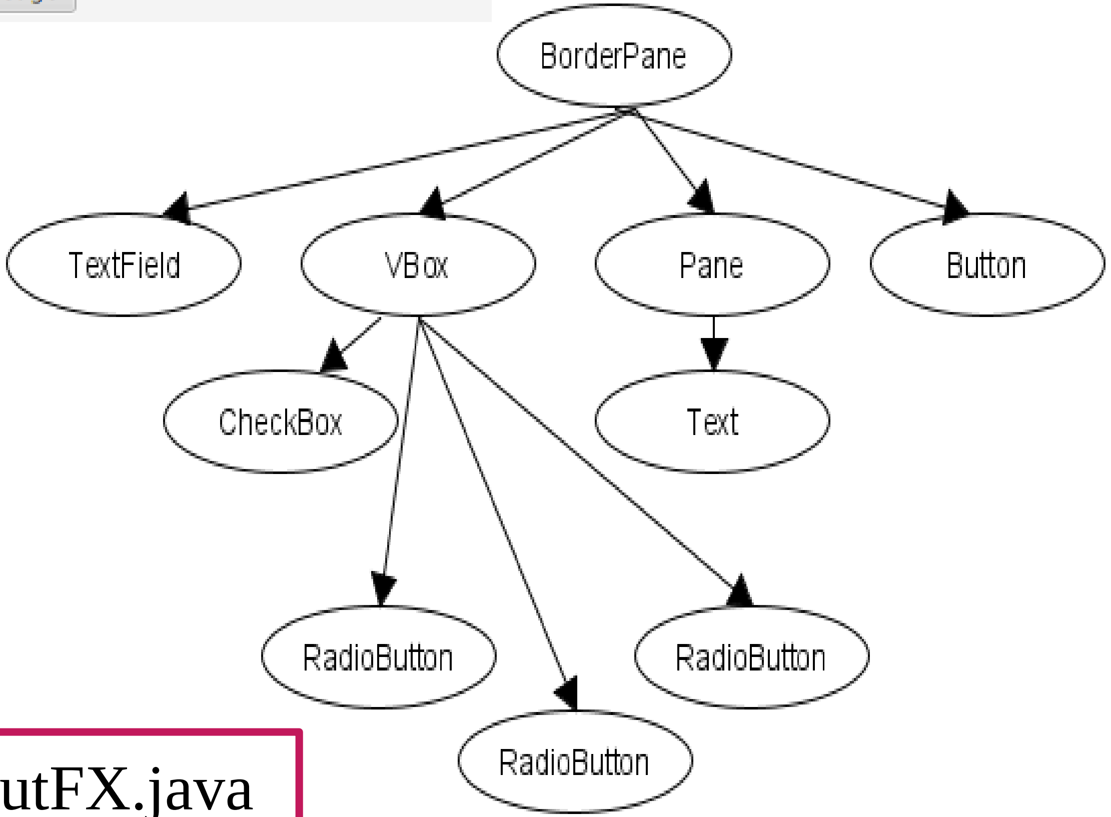
Message Example Scene Graph





Message Example Scene Graph





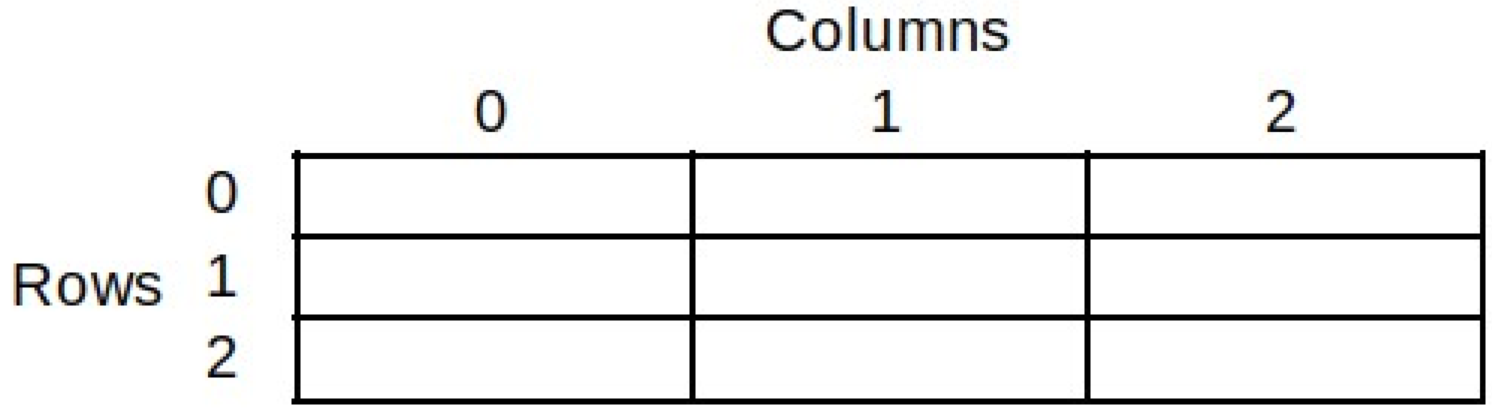
Refer to MessageLayoutFX.java

GridPane

GridPane is like a table.

You can span columns and rows.

Here is a 3x3 GridPane



GridPane

All cells in the same row have the same height

All cells in the same column have the same width

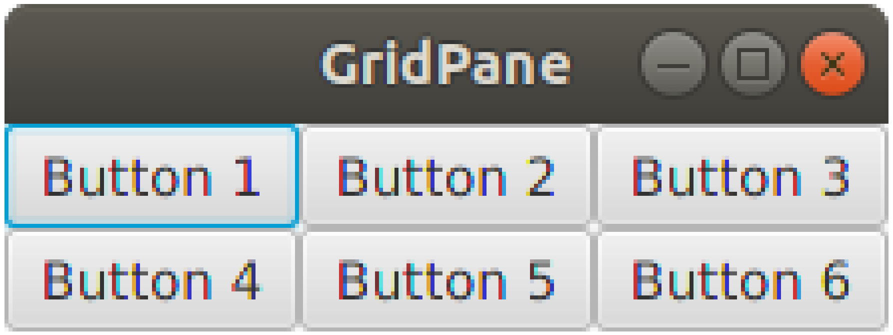
Different rows can have different heights
Different columns can have different widths

GridPane

add(Node child, int columnIndex, int rowIndex)

Adds a child to the gridpane at the specified column,row position.

```
GridPane gridPane = new GridPane();
gridPane.add(button1, 0, 0);
gridPane.add(button2, 1, 0);
gridPane.add(button3, 2, 0);
gridPane.add(button4, 0, 1);
gridPane.add(button5, 1, 1);
gridPane.add(button6, 2, 1);
```



Refer to GridPaneFX.java

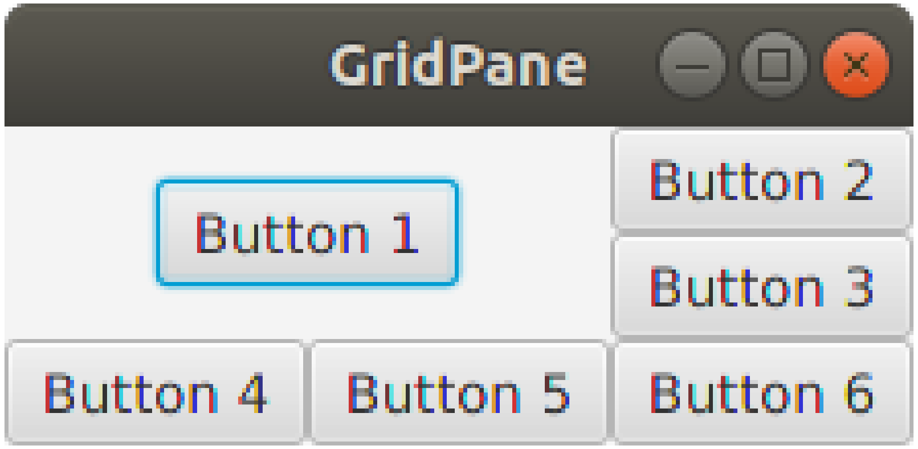
Documentation

GridPane(span)

add(Node child, int columnIndex, int rowIndex, int colspan, int rowspan)

Adds a child to the gridpane at the specified column,row position and spans.

```
gridPane.add(button1, 0, 0,2,2); // span two columns and rows
GridPane.setHalignment(button1, Pos.CENTER); // center aligned
gridPane.add(button2, 2, 0);
gridPane.add(button3, 2, 1);
gridPane.add(button4, 0, 2);
gridPane.add(button5, 1, 2);
gridPane.add(button6, 2, 2);
```



Refer to GridPaneSpanFX.java

For more examples refer to:

http://tutorials.jenkov.com/javafx/gridpane.html

Group

Group is a group of Shape s that's drawn in the order they are specified in the Group constructor.

```
Group boat = new Group(mast, sail, hull);
```

By grouping shapes together, it allows us to move them around!

```
boat.setTranslateX(100);
boat.setTranslateY(50);
```

Group

It also allows us to apply effects to it. Let's make the ship disappear

```
boat.setOpacity(0.2);
```

Refer to GroupFX.java