Unit - I

Introducing javafx GUI programming

Introducing Javafx GUI programming, Javafx basic concepts, Javafx packages, the stage and scene class nodes and scene graphs, layouts. The application class and lifecycle methods, launching the javafx application, the javafx application skeleton. Compiling and running javafx programme, The application thread,

Part - II -> A Simple Javafx Controlls.

label, using buttons and Events, Event basics, Introducing the button Control, demonstrate the event handling in the button, The button drawing directly on to the Canvass.

Introducing javafx GUI programming:-

Javafx is a software platform for creation and delivering desktop applications and as well as rich internet application. This javafx application are run across the wide variety of devices such as desktops, mobile phones, Tv's and tablets etc.

Javafx is intended to replace the swings it provides powerful, streamline and flexible frame work to specify the modern, visually exiting GUI's (graphical user interface). It provides more visual and dynamic approach to the GUI. It has all features of swings for Example, it is a light weight and it supports mvc (model view Controllers). where model represents architecture data and view represents presentation. The Controllers is an interface between model and view.

Initially Javafx has implemented in two phases. In first phases it has only scripts that are referred as Javafx scripts but after releasing of Javafx 2.0. It was fully programmed with java.

features of Javafx:-

1) Javafx is fully implemented using java programming
2) The javafx is also support to the FXML just like HTML markup language.
3) The javafx also supports CSS (cascading style sheets) to design the stylish user interface.
4) The javafx has support to the scene builder application in IDE's we can attach this application to provide the dragon drop scenario.
5) It has the built in controls such as button, textfield, textlabel etc.
6) It has the built in integrated graphics libraries. This libraries are used to create 2D or 3D objects in javafx application.
*7) It also has the built in API's to provide more support to develop javafx application.
8) It also has the graphics pipeline that means it will support the hardware accelerated animation games, by adding graphics card.

Installation of JDK 1.8

If you want to develop javafx applications we must install JDK 1.8 or latest versions because in JDK 1.8 we have Javafx library in addition to that we can also install the IDE's (Integrated development environment) like Eclipse, netbeans etc.
* It was developed by chris Oliver. Initially it was referred as FFF (form follows functionality). At the starting stage it was acquired by Sun microsoft systems now it was acquired by Oracle Corporation.

Javafx basic Concepts:-

Javafx has several features like as a swing. Javafx application is developed in JDK 1.8 environment. It has several API's each and every API has several classes and interfaces and methods to develop javafx application. It has several packages to provide support to the newly created javafx application. It has more than 30 javafx packages

The important packages are as follows.

Javafx packages:-

1) javafx.application:- This package Contains various classes and interfaces to use the life cycle methods of application class.
2) javafx.stage:- This package is import to create a stage in javafx application.
3) javafx.scene:- This package has imported to create scene in a stage class. It also Contains the subpackages like layout, control, text, image view, Paint, chart etc.
4) javafx.animation:- This package is imported to create animational object in javafx application
5) javafx.geometry:- This package is imported to create graphical objects like 2D and 3D shapes.
6) javafx.event:- This package is imported to handle events in javafx applications.

We can import all this javafx packages by using import statement as follows.

for Example, import javafx.application.*;
import javafx.scene.layout.*;

Stage and scene class:-

stageClass: The stage is a top level Container of javafx application and also referred as a space in javafx application. The stage is automatically created by java run time system by passing stage object as a argument to the start method. The start method is a one of the life cycle method of Application Class. Every javafx application has atleast one stage. stage is referred as primary stage.

The stages having the following types:

a) decorated stage
2) undecorated stage
3) transparent stage
4) unified stage
5) utility stage

Based on the requirement we will add or create more stages to our javafx application.

```
Stage st = new Stage();
```

where Stage is a class and st is a stage object.

Stage class has the following methods.

1) setTitle():- This method is used to set the title to our stage on javafx application

Ex:- `st.setTitle("This is Demo");`

where st is a stage class object.

2) setScene():- It is used to set scene object to the stage object.

Ex:- `st.setScene(s);`

where st is the stage object and s is the scene object.

3) show():- It is used to display a created stage in a javafx application

Ex:- `st.show();`

A Loosely speaking that the stage is a space or Container which contains a scene.

Scene Class:- scene is a Container which Contains the items or Controls. loosely speaking that stage is a space and scene is a what goes inside the space. (or)

The Stage is a Container which Contains a scene A scene is a Container which Contains items or Controls every javafx application has atleast one scene in a stage we can create a scene using following statement.

```
Scene S = new Scene(root);
```

where Scene is a Class and s is a Scene object and root is a object of parent node In Scene Class Construction we have atleast one parameter that is a root node. The scene class Construction has following Syntax.

Scene (Parent root, double width, double height);

The scene is added to the Stage class by using a stage class method that is setscene method.

Ex:- st.setscene(s);

where s is a scene class object and st is a stage class object.

Nodes and scene graphs:-

Node:- The individual element of a scene is called a node. Nodes may be created as a group and a node may contain child nodes which is referred as parent node or branch node. All child nodes are referred as leaf nodes.

Scene graph:- A collection of all nodes in a scene to be referred as scene graph which is equals to the scene tree structure. In a scene graph we have parent nodes, child nodes and root node. The root node is a node which doesn't have any parent node.

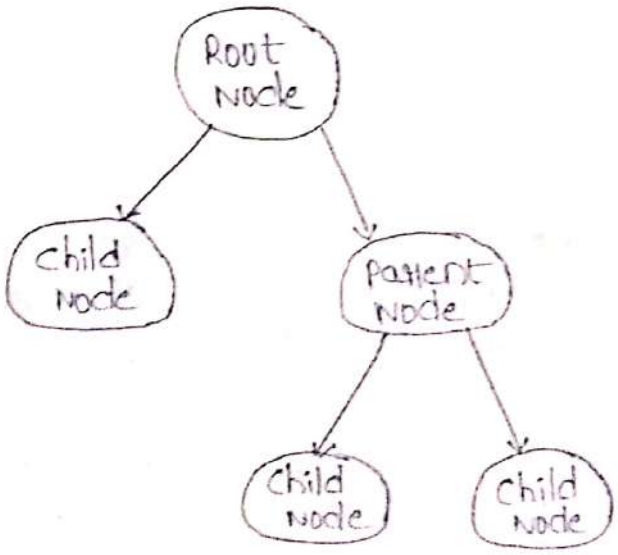
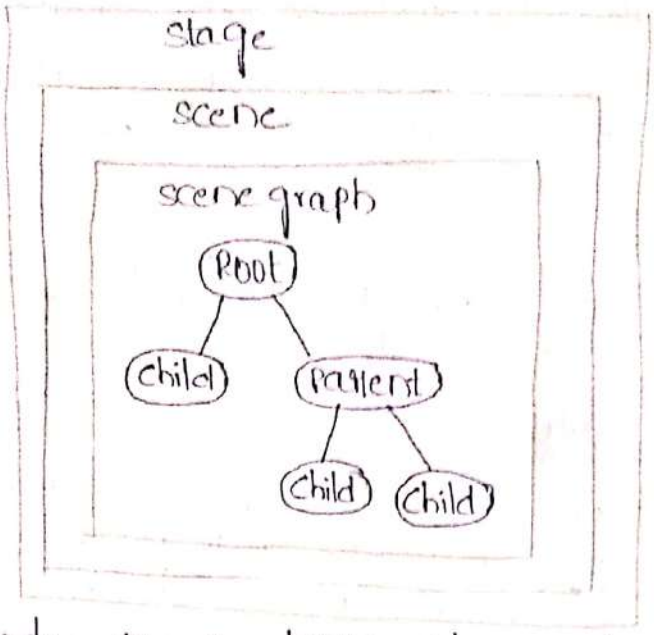


Fig:- Scene graph

The relationship between stage, scene and scene graph as follows



A Node is a base class of all Nodes it has the subclasses like Parent, Region, group etc.

Layouts:- The arrangement of nodes are a scene graph in a particular place of scene is referred as layout Container. The following are the layouts

1) HBox
2) V.Box
3) Text flow
4) Border pane
5) Stack pane
6) Flow pane
7) Anchor pane
8) Grid pane
9) Tile pane

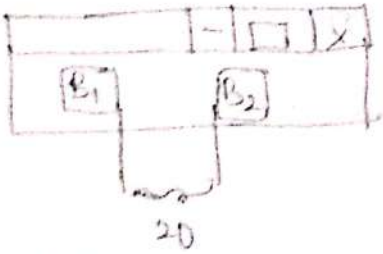
All these layouts are specified in javafx.scene.layout package. we have to import these package into our javafx application to use any one of the layout

HBox:- HBox layout is used to arrange the controls in a scene as a horizontal manner. where 'H' refers horizontal. This layout class is packaged javafx scene.layout.*. It has the following methods

1) setAlignment:- this Method is used to align the layout at a particular border. (Top, Centered, left, right, bottom) Center_left (horizontally it is in center and vertically it is in left). Center_right

2) setspacing:- using this method we can set the space

between Controls



setspace(20);

we can create a HBox layout object as follows

```
HBOX Hb = new HBOX();
Button b1 = new Button("B1");
Button b2 = new Button("B2");
Scene S = new Scene(hb, 300, 200);
hb.getchildren().add(b1);
hb.getchildren().add(b2);
st.setscene(s);
st.show();
```

VBox:- VBox is layouts its childrens in vertical manner it is also packaged in javafx.scene.layou

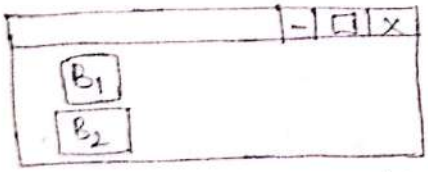
vBox object is created as follows

```
VBOX vb = new VBOX
```

and we have the following two methods.

1) set alignment and 2) set spacing

Ex:-




```
VBOX Vb = new VBOX();
Button b1 = new Button("B1");
button b2 = new Button("B2");
SCene S = new SCene(Vb, 300, 200);
vb.getchildren().add(b1);
vb.getchildren().add(b2);
St.setscene(s);
St.show();
```

flowpane flowpane layout is used to layouts its childrens either in horizontal or vertical based on its orientation property it flows the controls in horizontal or vertical it simply wrap the controls when there is not enough space at boundary of

yours scene.

If there is no space then it immediately layout its children in a next row or next column.

we can create a layout as follows

```
flowpane fp = new flowpane();
```

It has the following methods:

1) setAlignment:- It will align the layout in a particular position

2) setHgap():- It will provide the horizontal space between controls.

3) setVgap():- It will provide vertical space between controls

4) setOrientation():- It is used to set the orientation either it horizontal or vertical, but the flow pane has the horizontal orientation defaulty.

```
Flowpane fp = new Flowpane();
Button b1 = new Button("B1");
Button b2 = new Button("B2");
scene S = new scene(fp, 300, 200);
fp.getchildren().add(b1)
fp.getchildren().add(b2);
St.setscene(S);
St.show();
```

Note:- we can set the orientation using following two values 1) orientation.horizontal

```
fp.setOrientation(orientation.HORIZONTAL)
```

Textflow:- It layouts text controls in a scene generally we can create text control using text class it has the following properties.

a) linespacing:- This properties shows to set the line space between Textlines for this we have setlinespacing method. If we have setlinespacing(5.0) then it will the set the linespace as 5.

b) TextAlignment: It is used to set Text Alignment as justifyed, center, left (or) right. This values will be set as follows.

Ex:- Set TextAlignment(TextAlignment.CENTER);

we can create a TextFlow layout object using following

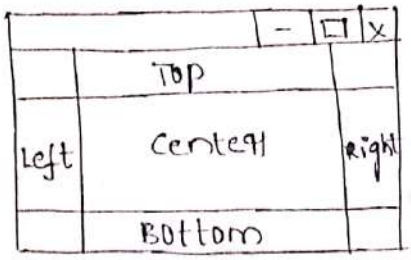
Syntax:- TextFlow Tf = new TextFlow();

Borderpane:- It is used to layout its children's at the border of layout the borders are like top, bottom, left, right and center. we can create a borderpane object as follows

Borderpane bp = new Borderpane();

It has the following methods

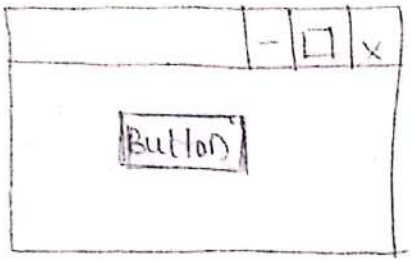
1) setTop:- It is used to set the Controls at the top borders.
2) setBottom:- It is used to set the Controls at the bottom of at the layout borders.
3) setLeft:- It is used to set the Control at the leftside of layout borders.
4) setRight:- It is used to set the Control at the right borders
5) setCenter:- It is used to set the Controls at the Center of borderpane.




```
Borderpane bp = new Borderpane();
    Button b1 = new Button("B1");
    Button b2 = new Button("B2");
    Scene s = new Scene(bp, 300, 200);
     bp.getChildren().add(b1);
     bp.getChildren().add(b2);
     bp.setTop(fp);
     bp.setBottom(fp);
```

Stackpane:- A stackpane is layouts its children on the top of another just like a stack it is also packaged in javafx.scene.layout."

The stackpane has the alignment property to move the position of a stackpane (setAlignment method). It also has one more property that is margin, we can set the margins using setMargin() method

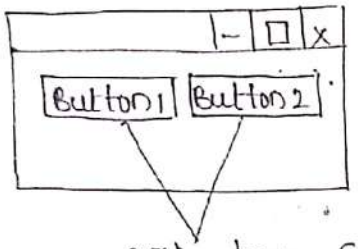


we can create a stackpane reference as follows

```
Stackpane sp = new Stackpane();
Scene S = new Scene(SP);
Sl.SelScene(S);
Sl.Show();
```

Tilepane: A Tile pane arranges the controls in our application as a uniform sized cells. it is similar to flowpane but a Flowpane creates cells or tiles based on different dimensions that is width and height of control. The tilepane creates uniform sized cells i.e., same dimension controls (width and height are same)

for Example, (Tile - cell)



Both have same dimensions (i.e., width and height)

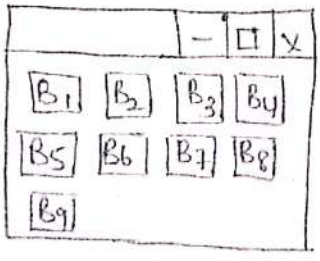
we can create an object as follows

```
Tilepane tp = new Tilepane();
```

* It has the following properties

1) TileAlignment:- we can set the alignment of tile
   SetTileAlignment().
2) Tilewidth:- this property shows the width of Tile
3) TileHeight:- this property shows height of Tile.
4) prefcols:- This shows preferrable no. of columns when we have an horizontal Tilepane.
5) preferrable rows: this shows preferrable no. of rows when we have an vertical Tilepane.

```
Setprefcols(4);
```

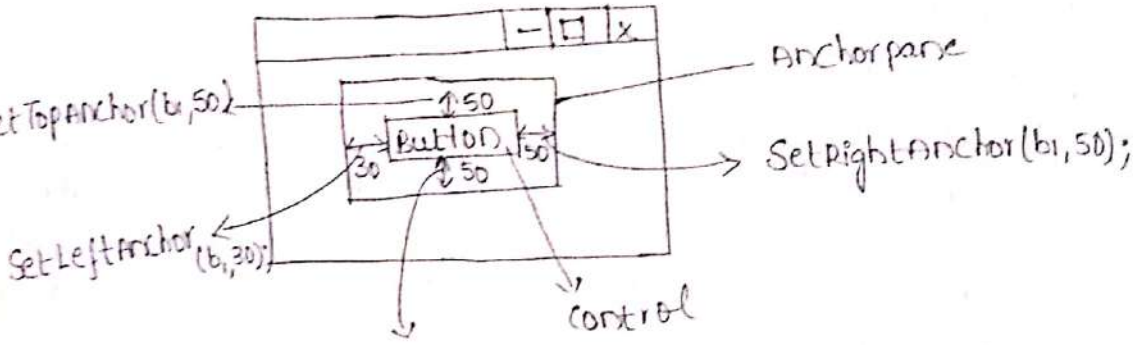


6) PrefTile width:- This property shows the PrefTile width.
7) PrefTile Height: This shows the preferrable Height of the Tile.

Anchorpane:- The Anchorpane Anchored its childrens at a particular distance of Anchored pane.
we can create an Anchorpane objects as follows,

Anchorpane ap = new Anchorpane ();

*It has four methods in order to maintain distance from its borders to place a control with the top distance 50 from the pane we have to use setTop Anchor() method. Likewise, we can also maintain the distance from the pane using setBottomAnchor(), setLeftAnchor(), setRightAnchor().

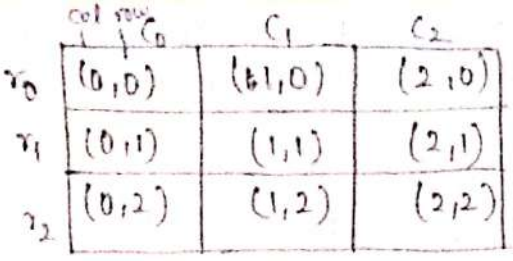


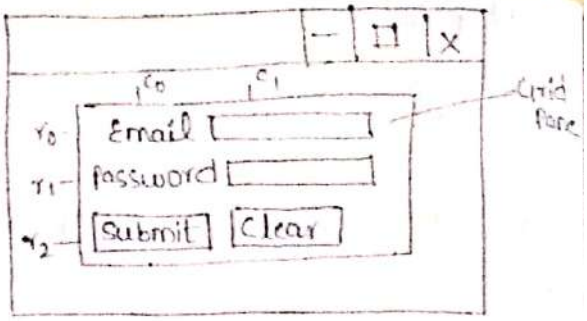
*The above four methods has only two parameters the first parameter is node and second parameter is distance from its pane.

*Grid pane: The Gridpane arranges the nodes in our application as a grid of rows and columns we can create Gridpane object as follows. Gridpane gp=new Gridpane();

*It has the following properties:.
1) Alignment:- we can change the Alignment of using SetAlignment() method.
2) Hgap:- This property shows the horizontal gap between the Columns
3) Vgap:- This property shows the vertical gap between the rows
4) GridLinesVisible:- This property shows the Gridlines to be visible in our application.

A gridpane has following cell values





l_1, l_2 are labels and tf_1, tf_2 are text fields b_1, b_2 are buttons

```
Gridpane gp = new Gridpane();
gp.add(l1, 0,0);
gp.add(l2, 0, 1);
gp.add(tf1, 1,0);
gp.add(tf2, 1,1);
gp.add(b1, 0, 2);
gp.add(b2, 1,2);
```

Note: All above layout classes are sub classes of pane class. A pane class is also a sub class of the region class. And we can also use Group class instead of layout classes, but it has its own specifications. It won't provide any specification to its children.

Application class and lifecycle methods:-

Every javafx application should extends an Application class. This class is packaged in javafx.Application. It has the following three life cycle methods these methods are implemented by the class which Extends Applicationclass.

```
void init()
abstract void start(Stage primaryStage)
void stop()
```

init():- The init() method is used to perform various inilizations but it cannot be used to build a stage or to construct a scene it is not compulsory to implement init() method because it is a default and empty version and it is not a abstract type.

Generally the init() method is called by main thread (or) launcher thread.

start():- The start() method will be called after

execution of init() method start() method will be used to construct a scene it is called whenever our application begins always the start() method will called by an application thread. It has only one parameter which is a referrence of stage class. this stage is automatically created by runtime system and it is called as primary stage. we must implement start method inside our class because its type is abstract type.

stop():- The stop(); method will be called whenever an application is terminated or shutdown it is also called by an application thread.

launching the javafx application:-

* we can launch javafx application by using an application class method which is called launch() method. It has the following syntax

public static void launch(string[] args)

* It has a string arguments which are the command line arguments

```
public static void main(string[] args)
{
 launch(args); // Application.launch(args);
}
```

Javafx application skeleton:-

1) Javafx application skeleton illustrates the demonstration of javafx application and lifecycle methods of application class. we have the following steps to create a application skeleton.

a) import we have to import all supported javafx packages.

b) Create a class which extends an application class

c) Implement init(), start(), and stop() methods

d) Inside start() method we have to do the following things

I) Create a layout class object

II) Create a scene class object and pass layout class object as a parameter to the Scene class.

III) Add scene Class object to the stageclass using setScene() method.

IV) We can also set title to the Stage using setTitle() method.

V) To show the Created stage we have to call show method.

6) Inside main() method we have to Call launch() method.

Programme:

```
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
Public Class Javafx skeleton extends Application
{
Public void init()
{
System.out.println("Init() method started");
}
Public void start(stage ps)
{
System.out.println("start() method Called");
 Flowpane fp = new Flowpane();
 Scene s = new Scene(fp, 300,400);
 ps.setscene(s);
 ps.SetTitle("Application skeleton");
 ps.show();
}
Public void stop()
{
System.out.println("stop() method Called");
}
public static void main(String[] args)
{
System.out.println("launching Javafx application");
launch(args);
}
}
```

Compiling and Running Javafx Programme:-

We can Compile and run Javafx programmes as a normal Java programs but before going to Compile we must check whether our system has JDK 1.8 or not After installation of JDK 1.8 we have to set path we can Compile Javafx programmes as follows

javac classname.java

we can run Javafx program has follows

java classname

* we can also Compile and run javafx programmes by using different environments these environments are created by IDE such as EclipseIDE and NetBeans IDE. In IDE's no need to use javac and java directly we can Compile and run javafx programmes.

Application thread:-

Generally the init() method will not be used to construct a scene or to build a stage even the application constructor also cannot be used. To construct the scene (or) to build a stage, because both init() method and application constructor will depends on main thread or launcher thread. we cannot construct and build a scene using a launcher thread instead of those we are using start() method which is depends on application thread. if any modification is require to our javafx application all these modifications are done by application thread. Even the stop() method is also run by the application thread.

Part-II Simple Javafx Controlls: Label

In javafx we have different Controls like buttons, labels, textfields, text, Checkboxes, radio buttons, images etc. All these Controlls are packaged in javafx.scene.Control. It is very easy to start will the label because a label is a simple text or message.

In javafx we can create a label using Labelclass it has the following three types of constructors.

1) Label ()
2) Label(String)
3) Label (String, Node)

But generally will use the second type of constructor because directly we can add a text to the label.

Label(String txt)

Ex:- Label l = new Label("This is javafx");

* It has the following methods

1) setText():- It is used to set a Text for label

2) setLabelFor():- It is used to set a Label for a node

write a javafx programme to add a label to the javafx application.

```
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
public class JavafxLabel extends Application
{
public void start(Stage ps)
{
Label l = new Label("Javafx label Demo");
Stackpane sp = new Stackpane();
  sp.getChildren().add(l);
  Scene s = new Scene(sp, 300, 400);
   ps.setScene(s);
   ps.setTitle("Javafx Label");
   ps.show();
  }
public static void main(String[] args)
{
 Launch(args);
}
}
```

write a javafx programme to add a button to the javafx application.

```
import javafx.application.*;
```

```
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.button.*;
public Class JavaFxButton extends Application
{
public void start(Stage ps)
{
Button b = new Button("JavaFx Button");
 StackPane sp= new StackPane();
   sp.getChildren().add(b);
 Scene s= new Scene(sp,300, 400);
   ps.setScene(s);
   ps.setTitle("JavaFxButton");
   ps.show();
  }
public static void main(String[] args)
 {
 launch(args);
 }
}
```

Using Buttons and Events:

In previous section we have created the label control but this label cannot be used to generate an event, we have different javafx controls in order to generate events, the most common control is button, because the button is a good way to demonstrate the events.

for Example: Button clicked, mouse pressed, mouse released, mouse entered, mouse exited, key pressed key released, key typed, scroll up, scroll down, view the list click the checkboxes etc.

Event basics:- In javafx GUI application, the user is always interact with the controls when user is interacted with the controls, the respective events will be occured.

Generally, we have the following two types of event.

1) Foreground event: These events are generated when the user interacted with the controls.
Ex:- Key pressed, Button clicked, mouse moved etc.

2) Background events: These events are generated by the end user for example operating system interrupted s/w or h/w failure and time expire etc.

In Javafx when we are dealing with the events we have to know the following things.

1) Source:- A source node from which an event is generated.

2) Target: The node on which the event is occured, the target may be a window (stage), or scene, or a node.

3) Type: It refers the type of event is generated.
Ex:- ActionEvent, MouseEvent, KeyEvent, WindowEvent and DragEvent etc.

EventHandler:- A EventHandler is a mechanism to process or control the events, and decide what happens after an event is generated. we have an event handler interface in javafx.Eventpackage it defines a simple method called Handle

interface EventHandler<T extends Event>

where T is a eventType for the button Control we have an event type ActionEvent. It is a subclass of an Eventclass which is in javafx.Event package.

```
void handle(T eventObj)
{
=
}
```

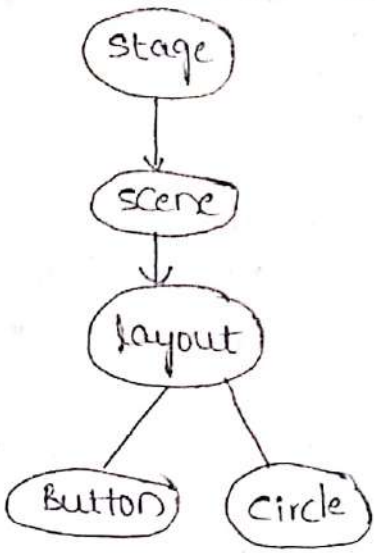
Inside handle method will implement the logic to do something when an event is occured.

Phases of EventHandler:

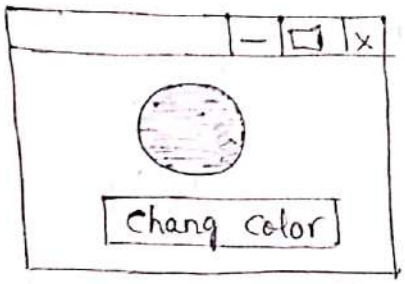
we have the following three phases in order to handle the events.

1) Route Construction:- After an event is generated the route will be constructed based on the event

dispatch chain. In Event dispatch chain the forme most node is stage, the last nodes are controls. suppose In javafx application, if we have an circle on button, then the route construction as follows



Ex:-



2) Event Capturing phase:- After an event occurs the event will be traverse from top to bottom along with route. we can register an event using a filter, if any one of the node is registered then the event will be handled by the node. if none of the node is registered for the event. Then the target node will be handle the event.

3) Event Bubbling phase: After an event is generated, the event is transversed from bottom node to top node. If any one of the node is register for the handler then the event will be process or handled. If none of this nodes are register then the event will be handler up to the root node and then it will be processed by root node.

Event handler and filter:-

the event filter is used in event capturing phase and handler is used in event bubbling phase. A node can registered more than one handler or(or) filter.

Add filter (or) handler:

To add a filter to the node we can use the following methods

void addEventFilter(Event type, EventHandler);

It has two parameters one is event type and another one is event handler.

for Example, btn is a object of button control then we can add a filter to the button as follows.

btn.addEventFilter(ActionEvent.ACTION, handler);

To add an event Handler to the button control we have to do following way

void addEventHandler(Event type, EventHandler);

for Example, if btn is a object of button control then we can add a Handler to the button as follows

btn.addEventHandler(ActionEvent.ANY, handler);

for the button control we have a seperate method called setonAction() it is also used for registering it has only one parameter the parameter type is EventHandler type.

void setOnAction(EventHandler<ActionEvent> handler);

Ex: write a javafx program to handle button related events (ActionEvent)

```
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.scene.event.*;
import javafx.scene.Geometry.*;
import javafx.scene.shape.*;
import javafx.scene.paint.*;
Public class JavafxButton extends Application
{ int colorCnid = 0;
Public void start(stage ps)
{
System.out.println("start() method called");
circle C = new Circle();
Button btn = new Button();
C.setFill(Color.GREEN);
```

```
c.setRadius (25);
c.setStroke(Color.BLUE);
btn.setText("changecolor");
Gridpane gp = new Gridpane();
 gp.add(c, 0,0);
 gp.add(btn,0,1);
 gp.setVgap(50);
 gp.setMinSize(400, 500);
 Scene s= new Scene(gp, 400, 500);
  ps.setScene(s);
  ps.setTitle("JAVAFX BUtton");
  ps.show();
 } Color[] clr ={ Color.RED, Color.PINK, Color.PURPLE,
                  Color.BLACK, Color.SKYBLUE};
  btn.setOnAction(new EventHandler<ActionEvent>(){
    public void handle(ActionEvent ae)
    {
     c.setFill(clr[clrid]);
     clrid++;
    if(clrid == clr.length)
     clrid=0;
    }
  });
}
public static void main(String[] args)
{
System.out.println("Launching JavaFx Application");
launch(args);
}
}
(or)
EventHandler<ActionEvent> event = new EventHandler<Action
                                              Event>()
{
Public void handle(ActionEvent ae)
 {
  c.setFill(clr[clrid]);
  clrid++;
  if(clrid == clr.length)
```

```
    clrid = 0;
   }
  }
 btn.setOnAction(event);
      (or)
 btn.addEventFilter(ActionEvent.ACTION, event);
 btn.addEventHandler(ActionEvent.ACTION, event);
                (or) another method
 btn.setOnAction((ActionEvent ae) ->
  {
  c.setFill(clr[clrid]);
  clrid++;
  if(clrid == clr.length)
  clrid = 0;
 });
```

Introducing the button Control:

We can create a button control using button class which is packaged in javafx.scene.control. It has the following three types of constructor.

Button(); → Empty
Button(String); → one parameter
Button(String, Node); → two parameters

* The first type of constructor we don't have any parameters it is an empty version if you want to set any text on the button which is created by an empty version. we have to called, setText() method.

Ex: Button b = new Button();
b.setText("SUBMIT");

SUBMIT

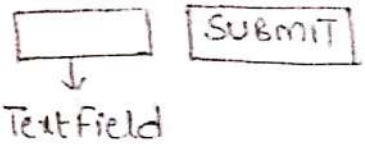
* In the second type of constructor we have one parameter. parameter type is string. instead of using setText() method we can directly mention the text to the constructor itself.

Button b = new Button("SUBMIT");

SUBMIT

* The third type of constructor we have two parameters the first parameter is String and the second parameter is node in this we can create a button besides the specified node.

```
TextField tf = new TextField();
Button b = new Button("SUBMIT", tf);
```



Demonstrating Event handling and the button:

We can handle the events which are generated by a button this type of events are specified by an event type "ActionEvent". The ActionEvent class hds only two parameters. Those are ACTION, ANY. we can use these two Constants as follows

ActionEvent.ACTION
ActionEvent.ANY

we have the following method to perform an action whenever an event is generated.

```
final void setOnAction(EventHandler<ActionEvent> handlerobj);
```

```
Button b = new Button("CSE");
b.setOnAction(new EventHandler<ActionEvent>(){
    public void handle(ActionEvent ae)
    {
    System.out.println("Computer science & Engineering");
    }
});
```

Drawing directly on Canvas:-

Javafx provides the automatic rendering this automatic rendering approach is helpful when we are drawing a different graphical objects such as lines rectangle and ovals etc. But in Swings and AWT's we have repaint() method. In order to display a window or drawing different shapes on a window. for Every request repaint() method stores the window content and it redisplayed whenever we request to paint.

* In Javafx we don't have repaint method it will perform the automatic rendering it simply keep track of what we display on the screen. This is called Retaining mode.

* we can create a drawing surface using Canvas Class which is packaged in javafx.scene.Canvas.

we can draw different graphical objects on the canvas to draw different shapes we have one more class called GraphicsContext. It is also packaged on

javafx.scene.canvas.

we can call different methods using reference of graphics context class. This reference is return by canvas class method

i.e., c.getGraphicsContext2D()

GraphicsContext getGraphicsContext2D()

* Before drawing a shape on the canvas first we have to create a canvas class object it has two types of constructors.

Canvas(); → default constructor

Canvas(double w, double h); → two parameter constructor

where w is a width of canvas and h is a height of canvas. we can also said the width and height of the canvas using setWidth() and setHight() methods

* second, we have to create reference of GraphicsContext class using this reference we will call GraphicsContext class methods to draw Graphical objects on the canvas

* A graphicsContext class has the following methods these methods are used to draw the basic shapes on the canvas we can also add different effects and transformations using GraphicsContext class methods.

StrokeRect():- It is used to create rectangle shape on the canvas.

void strokeRect(double x, double y, double width, double height);

where x and y values are pixels.

fillRect():- It is used to create a filled rectangle

void fillRect(double x, double y, double width, double height);

StrokeRoundRect():- It is used to create a Round rectangle shape.

void strokeRoundRect(double x, double y, double width, double height, double arc width, double arc height);

fillRoundRect():- It is used to create a filled round Rectangle.

void fillRoundRect(double x, double y, double width, double height, double arc width, double arc height);

strokeOval():- It is used to create an oval shape

void strokeOval(double x, double y, double width, double height);

fillOval():- It is used to create filled oval shape

void fillOval(double x, double y, double width, double height);

strokeLine():- It is used to draw a line

void strokLine(double startx, double starty, double endx, double endy);

strokeArc():- It is used to draw an arc shape.

void strokeArc(double x, double y, double w, double h, double startAngle, double angleExtent, ~~Arcs~~ ArcType.OPEN);

where Arc-types are open and round.
OPEN ROUND.

fillArc():- It is used to create a filled Arc

void fillArc(double x, double y, double w, double h, double startAngle, double angleExtent, ArcType.ROUND);

strokePolygon():- It is used to draw a polygon shape.

void strokPolygon(double xpoints[], double ypoints[], int pointslength);

fillPolygon():- It is used to create a filled polygon

void fillpolygon(double xpoints[], double ypoints[], int points, length);

strokeText():- It is used to draw a text on the canvas.

void strokeText(String str, double x, double y, double MAXwidth);

fillText():- It is used to create a filled text on the canvas.

void fillText(String str, double x, double y, double maxwidth);

setFont():- It is used to set font properties

void setfont(font f);

setFill():- It is used to fill a particular shape with to specify a colour.

void setfill(Paint p);

strokefill():- It is used to fill the stroked lines

void strokefill (paint p);

write a Javafx program to draw directly on canvas.

```
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.canvas.*;
import javafx.scene.shape.*; import javafx.scene.paint
public class JavafxCanvas extends Application
{
public void start(Stage ps)
{
system.out.println("start() method called");
canvas c = new canvas(300, 200);
GraphicsContext gc = c.getGraphicsContext2D();
 gc.setlinewidth(2.0);
 gc.setFill(Color.RED);
 gc.strokeRoundRect(10, 10, 50, 50, 10, 10);
 gc.fillRoundRect(100, 10, 50, 50, 10, 10);
 gc.strokeoval(10, 70, 50, 30);
 gc.filloval(100, 70, 50, 30);
 gc.setfill(Color.GREEN);
 gc.strokeline(200, 50, 300, 50);
 gc.strokeArc(320, 10, 50, 50, 40, 50, ArcType.ROUND);
 gc.fillArc(320, 70, 50, 50, 00, 120, ArcType.OPEN);
 Pane p = new Pane();
 p.getchildren().add(c);
p.setStyle("fx-Padding:10;"+
          "-fx-border-style: solid inside;"+
          "-fx-border-width: 5;"+
          "-fx-border-insets: 5;"+
          "-fx-border-radius: 2;"+
          "-fx-border-Color: blue;");
    Scene s = new Scene(p, 400, 300);
     ps.setscene(s);
     ps.setTitle("Javafx Canvas");
```

```
ps.show();
}
public static void main(String[] args)
{
System.out.println("launching Javafx Application");
launch (args);
}
}
```

Note: A Canvas is a transparent that means if we create two canvases and then you have to keep one on another so that the top Canvas will show the both Canvas Shapes.