UNIT-5 JavaFx Event Handling, Controls and Components... (Notes)

- JavaFX is a library for building rich client applications with java. It provides an API for designing GUI applications that run on almost every device with Java Support.

- JavaFx is a set of graphics and media packages that enables developers to design, create, test, debug and deploy rich client applications that operate consistently across diverse platforms.

- JavaFx stands for Java "Special EFFECTS" as Fx is normally the abbreviation given to special effects mostly Sound or visual.

JavaFx API :- In Java 8, 9, 10 no additional setup is necessary to start working with the javaFx library.

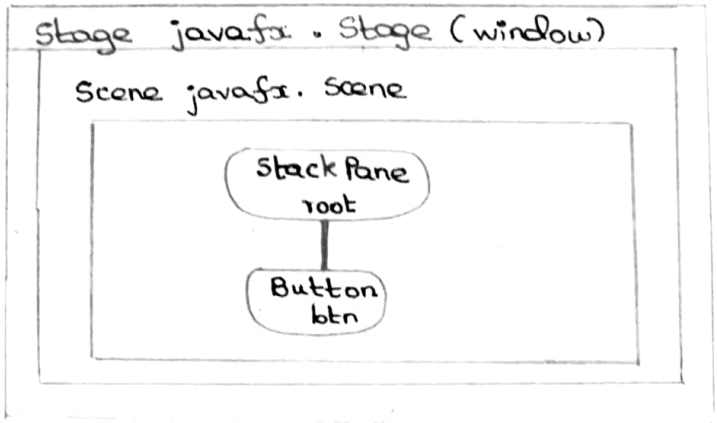
JavaFx Architecture:

JavaFx uses hardware accelerated graphics pipeline for the rendering, known as Prism. What's more, to fully accelerate the graphics usage, it leverages either software or hardware rendering mechanism, by internally using DirectX and open GL.

JavaFx has a platform dependent Glass windowing toolkit layer to connect to the native operating System. It uses the operating System's event queue to schedule thread usage. Also, it asynchronously handles windows, events, timers.

The Media and web engines enable media playback and HTML/CSS Support.

Let's see what the Main Structure of a JavaFx application looks like:

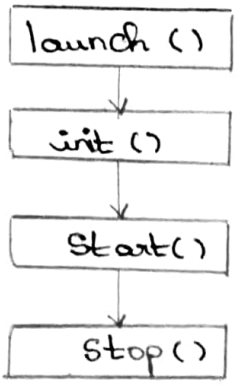


Here, we notice two main Containers:

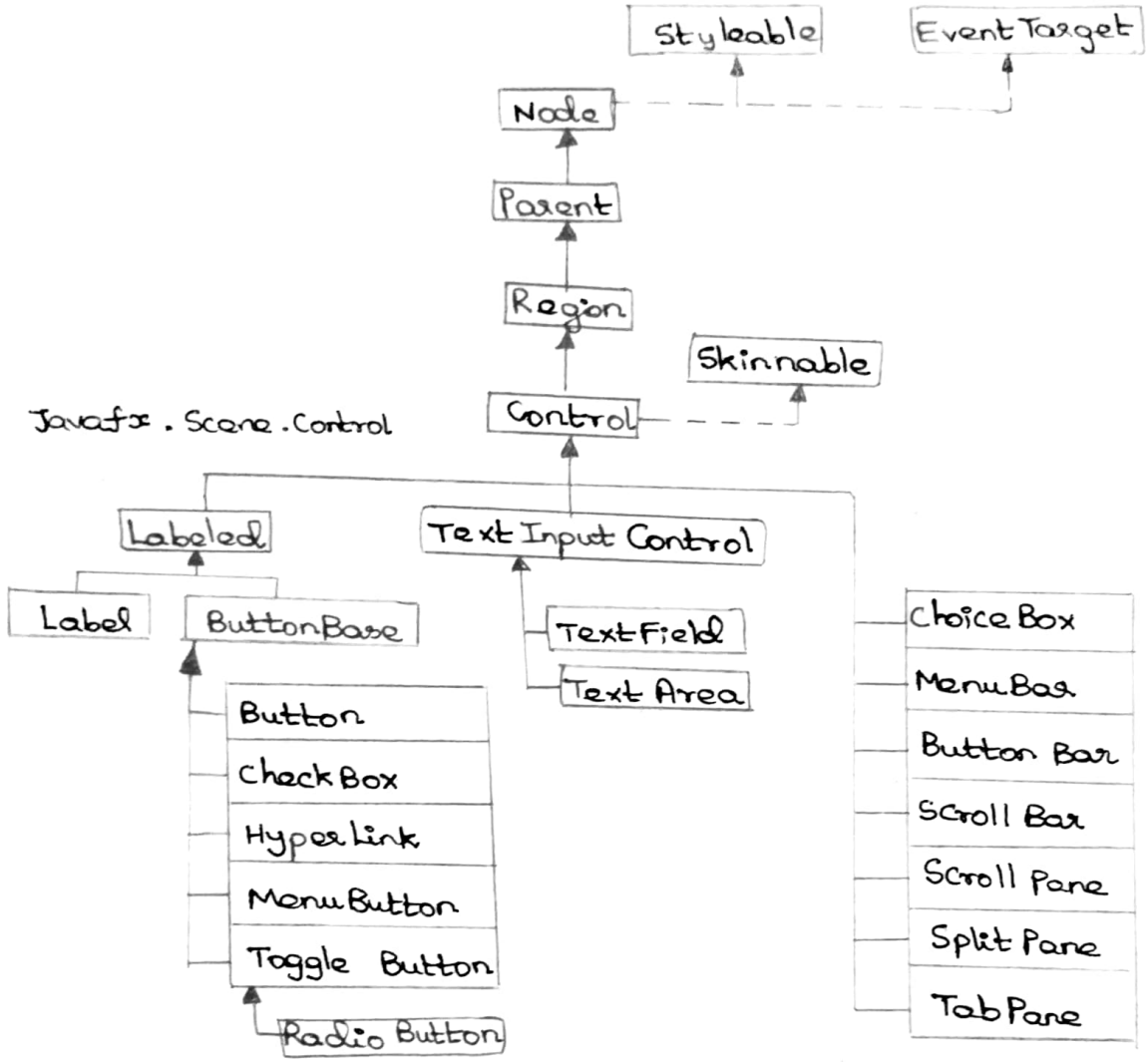
- Stage is the main container and the entry point of the application. It represents the main window and passed as an argument of the Start() method.
- Scene is a Container for holding the UI elements, Such as Image views, Buttons, Grids, TextBoxes.

The Scene can be replaced or switched to another Scene. This represents a graph of hierarchical objects, which is known as a Scene Graph. Each element in that hierarchy is called a node. A single node has its ID, Style, effects, event handlers, state.

<u>JavaFx Life Cycle</u>:



JAVAFX Controls Hierarchy :-



The javafx . application . Application class has the following life cycle methods :

- init() - is called after the application instance is created. At this point, the JavaFX API isn't ready yet, so we can't create graphical components here.

- Start (Stage stage) - all the graphical components are created here . Also, the main thread for the graphical activities starts here.

- Stop () - is called before the application shutdown ; for example, when a user closes the main window . It's useful to override this method for some cleanup before the application termination.

The static launch() method starts the javafx application

Simple HelloWorld Application using JAVAFx

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.Scene.Scene;
import javafx.Scene.Control.Button;
import javafx.Scene.Layout.StackPane;
import javafx.Stage.Stage;

public class HelloWorld extends Application {
 public static void main (String args[])
{ launch(args); }

 @override
  public void start (Stage primaryStage) {
   primaryStage.SetTitle("HelloWorld!");
   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();
 }
}
```

Here important things to know about the basic Structure of a JavaFx application:

- The main class for a JavaFx application extends the javafx.application.Application class. The Start() method is the main entry point for all JavaFx applications.

- A JavaFx application defines the user interface container by means of a Stage and a Scene. The JavaFx Stage class is the top-level JavaFx container. The JavaFx Scene class is the Container for all content.

- In JavaFx, the content of the Scene is represented as a hierarchical scene graph of nodes. In this example, the root node is a StackPane object, which is a resizable layout node. This means that the root node's size tracks the scene's size and changes when the stage is resized by a user.

- The root node contains one child node, a button control with text, plus an event handler to print a message when the button is pressed.

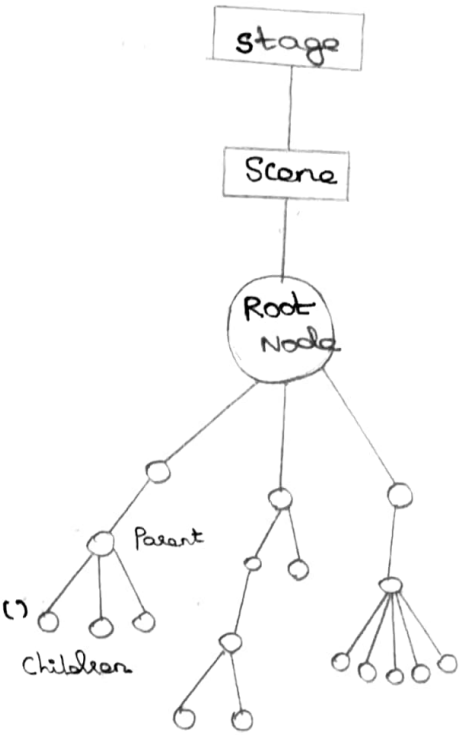
Controls in JavaFx:

The Scene is arranged as a tree-like graph containing Node objects, which is called the scene graph. The scene graph begins with a root node. The nodes of a graph are basically the components of the window as well as the layout regions and the components within them. If node c is contained within another node p, then c is called a child of p and p is the parent of c.

All window components actually keep pointers to their parent. Parents of nested components are stored recursively.

```
control   aComponent;
Parent    P;
Parent    parentOfParent;

aComponent = ...;
P = aComponent.getParent();
ParentOfParent = P.getParent();
```

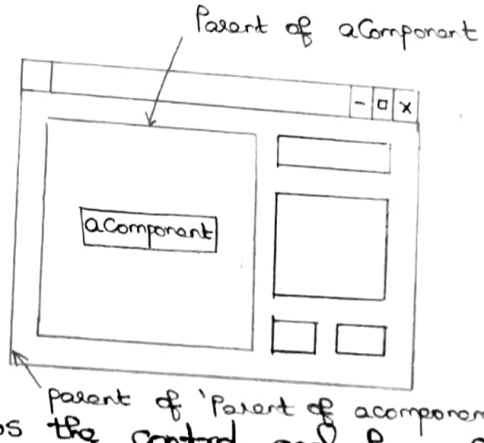


Parents keep pointers to their children and we can access these using the parent's getChildren() method.

```
Parent aParent = ...;

ObservableList <Node> children
= aParent.getChildren();

Node C1 = children.get(0);
Node C2 = children.get(1);
Node C3 = children.get(2);
```



The class hierarchy shown ealier shows the control and Pane classes. Each of these classes has many subclasses representing various window components and layout managers.

JavaFx Controls:

Label class

javafx.Scene.Control.Labeled

javafx.scene.control.Label

+ Label() → creates an empty label.

+ Label (text : String) → Creates a label with specified text.

+ Label (text : String, graphic : Node) → Creates a label with the specified text and graphic.

Button:

- A button is a control that triggers an action event when clicked.
- JavaFx provides regular buttons, toggle buttons, check box buttons, and radio buttons.
- The common features of these buttons are defined in ButtonBase and Labeled classes.

javafx.Scene.Control.Labeled

↑

javafx.Scene.Control.ButtonBase

The getter and setter methods for property values & a getter for property itself are provided in classes but omitted in UML diagram for brevity.

-onAction : object property <EventHandler<ActionEvent>>

Defines a handler for handling a button's action.

↑

javafx.scene.control.Button

+ Button() → Creates an empty Button.

+ Button (text : String) → Creates a button with the specified text.

+ Button (text : String, graphic : Node) → Creates a button with specified text and graphic.

CheckBox:

- A checkBox is used for the user to make a selection (square box). checkBox inherits all the properties from ButtonBase and Labeled: onAction, text, graphic, alignment, graphicTextGap, textFill, contentDisplay.

javafx . Scene . Control . labeled

↑

javafx . Scene . Control . ButtonBase

-onAction : objectProperty
<EventHandler><ActionEvent>

↑

javafx . Scene . Control . checkBox

- Selected : BooleanProperty → Indicates whether this checkbox is checked.

+ checkBox() → creates an empty checkBox.

+ checkBox (text : String) → Creates a check box with specified text.

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in UML diagram for brevity.

Defines a handler for handling a button's action.

Radio Button

- Radio button allow to choose a single item from a group of choices.
- Radio buttons display a circle that is either filled (if selected) or blank (if not selected).

javafx . Scene . Control . ToggleButton

- Selected : BooleanProperty
- toggle Group :
objectProperty <ToggleGroup>

Indicates whether the button is selected. Specifies the button group to which the button belongs.

+ ToggleButton () → Creates an empty toggle button.

+ ToggleButton (text : String) → Creates toggle button with specified text.

+ ToggleButton (text : String , graphic : Node) → Creates toggle button with specified text and graphic.

↑

javafx . Scene . Control . RadioButton

+ RadioButton () → creates an empty radio button.

+ RadioButton (text : String) → Creates a radio button with the specified text.

TextField :

- A text field can be used to enter or display a string.

TextField is a subclass of TextInputControl.

javafx.scene.control.TextInputControl

-text : StringProperty. → The text content of this control.

-editable : BooleanProperty → Indicates whether the text can be edited by the user.

javafx.scene.control.TextField

-alignment : objectProperty <Pos> → specifies how the text should be aligned in the text field.

-PrefColumnCount : IntegerProperty → specifies the preferred no. of columns in the text field.

-onAction :

ObjectProperty < EventHandler <ActionEvent>> → specifies the handler for processing the action event on textfield.

+ TextField() → Creates an empty textfield.

+ TextField (text : String) → Creates a textfield with the specified text.

TextArea

- A TextArea enables the user to enter multiple lines of text.

javafx.scence.control.TextInputControl

-text : String property. → The text content of this control.

-editable : Boolean property. → Indicates whether the text can be edited by the user.

> The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.Scene.Control.TextArea

- PrefColumnCount : Integer Property. → specifies the preferred no. of text columns.
- PrefRowCount : Integer property. → specifies the preferred no. of text rows.
- wrapText : BooleanProperty. → specifies whether the text is wrapped to the next line.

+ TextArea() → creates an empty text area.

+ TextArea(text : String) → Creates a text area with the specified text.

ComboBox :-

- A combo box, also known as a choice list or drop-downlist, contains a list of items from which the user can choose.

Javafx.Scene.Control.ComboBoxBase<T>

- value : objectProperty<T> → The value selected in the combo box.
- editable : BooleanProperty → Specifies whether the combo box allows user input.
- onAction : objectProperty<EventHandler<ActionEvent>> → Specifies the handler for processing the action event.

javafx.Scene.Control.ComboBox<T>

- items : ObjectProperty<observablelist<T>> → The items in the combo box popup.
- visibleRowCount : IntegerProperty → The maximum number of visible rows of the items in the combo box pop up.

+ ComboBox() → Creates an empty Combo box.

+ ComboBox(items : observablelist<T>) → Creates a Combo box with the specified items.

List View :-

- A list view is a Component that performs basically the Same function as a Combo box, but it enables the user to choose a single value or multiple values.

javafx.Scene.Control.ListView<T>

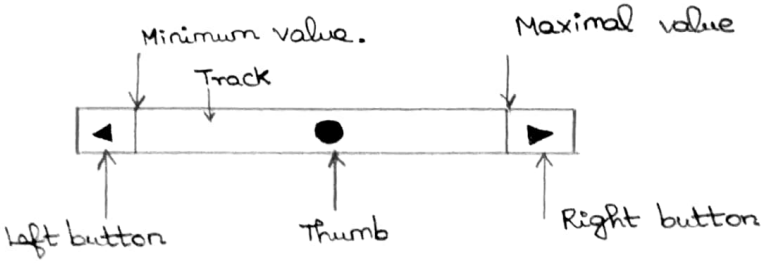
- items : objectProperty <observablelist <T>> → The items in the list view.
- orientation : BooleanProperty. → Indicates whether the items are displayed horizontally or vertically in the list view.
- SelectionModel :

objectProperty <MultipleSelectionModel <T>> → specifies how items are selected. The SelectionModel is also used to obtain the selected items.

+ ListView() → Creates an empty list view.
+ ListView (items : observableList <T>) → Creates a list view with the specified items.

ScrollBar :

- A Scroll bar is a Control that enables the user to select from a range of values. The Scrollbar appears in two Styles : horizontal and vertical.



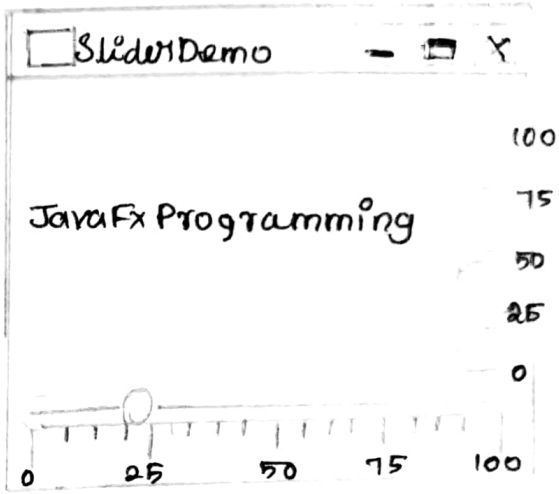
javafx.scene.control.ScrollBar

- blockIncrement : DoubleProperty → The amount to adjust the scroll bar if the track of the bar is clicked (default : 10)
- max : DoubleProperty → The Maximum value represented by this scroll bar (default : 100)
- min : DoubleProperty → The Maximum value represented by this scroll bar (default : 0).
- unitIncrement : DoubleProperty.
  ↳ The amount to adjust the scroll bar when the increment() and decrement() methods are called (default : 1).
- value : DoubleProperty → current value of the scroll bar (default : 0)
- visibleAmount : DoubleProperty → The width of the scroll bar (default : 15)
- orientation : ObjectProperty <orientation>
  ↳ Specifies the orientation of the scroll bar (default : HORIZONTAL).

+ ScrollBar() → Creates a default horizontal scroll bar.
+ increment() → Increments the value of the scroll bar by unit Increment.
+ decrement() → Decrements the value of the scroll bar by unit Increment.

Slider:

Slider is similar to scrollBar, but slider has more properties and can appear in many forms.



The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted int the UML diagram for brevity.

javafx.scene.control.Slider
- blockIncrement : Double property - max : Double property - min : Double Property. - value : Double Property. - orientation : ObjectProperty<orientation> - majorTickUnit : Doubleproperty - minorTickUnit : IntegerProperty. - showTickLabels : BooleanProperty. - showTickMarks : BooleanProperty
+ slider() + slider(min : double, max : double, value : double)

The amount to adjust the slider if the track of the bar is clicked (default : 10).

The maximum value represented by this slider (default : 100)

The minimum value represented by this slider (default : 0).

Current value of the slider (default : 0).

Specifies the orientation of the slider (default : HORIZONTAL)

The unit distance between major tick marks.

The number of minor ticks to place between two major ticks.

Specifies whether the labels for tick marks are shown.

Specifies whether the tick marks are shown	
Creates a default horizontal slider.	
Creates a slider with the specified min, max and value	

Using Built-in Layout Panes

JavaFX provides many types of panes for organizing nodes in a container.

Class	Description
Pane	Base class for layout panes. It contains the getChildren() method for returning a list of nodes in the pane.
StackPane	Places the nodes on top of each other in the center of the pane.
FlowPane	Places the nodes row-by-row horizontally or column-by-column vertically.
GridPane	Places the nodes in the cells in a two-dimensional grid.
BorderPane	Places the nodes in the top, right, bottom, left and center regions.
HBox	Places the nodes in a single row
VBox	Places the nodes in a single column.

Flow Pane

The nodes within a Flowpane layout pane are laid out consecutively and wrap at the boundary set for the Pane.

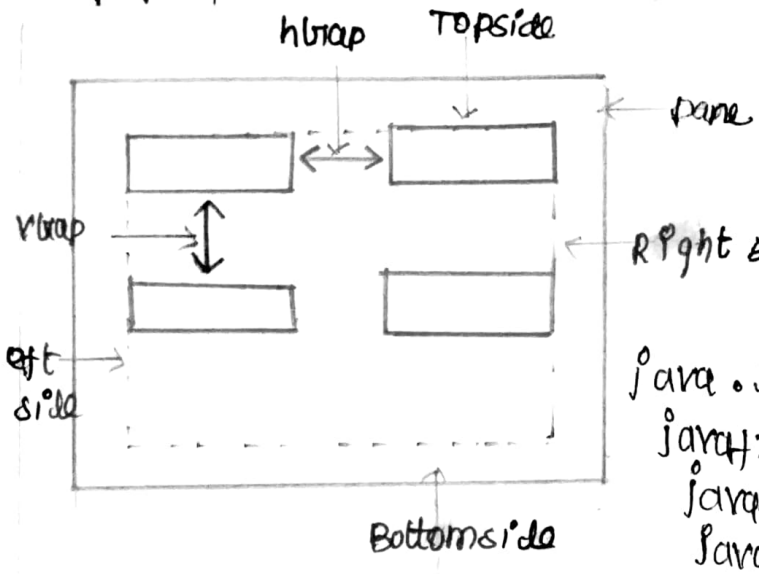
Nodes can flow vertically (in columns) or horizontally (in rows).

Figure shows a sample horizontal flow pane using numbered icons.

By contrast, in a vertical flow pane, column one would contain pages one through four and column

two would contain pages five through eight

Gap properties and padding property can be set.

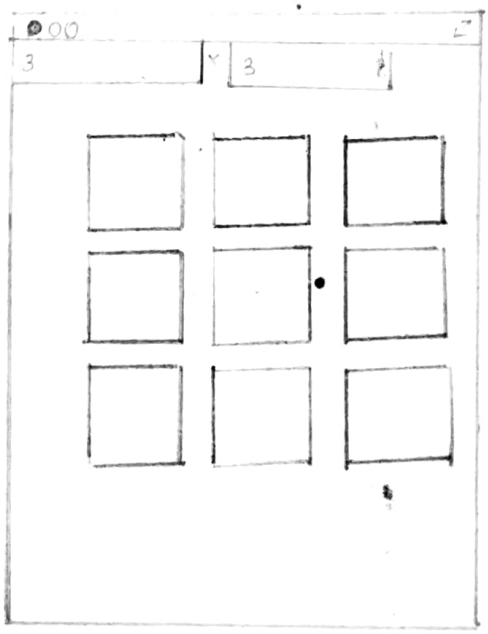


class FlowPane

java.lang.Object
javafx.scene.Node
javafx.scene.Parent
javafx.scene.layout.Region
javafx.scene.layout.Pane
javafx.scene.layout.FlowPane

GridPane

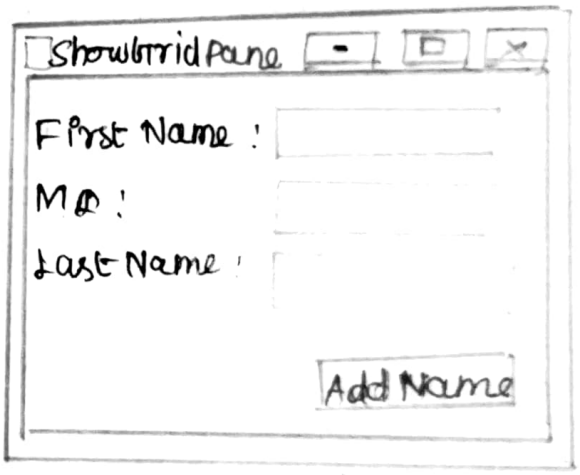
* The GridPane layout pane enables you to create a flexible grid of rows and columns in which to lay out nodes.
* Nodes can be placed in any cell in the grid and can span cells as needed.
* A grid pane is useful for creating forms or and layout that is organized in rows and columns.
* The total number of rows/columns does not need to be specified up front as the grid pane will automatically expand/contract the grid to accommodate the content.
* Scene size is automatically computed according to the sizes of the nodes placed inside the scene.



Example : Using Grid Pane.

```
public void start(Stage primaryStage){
// Create a pane and set its properties
GridPane pane = new GridPane();
pane.setPadding(new Insets(11.5,12.5,13.5,14.5));
pane.setHgap(5.5);
pane.setVgap(5.5);
// Place nodes in the pane
pane.add(new Label("First Name:"),(0,0));
pane.add(new TextField(), 1,0);
pane.add(new Label("MI:"), 0,1);
pane.add(new TextField(), 1, 1);
pane.add(new Label("Last Name:"),0,2);
pane.add(new TextField(),1,2);
Button btAdd = new Button("Add Name");
Pane.add(btAdd, 1, 3);
GridPane.setHalignment(btAdd, HPos.RIGHT);
// Create a scene and place it in the stage
Scene scene = new Scene(pane);
PrimaryStage.setTitle("ShowGridPane");// set the stage title
primaryStage.setScene(scene);// place the scene in the stage
primaryStage.show();// Display the stage
}
```

(0,0)	(1,0)
(0,1)	(1,1)
(0,2)	(1,2)
(0,3)	(1,3)



BorderPane

The getter and setter methods for property values and a getter for property itself are provided in the class but omitted in the UML diagram for brevity

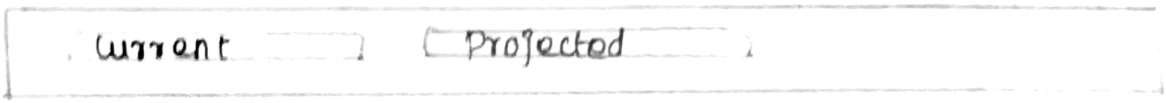
javafx.scene.layout.BorderPane	
-top : ObjectProperty<Node>	The node placed in top region (default : null).
-right : ObjectProperty<Node>	The node placed in the right region (default : null).
-bottom : ObjectProperty<Node>	The node placed in the bottom region (default : null).
-left : ObjectProperty<Node>	The node placed in the center region (default : null).
-center : ObjectProperty<Node>	
+ BorderPane()	Creates a BorderPane.
+ setAlignment(child : Node, pos : Pos)	Set the alignment of the node in the BorderPane.

javafx.scene.layout.BorderPane	
void	setCenter(Node value). Sets the value of the property center.
void	setLeft(Node value) Sets the value of the property left.
static void	setMargin(Node child, Insets value) Sets the Margin for the child when contained by border pane.
void	setRight(Node value) Sets the value of the property right.
void	setTop(Node value) Sets the value of the property top.

HBOX - VBOX

HBOX :

The HBox layout pane provides an easy way for arranging a series of nodes in a single row



VBOX :

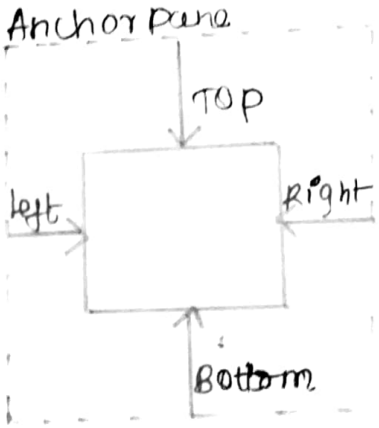
The VBox layout pane is similar to the HBox layout pane, except that the nodes are arranged in a single column.

Data
Sales
Marketing
Distribution
Costs

The padding property can be set to manage the distance between the nodes and the edges of the VBox pane

AnchorPane

* Anchor pane allows the edges of child nodes to be anchored to an off set from the anchor pane's edges
* As the window is resized, the nodes maintain their position relative to their anchor point.
* Nodes can be anchored to more than one position and more than one node can be anchored to the same position

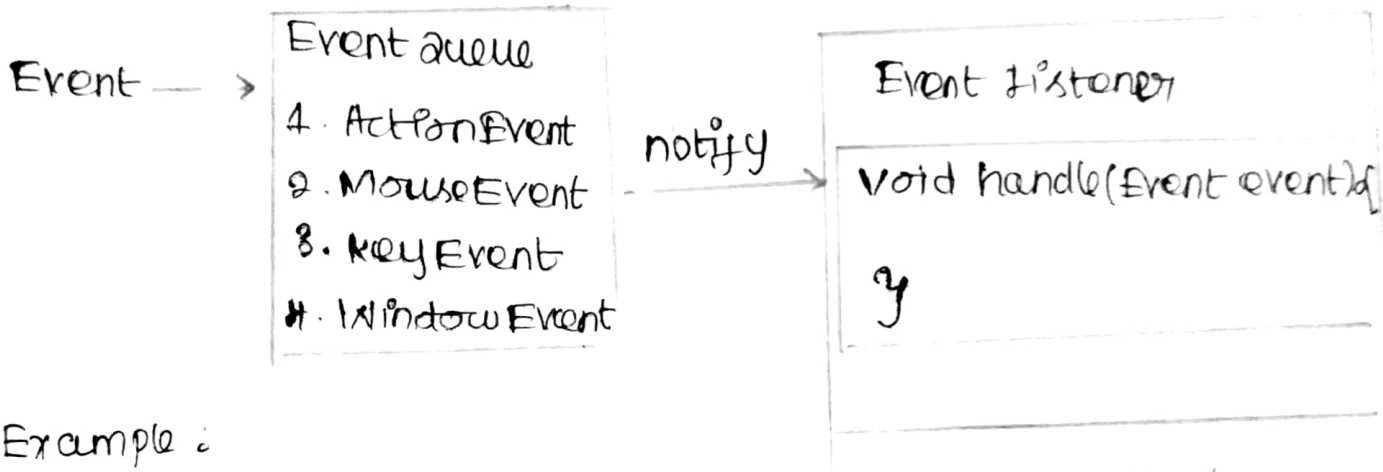


* static void setBottomAnchor(Node child, Double value)
Sets the bottom anchor for child when contained by an anchor pane.
* static void setLeftAnchor(Node child, Double value)
Sets the left anchor for the child when contained by an anchor pane.
* static void setRightAnchor(Node child, Double value)
Sets the right anchor for child when contained by an anchor pane.

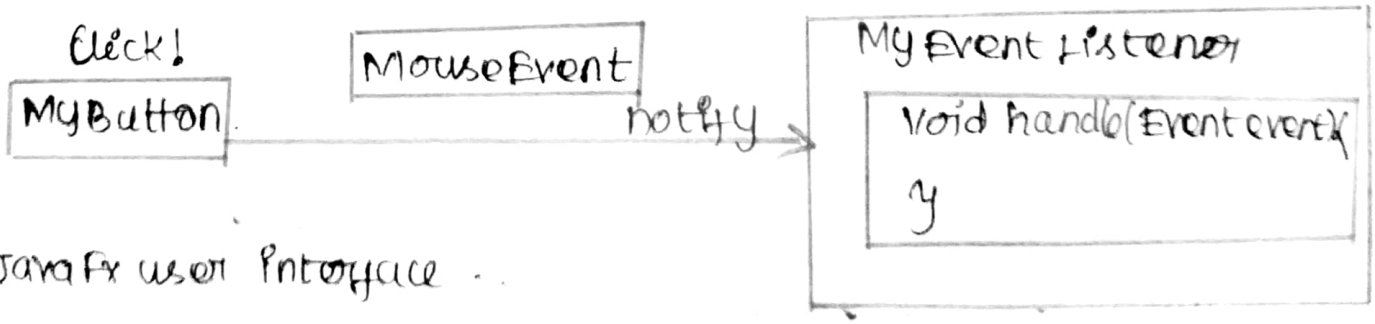
static void setTopAnchor (Node child, Double value)
Sets the top anchor for the child when contained by anchor pane.

Event Handling in JAVAFX

Graphics applications use events.
An event dispatcher receives events and notifies interested objects.



Example:
User clicks mouse on a button - that's an Event
JavaFx creates a MouseEvent object.
the MouseEvent describes what happend (which mouse button was presses, which field it was in).
JavaFx looks for a registered "Event listener", and calls it using the MouseEvent as parameter.



JavaFx user Interface.

All Events are subclasses of Event

Event

- ActionEvent
- Input Event
  - KeyEvent
  - MouseEvent
- Web Event
- Window Event

Source of Events

A component or node can be "source" of many kinds of events.

Some event types are different for each node or component.

Its not complicated! Mostly you can guess event types.

Button	ActionEvent (button press)
TextField	Action Event Key Event (Key Press, Key Release, KeyTyped)
Any kind of Node	Mouse Event: MousePress, MouseReleased, Mouse Clicked, MouseDragged, etc. Rotation events, Touch events

Menus

Menus make selection easier and are widely used in window applications.

JavaFx provides five classes that implement menus; MenuBar Menu, MenuItem, CheckMenuItem, and RadioButtonMenuItem

MenuBar is a top-level menu component used to hold the menus.

A menu consists of menu items that the user can select (or toggle on or off).

A menu item can be an instance of MenuItem, checkMenuItem, or RadioButtonMenuItem

Menu items can be associated with mouse and keyboard.

To show a message, construct an Alert object by supplying the alert type and text. Then call the showAndWait method.

```
Alert alert = new Alert (Alert.AlertType.INFORMATION,
" Everything is fine.");          alert.showAndWait();
```

The dialog is displayed until the user dismisses it (by clicking the OK button, closing the window, or hitting the Escape key). By changing the type to the CONFIRMATION, WARNING, or ERROR type, the header text and image change-

See figure 1.

Figure 1 Alert dialogs

Slider

Slider is simi

The CONFIRMATION alert type shows two buttons. To see whether the user confirms or cancels, you need to capture the return value of the showAndWait method. It has type Optional<ButtonType>. The ButtonType enumeration lists the button types that are allowed in an alert dialog. The optional class, which we will discuss in chapter 1 of Volume II, denotes a value that may be present or absent. A confirmation dialog always returns ButtonType.OK or ButtonType.CANCEL, wrapped in an optional. Since there is no possibility of an empty optional, you can call the get method to find out whether the user has clicked OK:

```
if (alert.showAndWait().get() == ButtonType.OK)
{
   // the user has confirmed.
   ...
}
```