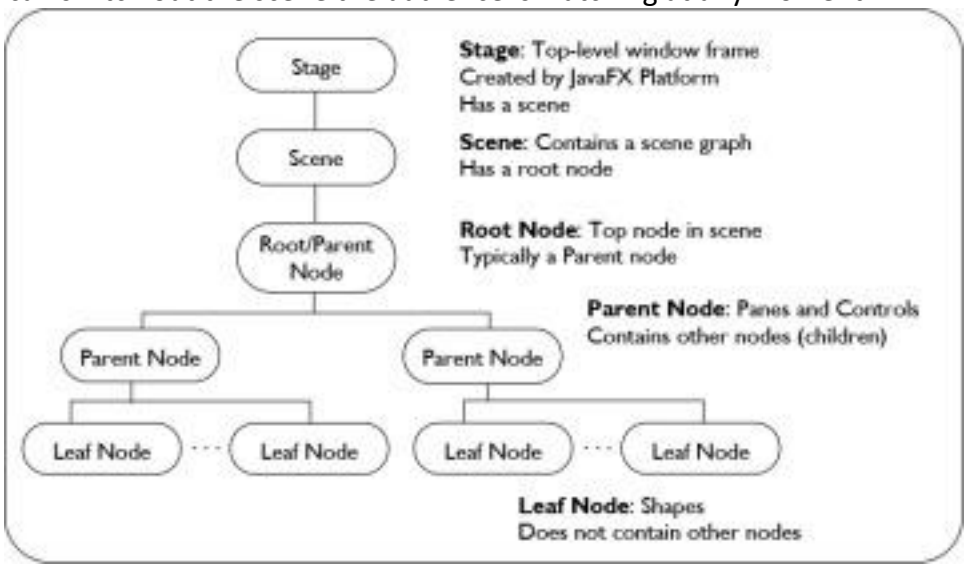
The Basics of JavaFX

Every JavaFX program extends **Application** – the main entry point for all JavaFX applications. When any JavaFX application is loaded, the init method within the Application class calls the start method.

The start method is the core of any JavaFX application. Not only is it called when the application is loaded, but it also contains a reference to the application's primary**Stage** as a parameter.

```
@Override public void start(Stage primaryStage)
{
    Group root = new Group();
    Scene scene = new Scene(root, 300, 250);
    primaryStage.setScene(scene);
    primaryStage.setTitle("The Click Me App");
    primaryStage.show();
}
```

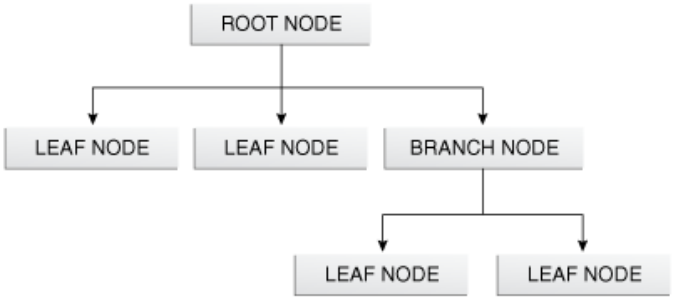
Now, you may be wondering, what is a stage? Good question! The stage is like the ground of your program. It can hold only one scene at a time, but can switch the its scene at any moment --- similarly to a stage in a play. Only one scene can be shown to the audience at once, yet the stage crew can switch out the scene the audience is watching at any moment.



When the start method is called by the init method in Application, the primaryStage is automatically passed into the parameters of the start method. It is up to the programmer to set the scene within this primaryStage.

A **scene** can be thought of as a surface that holds all the visual elements, or nodes, in an application.
The amount of **nodes** in each scene varies, and is represented in the above photo from the "Root/Parent Node" and below. A scene in javaFX is actually a graph. Don't worry about the

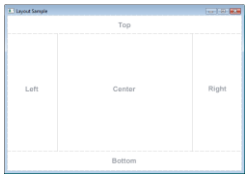
term graph now if you've never heard of it before, but a graph is essentially a set of nodes, as seen below.

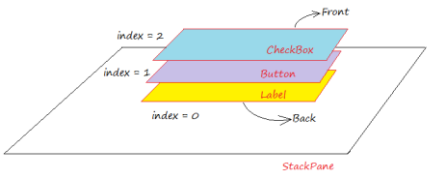


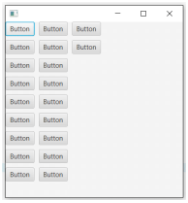
These nodes can be buttons, shapes, colors, images, or any other User Interface element.

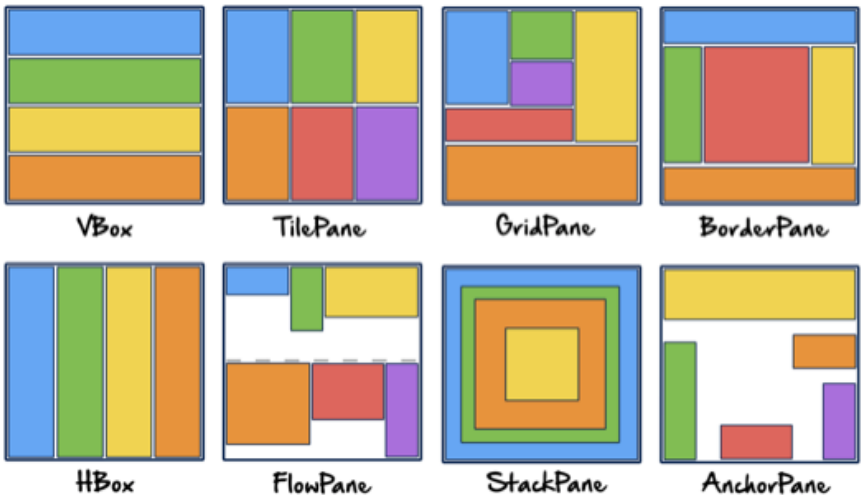
After creating all the required nodes in a scene, we generally need to arrange them in a specific order. Maybe you need an image to lie on top of a button (StackPane). Or perhaps you need a photo on the far right and a "Go!" button on the bottom of the screen (BorderPane).
Layout Panes, or containers, allow us to arrange our nodes. Some common layout panes include **HBox**, **VBox**, **BorderPane**, **StackPane**, and **GridPane**. Each of these layout panes are represented by a class whose purpose is detailed below:

S.No	Shape & Description
1	**HBox** The HBox layout arranges all the nodes in our application in a single horizontal row. The class named **HBox** of the package **javafx.scene.layout** represents the text horizontal box layout.
2	**VBox** The VBox layout arranges all the nodes in our application in a single vertical column. The class named **VBox** of the package **javafx.scene.layout** represents the text Vertical box layout.
3	**BorderPane** The Border Pane layout arranges the nodes in our application in top, left, right, bottom and center positions. The class named **BorderPane** of the package **javafx.scene.layout** represents the border pane layout.
4	**StackPane** The stack pane layout arranges the nodes in our application on top of another just like in a stack. The node added first is placed at the bottom of the stack and the next node is placed on top of it. The class named **StackPane** of the package **javafx.scene.layout** represents the stack pane layout.



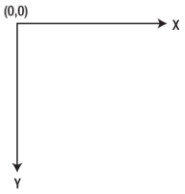


8	**GridPane** The Grid Pane layout arranges the nodes in our application as a grid of rows and columns. This layout comes handy while creating forms using JavaFX. The class named **GridPane** of the package **javafx.scene.layout** represents the GridPane layout.	
9	**FlowPane** The flow pane layout wraps all the nodes in a flow. A horizontal flow pane wraps the elements of the pane at its height, while a vertical flow pane wraps the elements at its width. The class named **FlowPane** of the package **javafx.scene.layout** represents the Flow Pane layout.	



(Don't worry about TilePane and AnchorPane in the picture above^.)

When utilizing a GridPane, it is also important to note that the coordinate system in JavaFX begins in the top left corner of the display. +x is located to the right, and +y points downward with gravity.



EventHandlers in JavaFX are used to detect when a user interacts with nodes in an Application. When is created by a user, it flows through the graph from the stage to the source node. This is called the event dispatch chain, but don't worry about remembering that name for now. As the event travels through the graph, any node with a filter for the event will execute. Once the event reaches the target node, the event will travel back to the top of the graph. Along the way, any of the nodes in the dispatch chain with a handler for the event will execute. Event **handler**s and filters contain application logic to process an event. A event can be recognized by more than one handler/filter.

You all created handler by implanting the functional interface EventHandler:

```
playButton.setOnMouseClicked((new EventHandler<MouseEvent>() {
   public void handle(MouseEvent event) {
      System.out.println("Hello World");
      pathTransition.play();
   }
}));
```