WOLDIA UNIVERSITY
INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING

ADVANCED PROGRAMMING
CHATER 1 GUI

LECTURE BY DEMEKE G.

AY-2017

Introduction

Definition of User Interface

- A graphical user interface (GUI) is a digital interface in which a user interacts with graphical components such as icons, buttons, and menus.
- In computer science and human-computer interaction, the user interface of a computer program refers to the graphical, textual and auditory information that program presents to the user.
- The user employs several control sequences (such as keystrokes with the computer keyboard, movements of the computer mouse, or selections with the touch screen) to control the program.

Cont..

Types of User Interfaces

Command-Line Interface (CLI)

- ✓ The user provides the input by typing a command string with the computer keyboard and the system provides output by displaying text on the computer monitor.

Graphical User Interface (GUI)

- ✓ The use of pictures rather than just words to represent the input and output of a program.
- ✓ Input is accepted via devices such as keyboard and mouse.

GUI individual Assignment 1

File Edit View Navigate

Full Name

Gender Male Female

Language you ... AMharic English Germeny

Department Computer Science

Year of Entry

Hobbies

Nationality Type Ethiopian Foriegn

Date of Birth

Favorite Foods Tuna Steak

Save Reset Form

Graphical user interface

- ✓ Graphical user interface (GUI) presents a user-friendly mechanism for interacting with an application.
- ✓ Gives an application a distinctive "look" and "feel".
- ✓ Providing different application with **consistent, and intuitive user interface**
- ✓ GUI allows users to be
  - ~~ somewhat familiar with an application
  - ~~ they can learn it more quickly

Cont..

- In java, to develop an application that has a graphical user interface ,we have use GUI components.
- Among some of java GUI components(packages)

  -- AWT [abstract window toolkit]

  -- Swing

  -- JavaFX
- **AWT (Abstract Window Toolkit)** package is an older package designed for doing windowing interfaces.

Cont..

- ✓ Swing is an improved version of the AWT
- ✓ However, Swing did not completely replace the AWT package
- ✓ Some AWT classes are replaced by Swing classes, but other AWT classes are needed when using Swing
- ✓ Swing designed using a particular form of object-oriented programming that is known as **event-driven programming**.
- ✓ Event-driven programming is a programming style that uses a signal-and-response approach to programming.

JavaFX

Introduction

- ✓ JavaFX is next-generation java GUI package
- ✓ allow the developers to create and deploy rich internet applications [RIAs] that run seamlessly across screens (desktop, mobile, or IP TV),providing a uniform user experience that behave consistently across multiple platforms.
- ✓ **RIAs, by (Wikipedia)** definition, are web applications that have most of the characteristics of desktop applications, typically delivered through web-browser plug-ins or independently via sandboxes or virtual machines

cont..

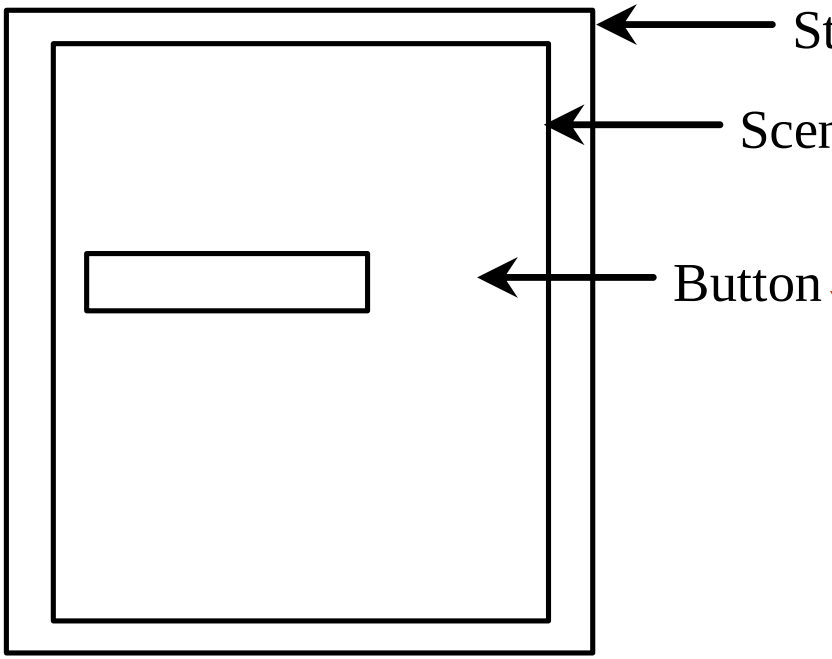
- JavaFX platform provides **a rich set of graphics** and **media API** with **high performance hardware-accelerated graphics** and media engines that simplify the development of data-driven enterprise client applications.
- JavaFX **applications will run on any desktop and browser** that runs the **JRE** and easily integrate with Java Platform, Mobile Edition (**Java ME**), opening the door to billions of mobile phones and other connected devices.

cont..

- ✓ JavaFX also **leverages the other benefits of the Java platform,** such as object-orientation, inheritance, polymorphism, a well-established security model, well-defined exception handling, memory management through garbage collection, and the mature Java Virtual Machine (JVM).
- ✓ The **reason why developers select javafx** is JavaFX platform contains an essential set of tools and technologies that enable developers and designers to collaborate, create, and deploy applications with expressive content.

Basic Structure of JavaFX

- **javafx.application.Application** class defines the essential framework for writing JavaFX programs
- Every JavaFX program is defined in a class that **extends** javafx.application.Application

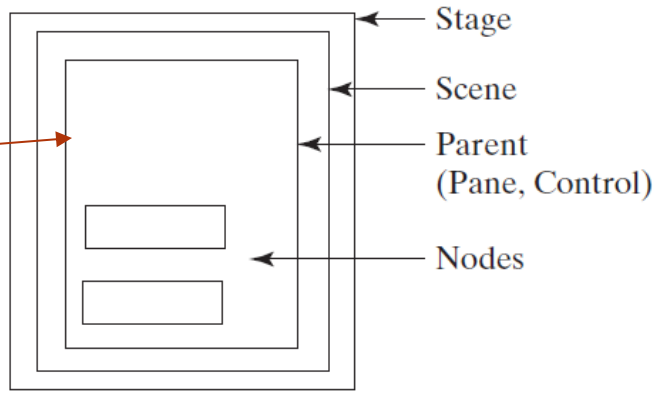


Stage object is automatically created by JVM when the application is launched – considered the **window –** you can have multiple stages.

Scene object is the container for the content – considered the **frame** in the window – can have only one scene per stage or window

Button object (or node) is a visual component such as a shape, image, UI control, groups, and panes. Used to invoke some type of action

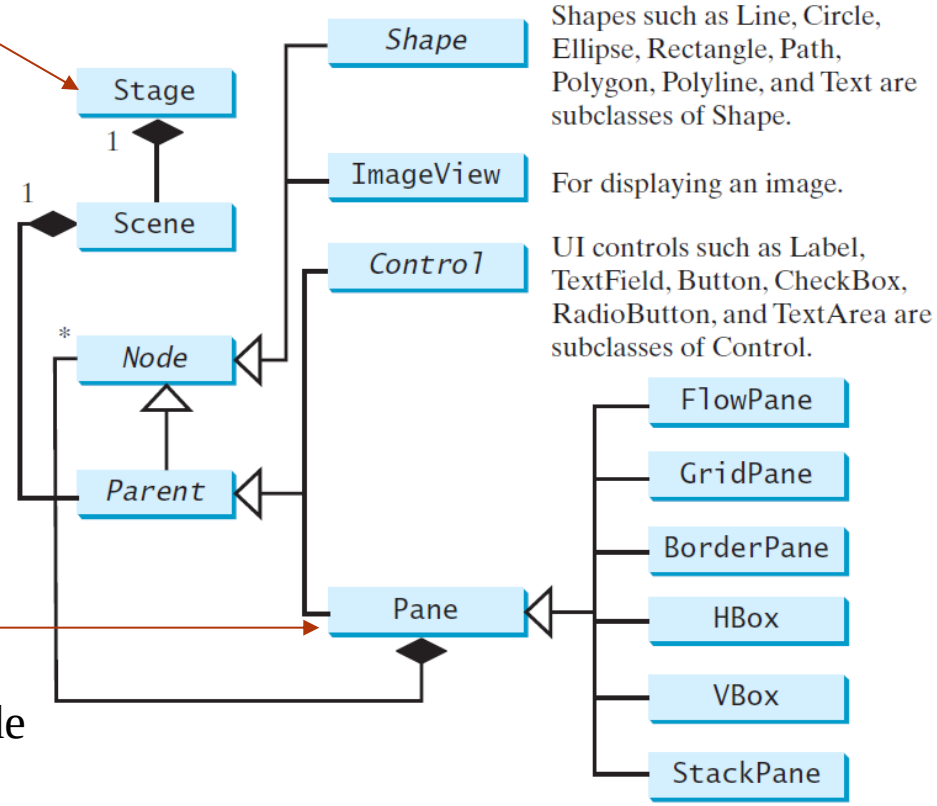
Container classes called **panes** can be used to help layout the nodes within a scene. **Control** refers to a label, check box, radio button, text field and etc. A **group** can be used to group a set of nodes



Panes, UI Controls, and Shapes

(1) One Scene per Stage

(2) A Scene can contain a Control, Group or Pane



(b)

(3) A Pane or Group can contain any subtype of Node

(4) Will cover the various types of Panes later

Basic Structure of JavaFX Program

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;

public class MyJavaFX extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a button and place it in the scene
    Button btOK = new Button("OK");
    Scene scene = new Scene(btOK, 200, 250);
    primaryStage.setTitle("MyJavaFX"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

Every JavaFX program extends javafx.application.Application

The main class overrides the start method defined in javafx.application.Application and JVM constructs an instance of the class and invoke the start method.

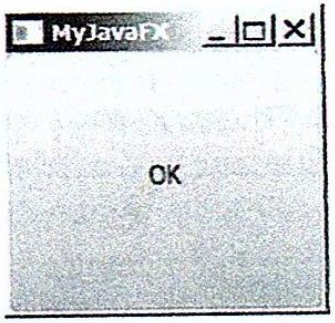
Create a button object and places it in a Scene object

Scene object created using the constructor – specifies width and height and places node in scene

The Stage object is automatically created by JVM when the app is launched

Name the Stage, set the scene in the stage, and display the stage.

The launch method is static method used for launching stand-alone JavaFX apps. Not needed if you run it from a command line.



Output of program displays a button in the window

JavaFX Program with Multiple Stages

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;

public class MultipleStageDemo extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a scene and place a button in the scene
    Scene scene = new Scene(new Button("OK"), 200, 250);
    primaryStage.setTitle("MyJavaFX"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage

    Stage stage = new Stage(); // Create a new stage
    stage.setTitle("Second Stage"); // Set the stage title
    // Set a scene with a button in the stage
    stage.setScene(new Scene(new Button("New Stage"), 200, 250));
    stage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

Every JavaFX program extends javafx.application.Application

The main class overrides the start method defined in javafx.application.Application and JVM constructs an instance of the class and invoke the start method.

The first Scene object is created using the constructor – specifies width and height and places button in scene

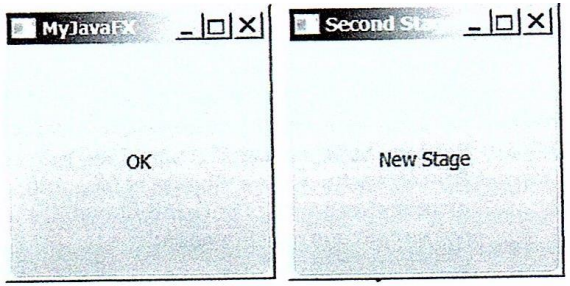
The first Stage object is automatically created by JVM when the app is launched

Name the first Stage, set the scene in the stage, and display the stage.

A new stage is created.

Name the second Stage, set the scene in the stage, places button in Scene, and display the stage.

The launches JavaFX app. Identical main method for all every JavaFX app.



Output of program displays multiple stages

1. Stage Class

- A stage in JavaFX is a top-level container that hosts a scene, which consists of visual elements.
- The Stage class in the javafx.stage package represents a stage in a JavaFX application.
- The stage has four variables that either affect its appearance or reflect its active state.
- When a JavaFX application is launched, a stage known as the primary stage is automatically created.
- A reference to this stage is passed to the application's start method via the primaryStage parameter:

Showing the primary stage

```
 import javafx.application.Application;
import javafx.stage.Stage;
public class First_stage extends Application {
public static void main(String[] args) {
Application.launch(args);
}
@Override
public void start(Stage stage) {
// Do  write any code here
}
}
```

Setting the Bounds of a Stage

- The bounds of a stage consist of four properties: x, y, width, and height.
- The x and y properties determine the location (or position) of the upper-left corner of the stage.

```
@Override
public void start(Stage stage) {
Group root = new Group(new Button("Hello"));
 Scene scene = new Scene(root);
stage.setScene(scene);
 Rectangle2D bounds = Screen.getPrimary().getVisualBounds();
double x = bounds.getMinX() + (bounds.getWidth() - stage.getWidth())/2.0;
double y = bounds.getMinY() + (bounds.getHeight() - stage.getHeight())/2.0;
stage.setX(x);
stage.setY(y);    stage.show();
  }
```

Contt..

- The width and height properties determine its size.

```
    @Override
public void start(Stage stage) {
stage.setTitle("A Sized Stage with a Sized Scene");
Group root = new Group(new Button("Hello"));
Scene scene = new Scene(root, 300, 100);
stage.setScene(scene);
stage.setWidth(400);
stage.setHeight(100);
stage.show();
}
```

Initializing the Style of a Stage

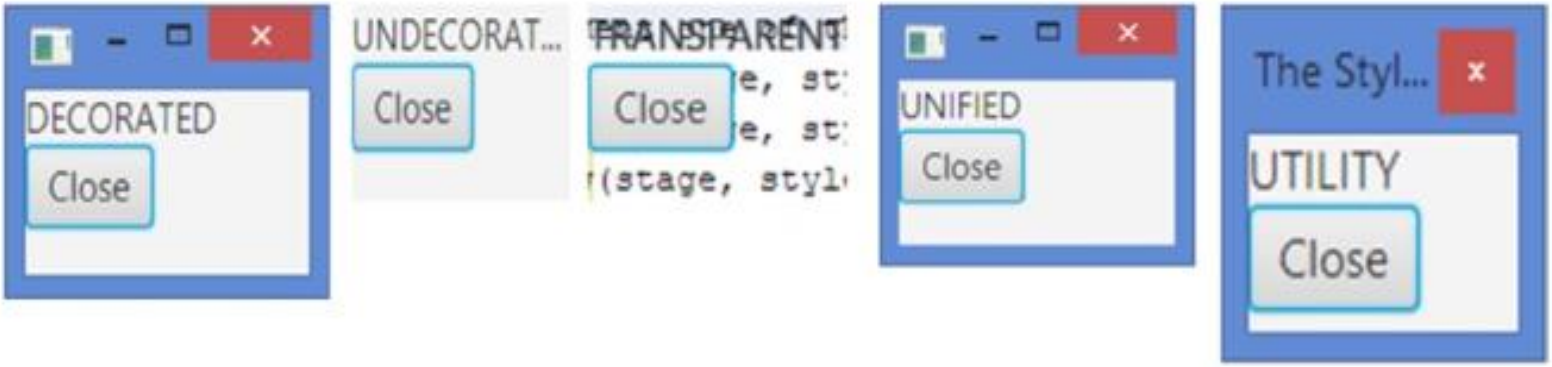
- The area of a stage can be divided into two parts: **content area** and **decorations**.
- The content area displays the visual content of its scene.
- Typically, **decorations consist of a title bar and borders**.
- The presence of a title bar and its content varies depending on the type of decorations provided by the platform.
- Title bar may consists of title word, minimize, maximize, restore, and close button.

Cont..

- the style attribute of a stage determines its background color and decorations. Based on styles, you can have the following five types of stages in JavaFX:
- **Decorated** stage has a solid white background and platform decorations.
- An **undecorated** stage has a solid white background and no decorations.
- A **transparent** stage has a transparent background and no decorations.
- A **unified** stage has platform decorations and no border between the client area and decorations; the client area background is unified with the decorations.

Cont..

- ✓ To see the effect of the unified stage style, the scene should be filled with Color.TRANSPARENT. Unified style is a conditional feature.
- ✓ A **utility** stage has a solid white background and minimal platform decorations.



- StageStyle.DECORATED
- StageStyle.UNDECORATED
- StageStyle.TRANSPARENT
- StageStyle.UNIFIED
- StageStyle.UTILITY

stage.initStyle(StageStyle. UTILITY)

Initializing Modality of a Stage

- In a GUI application, you can have two types of windows: **modal** and **modeless**.
- In **modal** window the user cannot work with other windows in the application until the modal window is dismissed.
- If an application has multiple **modeless** windows showing, the user can switch between them at any time.
- Modality of a stage is defined by one of the following three constants in the Modality enum in the javafx.stage package:
  - **Modality.NONE --** stage that does not block any other window.
  - **Modality.WINDOW_MODAL** : a stage that blocks input events from being delivered to all windows from its owner (parent) to its root. Its root is the closest ancestor window without an owner.
  - **Modality.APPLICATION_MODAL :** a stage that blocks input events from being delivered to all windows from the same application, except for those from its child hierarchy.

Cont..

- can set the modality of a stage using the **initModality(Modality m)** method of the Stage class as follows:

```
Stage stage = new Stage();
stage.initModality(Modality.WINDOW_MODAL);
```

Stage Title and icon

- Image icon = new Image (getClass(). getResourceAsStream ("wdulogo.png"));

```
primaryStage.getIcons().add(icon);
primaryStage.setTitle("Title");
```

Cont..

Closing the Stage

- The stage can be closed either by calling close() method in the program(code) or by manually close button when using the app.
- The setMinWidth(), setMinHeight(), setMaxWidth(), and setMaxHeight() methods of the Stage class let you set the range within which the user can resize a stage. Calling the setResizable(false) method on a Stage object prevents the user from resizing the stage.

Cont..

- A stage may enter full-screen mode by calling the setFullScreen(true) method.
- When a stage enters full-screen mode, a brief message is displayed about how to exit the full-screen mode: You will need to press the ESC key to exit full-screen mode.
- You can exit full-screen mode programmatically by calling the setFullScreen(false) method.
- Use the isFullScreen() method to check if a stage is in full-screen mode

Cont..

2. Scene

- ✓ The scene is the part of the stage that hosts the user interface of a JavaFX application.
- ✓ Every stage has a single scene.
- ✓ Scene is the container of the content inside the stage.
  - Setting the width and height of the scene is setting the width and height of the entire window(stage).

Cont..

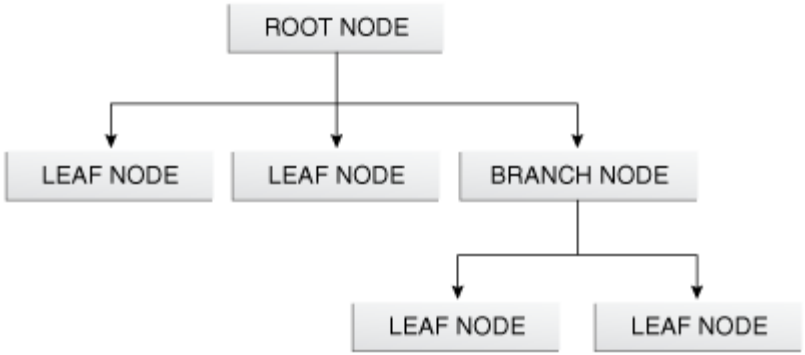
Variables	Type	Description
stage	Stage	Points to the stage hosting this scene.
x	Number	The X coordinate of the scene relative to the stage
y	Number	The Y coordinate of the scene relative to the stage
width	Number	The width of the scene
height	Number	The height of the scene

The JavaFX Scene Graph API

- the underlying framework that renders your GUI to the screen.
- makes graphical user interfaces easier to create, especially when complex visual effects and transformations are involved.
- a scene graph is a tree data structure, most commonly found in graphical applications and libraries such as vector editing tools, 3D libraries, and video games.
- the graphical objects are managed by the scene graph.

Cont..

- The individual items held within the JavaFX scene graph are known as nodes.
- Each node is classified as either a branch node(meaning that it can have children), or a leaf node (meaning that it cannot have children).
- The first node in the tree is always called the root node, and it never has a parent.



Setting the Cursor for a Scene

- An instance of the javafx.scene.Cursor class represents a mouse cursor.
- The Cursor class contains many constants, for example, HAND, CLOSED_HAND, DEFAULT, TEXT, NONE, WAIT, for standard mouse cursors. The following snippet of code sets the WAIT cursor for a scene:
    - Scene scene;
    - scene.setCursor(Cursor.WAIT);

Javafx Layouts

- A JavaFX application can manually lay out the UI by setting the position and size properties for each UI element.
- However, an easier option is to make use of layout panes.
- A layout pane is a node that contains other nodes, which are known as its children (or child nodes).
- The JavaFX SDK provides several layout panes for the easy setup and management of classic layouts such as rows, columns, stacks, and others.
- As a window is resized, the layout pane automatically repositions and resizes the nodes that it contains according to the properties for the nodes.

cont...

- Pane
- BorderPane
- HBox
- VBox
- GridPane
- FlowPane
- TilePane
- StackPane

Pane layout

- ✓ The base class for all JavaFX layouts, which can hold multiple nodes without any specific layout arrangement
- ✓ A Pane provides the following layout features:
  - It can be used when absolute positioning is needed. By default, it positions all its children at (0, 0).
  - You need to set the positions of the children explicitly.
  - It resizes all resizable children to their preferred sizes.
- ✓ The instances of the Pane class and its subclasses can add any children.

```
Pane pane = new Pane();
Button button = new Button("ClickMe");
 button.setLayoutX(50);
button.setLayoutY(100);
pane.getChildren().add(button);
```

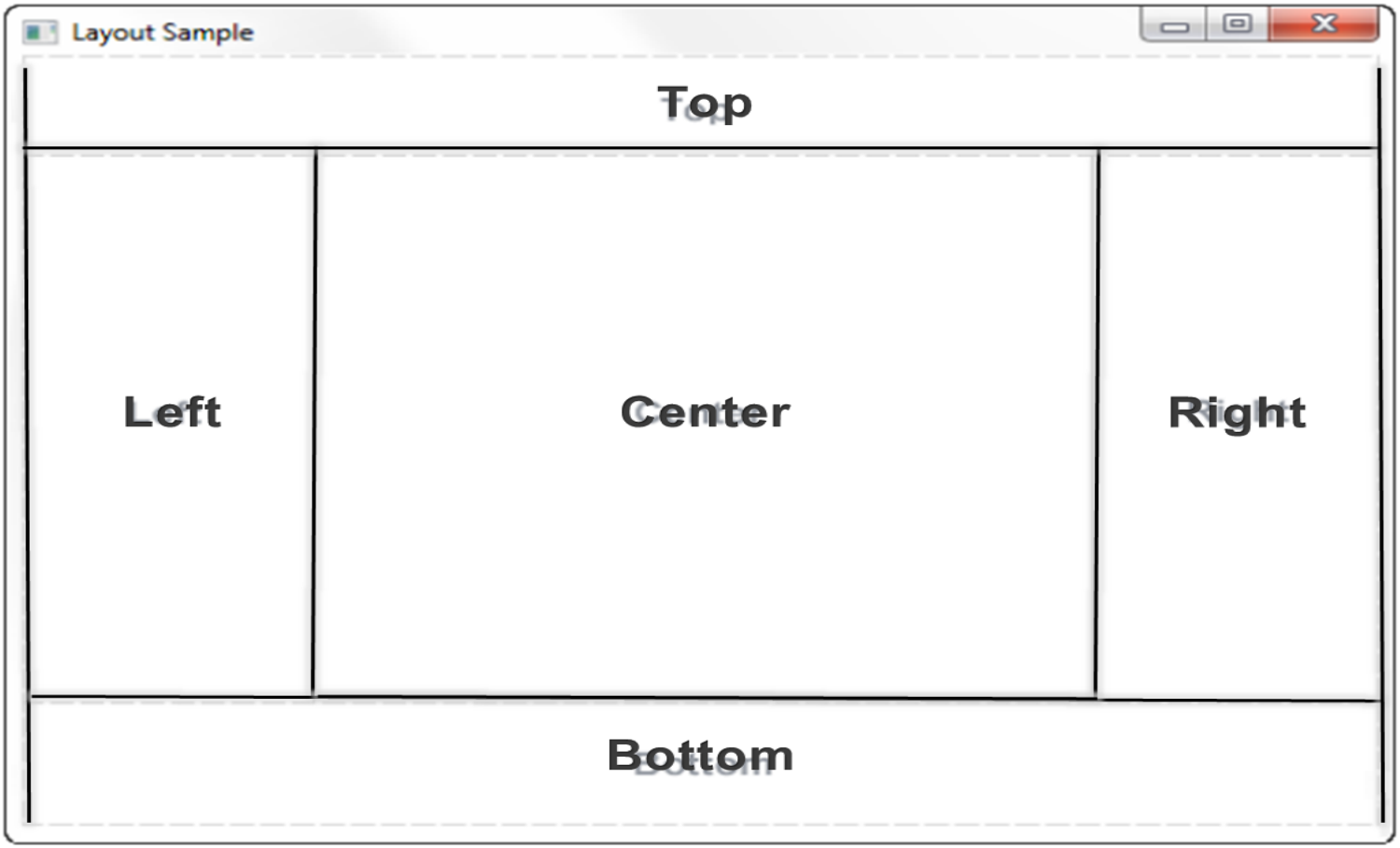
BorderPane

- The BorderPane layout pane provides five regions in which to place nodes: **top, bottom, left, right, and center**.
- The regions can be any size.
- If your application does not need one of the regions, you do not need to define it and no space is allocated for it.
- A border pane is useful for the classic look of a tool bar at the top, a status bar at the bottom, a navigation panel on the left, additional information on the right, and a working area in the center.

Cont..

- If the window is larger than the space needed for the contents of each region, the extra space is given to the center region by default.
- If the window is smaller than the space needed for the contents of each region, the regions might overlap.
- The overlap is determined by the order in which the regions are set.
- For example, if the regions are set in the order of **left, bottom, and right**, when the window is made smaller, the bottom region overlaps the left region and the right region overlaps the bottom region.

BorderPane bpane3 = new BorderPane(center, top, right, bottom, left)



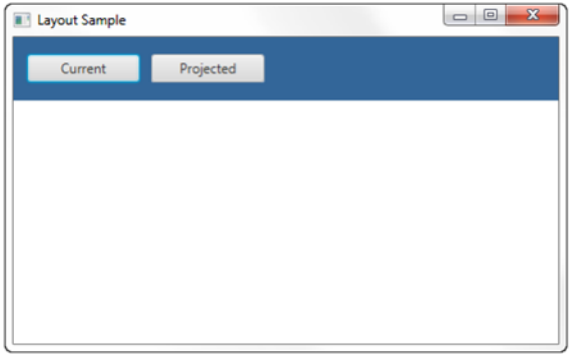
Example:

```
BorderPane borderPane = new BorderPane();
borderPane.setTop(new Label("Top Region"));
borderPane.setBottom(new Label("Bottom Region"));
borderPane.setLeft(new Label("Left Region"));
borderPane.setRight(new Label("Right Region"));
borderPane.setCenter(new Label("Center Region"));
```

HBox

- The Hbox layout pane provides an easy way for arranging a series of nodes in a single row (horizontally).
- The **padding** property can be set to manage the distance between the nodes and the edges of the **Hbox** pane.
- **Spacing** can be set to manage the distance between the nodes.
- The style can be set to change the background color.

```
HBox hbox = new HBox(); //creating HBox
Button bt1= new Button("Current");
Button btn2= new Button("Projected");
hbox.setPadding(new Insets(20));
hbox.getChildren().addAll(btn1, btn2); //adding buttons on the HBox
```



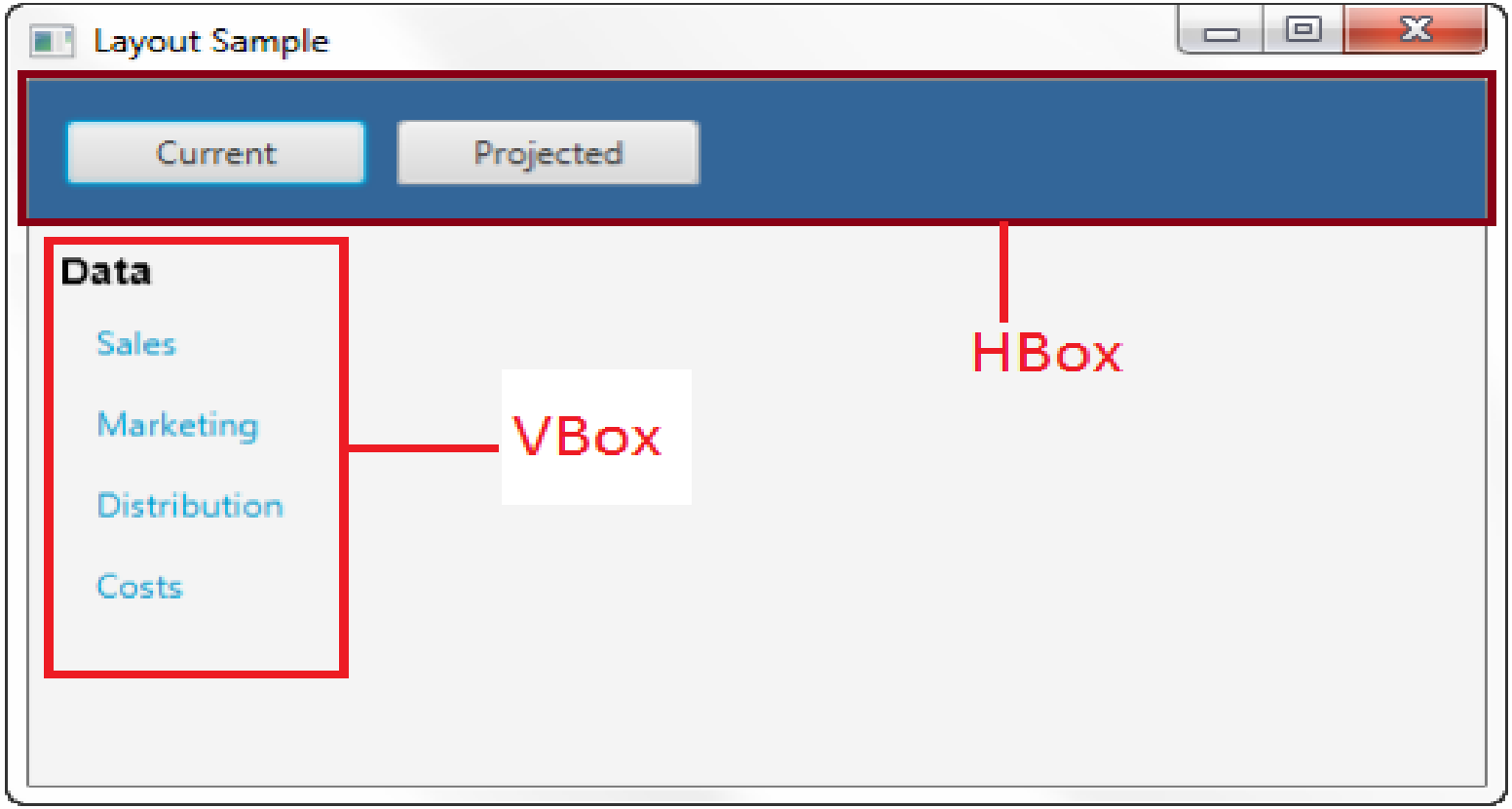
VBox

- The Vbox layout pane is similar to the Hbox layout pane, except that the nodes are arranged in a single column (vertically)
- The padding property can be set to manage the distance between the nodes and the edges of the Vbox pane.
- Spacing can be set to manage the distance between the nodes.
- Margins can be set to add additional space around individual controls.

Example:

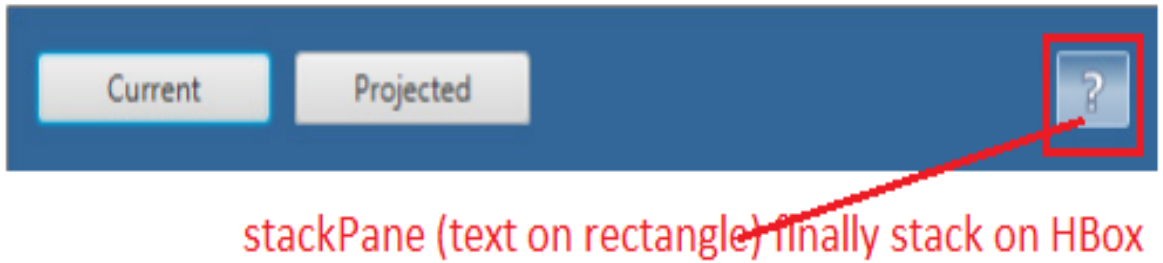
```
VBox vbox = new VBox(); // creating VBox

Text txt1= new Text("Text 1");

Text txt2= new Text("Text 2");

vbox.setMargin(txt1, new Insets(10, 20, 10, 20)); // Adds margin around txt1

vbox.getChildren().addAll(txt1, txt2); //adding text on the VBox
```

Cont..


Layout Sample
Current
Projected
Data
Sales
Marketing
Distribution
Costs
HBox
VBox

StackPane

- The StackPane layout pane places all of the nodes within a single stack with each new node added on top of the previous node.
- This layout model provides an easy way to overlay text on a shape or image or to overlap common shapes to create a complex shape.




```
StackPane stackPane = new StackPane();
Rectangle background = new Rectangle(100, 100, Color.LIGHTBLUE);
 Label text = new Label("Stacked Label");
stackPane.getChildren().addAll(background, text);
```

GridPane

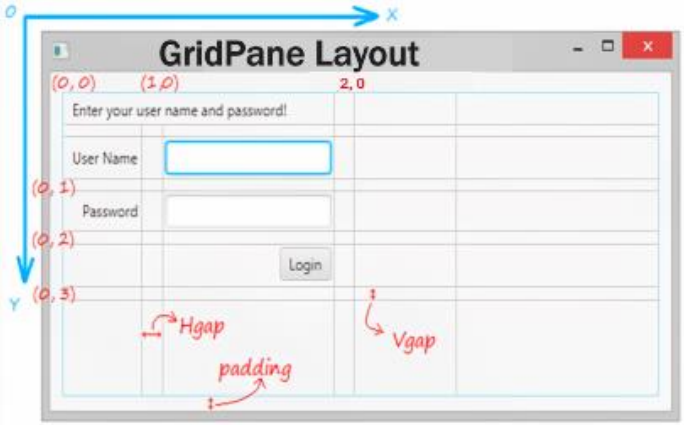
- The GridPane layout pane enables you to create a flexible grid of rows and columns in which to layout nodes.
- Nodes can be placed in any cell in the grid and can span cells as needed.
- A grid pane is useful for creating forms or any layout that is organized in rows and columns.
- **Gap** properties can be set to manage the spacing between the rows and columns.
- The **padding** property can be set to manage the distance between the nodes and the edges of the grid pane.
- The vertical and horizontal alignment properties can be set to manage the alignment of individual controls in a cell.

Properties

The properties of the class along with their setter methods are given in the table below.

Property	Description	Setter Methods
alignment	Represents the alignment of the grid within the GridPane.	setAlignment(Pos value)
gridLinesVisible	This property is intended for debugging. Lines can be displayed to show the gidpane's rows and columns by setting this property to true.	setGridLinesVisible(Boolean value)
hgap	Horizontal gaps among the columns	setHgap(Double value)
vgap	Vertical gaps among the rows	setVgap(Double value)

```
public class GridPaneDemo extends Application {
    @Override
   public void start(Stage primaryStage) throws Exception {
      GridPane root = new GridPane();
       Insets padding = new Insets(10, 15, 20, 25); //Top: 10, Right: 15, Bottom: 20, Left: 25
       root.setPadding(padding);
      root.setHgap(25);       root.setVgap(15);
      Label labelTitle = new Label("Enter your user name and password!");
      // Put on cell (0,0), span 2 column, 1 row.
      root.add(labelTitle, 0, 0, 2, 1);
      Label labelUserName = new Label("User Name");
      TextField fieldUserName = new TextField();
       Label labelPassword = new Label("Password");
     PasswordField fieldPassword = new PasswordField()
       Button loginButton = new Button("Login");
       GridPane.setHalignment(labelUserName, HPos.RIGHT);
     primaryStage.show();   }
       public static void main(String[] args) {launch(args);   }
```



Cont..

```
root.add(labelUserName, 0, 1);
GridPane.setHalignment(labelPassword, HPos.RIGHT);
  root.add(labelPassword, 0, 2);
   // Horizontal alignment for User Name field.
  GridPane.setHalignment(fieldUserName, HPos.LEFT);
  root.add(fieldUserName, 1, 1);
  // Horizontal alignment for Password field.
  GridPane.setHalignment(fieldPassword, HPos.LEFT);
  root.add(fieldPassword, 1, 2);
  // Horizontal alignment for Login button.
  GridPane.setHalignment(loginButton, HPos.RIGHT);
  root.add(loginButton, 1, 3);
   Scene scene = new Scene(root, 300, 300);
  primaryStage.setTitle("GridPanel Layout Demo");
  primaryStage.setScene(scene);
```

FlowPane

- 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).
- A vertical flow pane wraps at the height boundary for the pane. A horizontal flow pane wraps at the width boundary for the pane.
- Gap properties can be set to manage the spacing between the rows and columns.
- The padding property can be set to manage the distance between the nodes and the edges of the pane.
- Creating FlowPane flowpane= new FlowPane();

TilePane

- A tile pane is similar to a flow pane.
- The **TilePane** layout pane places all of the nodes in a grid in which each cell, or tile, is the same size
- Nodes can be laid out horizontally (in rows) or vertically (in columns).
- Horizontal tiling wraps the tiles at the tile pane's width boundary and vertical tiling wraps them at the height boundary.
- Use the **prefColumns** and **prefRows** properties to establish the preferred size
- does not support spanning rows or columns.
- **Gap** properties can be set to manage the spacing between the rows and columns.
- The padding property can be set to manage the distance between the nodes and the edges of the pane.

Example

```
TilePane tilePane = new TilePane();
tilePane.setPrefColumns(3);  // Number of columns
tilePane.setHgap(5);
tilePane.setVgap(5);
 tilePane.setAlignment(Pos.CENTER)
tilePane.getChildren().addAll( new Button("Tile 1"), new Button("Tile 2"), new Button("Tile 3"));
```

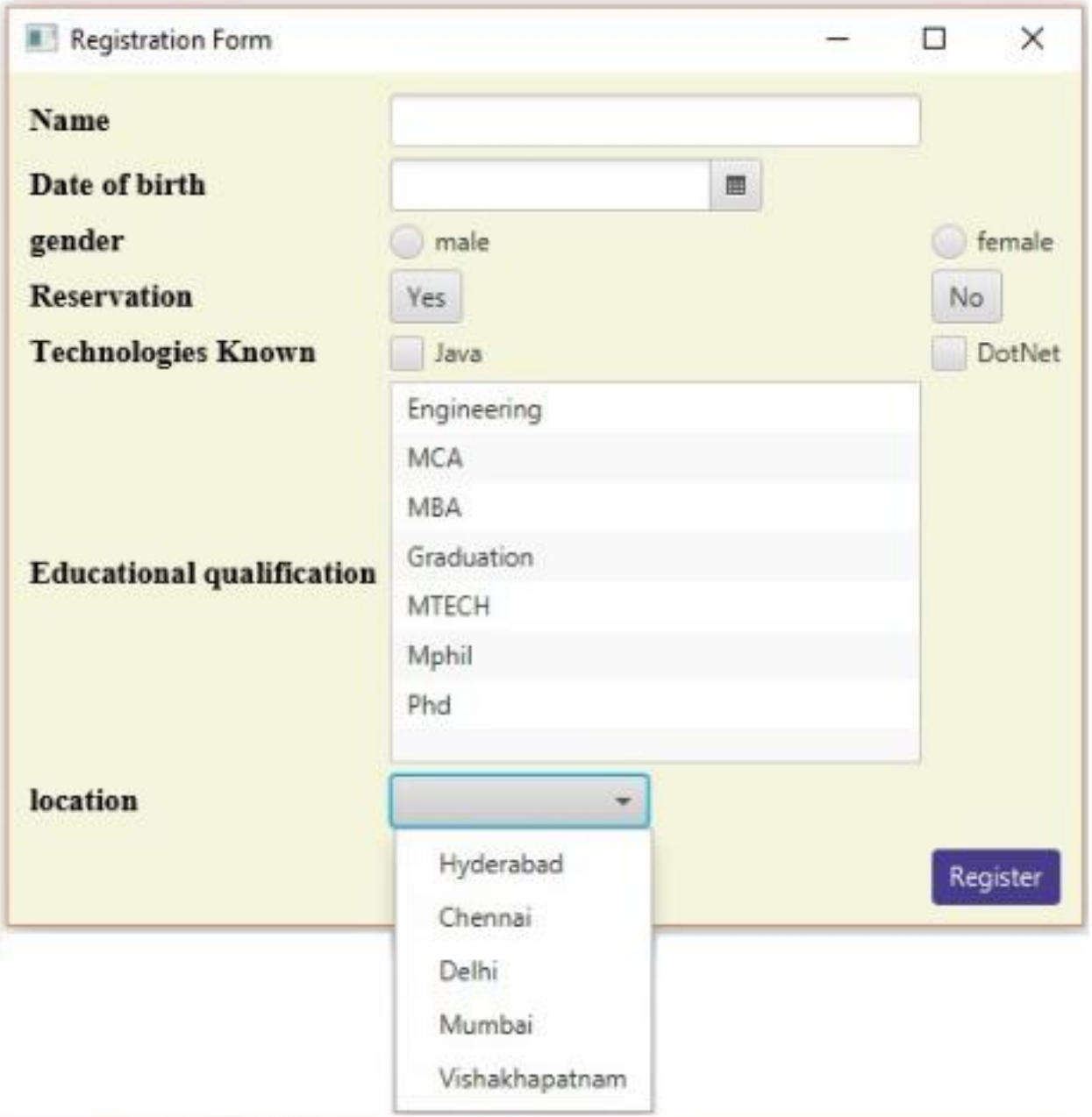
- setOrientation(Orientation): Sets the orientation of the tiles (Orientation.HORIZONTAL or Orientation.VERTICAL) to control how tiles are added to the layout

Javafx UI Controls

- Javafx UI controls are classes those resides in the javafx.scene.control package of the JavaFX API.
- **Common javafx UI Controls**

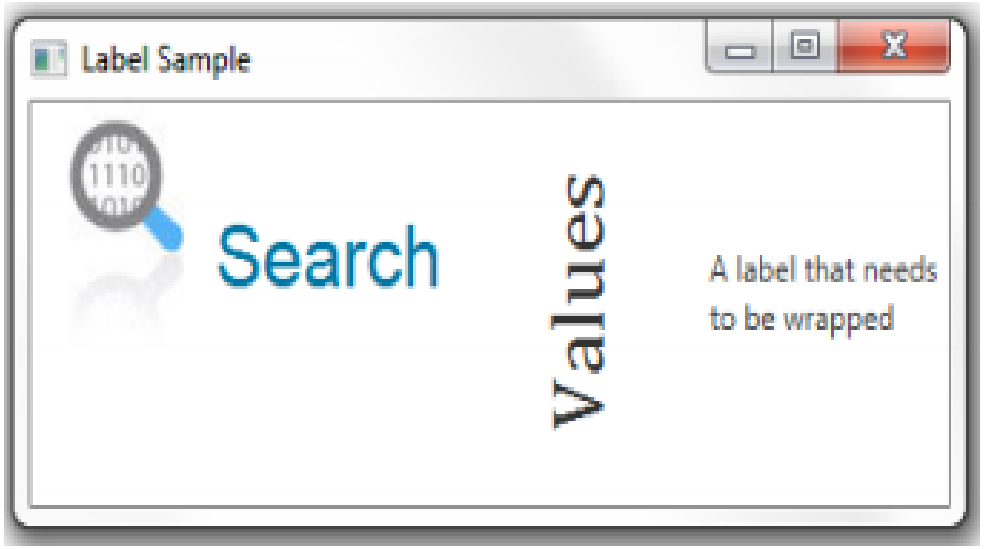
Label
Button
Radio Button
Checkbox
Choice Box
Combo Box
Text Field
Menu
Password Field

List View
Table View
Tree View
Tree Table View
Separator
Slider
Color Picker
Date Picker
Tooltip


Registration Form
Name
Date of birth
gender
male
female
Reservation
Yes
No
Technologies Known
Java
DotNet
Engineering
MCA
MBA
Graduation
MTECH
Mphil
Phd
Educational qualification
location
Hyderabad
Chennai
Delhi
Mumbai
Vishakhapatnam
Register

Label

- Label class resides in the javafx.scene.control package of the JavaFX API to display a text element.
- The below figure shows three common label usages. The label at the left is a text element with an image, the label in the center represents rotated text, and the label at the right renders wrapped text



Cont..

Creating Label

```
//An empty label

Label label1 = new Label();

//A label with the text element

Label label2 = new Label("Search");

//A label with the text element and graphical icon

Image image = new Image( getClass().getResourceAsStream("labels.jpg"));

Label label3 = new Label("Search", new ImageView(image));
```

Cont..

- You can add textual and graphical content by using the following methods of the Labeled class.
  - setText(String text) method – specifies the text caption for the label
  - setGraphic(Node graphic)– specifies the graphical icon
  - setTextFill() method specifies the color to paint the text element of the label.

Cont..

Adding an Icon and Text Fill to a Label

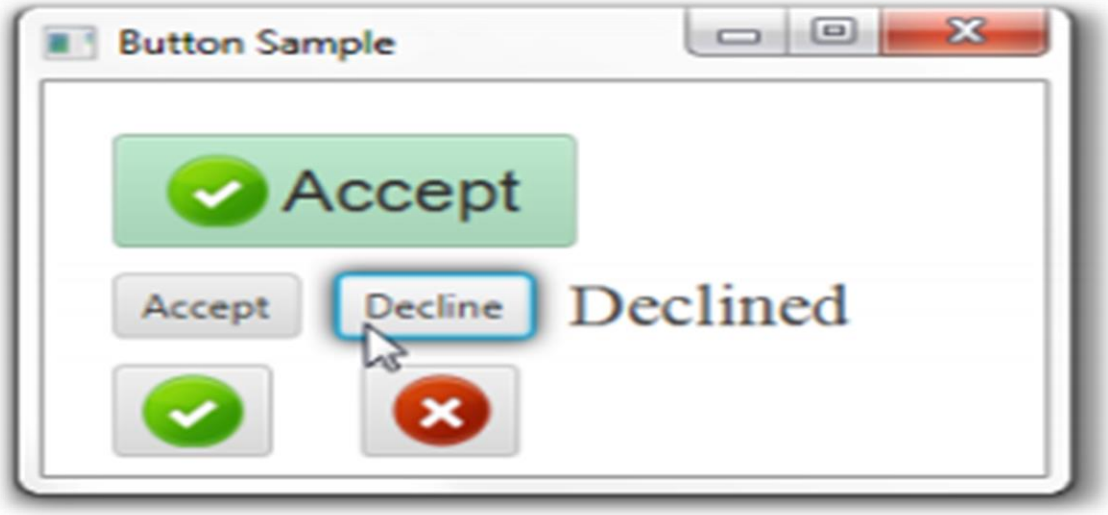
```
Label label1 = new Label("Search");

Image image = new Image (getClass().getResourceAsStream("labels.jpg"));

label1.setGraphic(new ImageView(image));

label1.setTextFill(Color.web("#0076a3"));
```

Applying Font Settings to a label

```
//Use a constructor of the Font class

label1.setFont(new Font("Arial", 30));

//Use the font method of the Font class

label2.setFont(new Font.("Cambria", 32));
```

Button

- Button class resides in the javafx.scene.control package of the JavaFX API enables developers to process an action when a user clicks it.
- is an extension of the Labeled class.
- It can display text, an image, or both.



Cont..

Creating Button

```
//A button with an empty text caption.
Button button1 = new Button();
//A button with the specified text caption.
Button button2 = new Button("Accept");
//Button with tooltip
Tooltip tooltip1 = new Tooltip("Creates a new tootip");
Button button1 = new Button("New");
button1.setTooltip(tooltip1);
//A button with the specified text caption and icon.
Image imageOk = new Image(getClass().getResourceAsStream("ok.png"));
Button button3 = new Button("Accept", new ImageView(imageOk));
```

Cont..

Because the Button class extends the Labeled class, you can use the following methods to specify content for a button that does not have an icon or text caption:

- setText(String text) method – specifies the text caption for the button
- setGraphic(Node graphic) method – specifies the graphical icon

Adding an Icon to a Button

```
Image imageDecline = new Image
(getClass().getResourceAsStream("not.png"));
Button button5 = new Button();
button5.setGraphic(new ImageView(imageDecline));
```

Cont..

The default skin of the Button class distinguishes the visual states of the button.

RadioButton

- ✓ RadioButton class resides in the javafx.scene.control package of the JavaFX API.
- ✓ A radio button control can be either selected or deselected.
- ✓ Radio buttons are combined into a group(ToggleGroup) where only one button at a time can be selected.

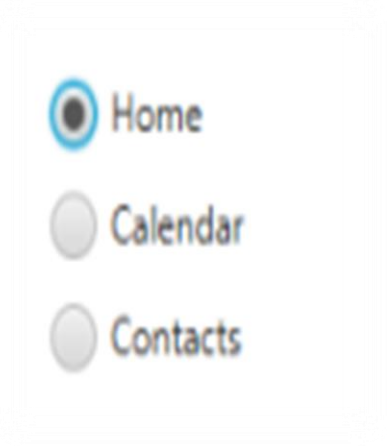
Cont..

Creating Radio Buttons

```
//A radio button with an empty string for its label
RadioButton rb1 = new RadioButton();
//Setting a text label
rb1.setText("Home");
//A radio button with the specified label
RadioButton rb2 = new RadioButton("Calendar");
```

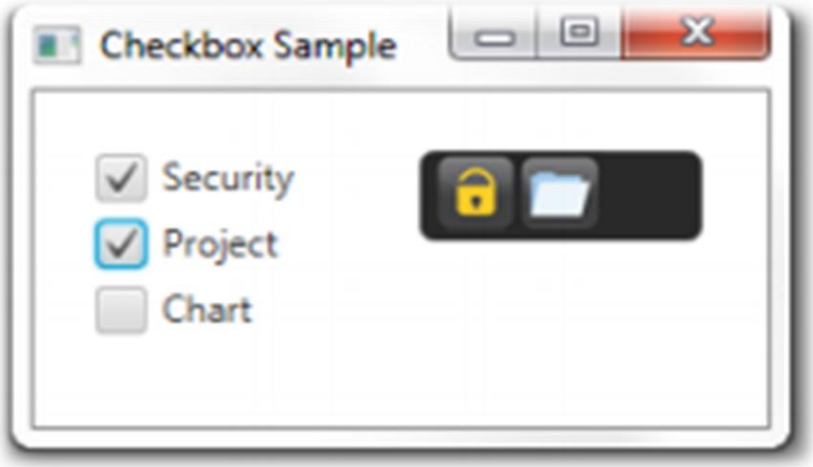
Cont..

```
final ToggleGroup group = new ToggleGroup();
RadioButton rb1 = new RadioButton("Home");
rb1.setToggleGroup(group);
rb1.setSelected(true);
RadioButton rb2 = new RadioButton("Calendar");
rb2.setToggleGroup(group);
RadioButton rb3 = new RadioButton("Contacts");
rb3.setToggleGroup(group);
```



CheckBox

- ✓ CheckBox class resides in the javafx.scene.control package of the JavaFX API.
- ✓ CheckBox control allow the user to select many options at a time.



Cont..

Creating Checkboxes

```
//A checkbox without a caption
CheckBox cb1 = new CheckBox();
//A checkbox with a string caption
CheckBox cb2 = new CheckBox("Second");
cb1.setText("First");
// set the checked box to be selected
cb1.setSelected(true);
```

ChoiceBox

- ✓ Choice Box class resides in the javafx.scene.control package of the JavaFX API.
- ✓ Choice Box control provides support for quickly selecting between a few options.
- ✓ Can able to select exactly one option/choice

Creating a Choice Box

- choiceBox.getValue() method is used to take selected choice value

```
ChoiceBox cb = new ChoiceBox(FXCollections.observableArrayList ( "First",
"Second",  "Third") );
```

TextField

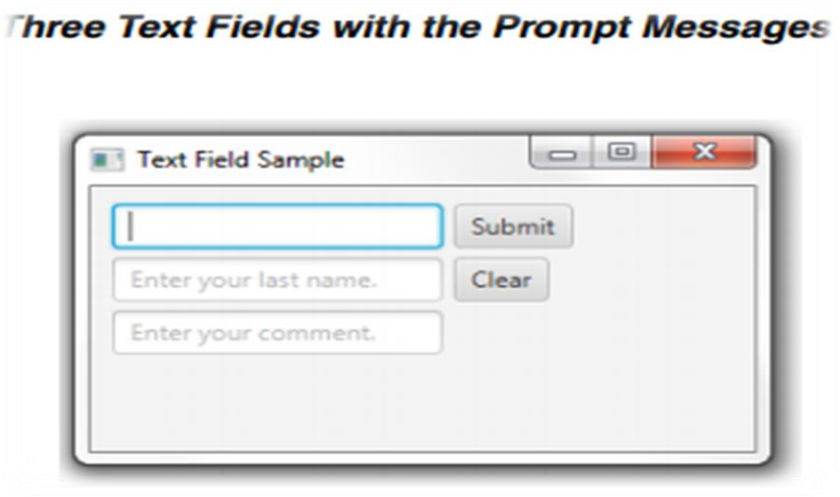
- ✓ The TextField class implements a UI control that accepts and displays text input.
- ✓ It provides capabilities to receive text input from a user. Along with another text input control, PasswordField, this class extends the TextInput class, a super class for all the text controls available through the JavaFX API.

- ✓ **<u>Creating TextField</u>**

- TextField tf = new TextField ();

Cont.

- To defines the string that appears in the text field when the application is started TextField use setPromptText() method.
- Prompt captions notify users what type of data to enter in the text fields.



- The difference between the prompt text and the text entered in the text field is that the prompt text cannot be obtained through the getText() method

Cont.

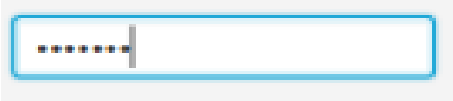
Some helpful methods that you can use with text fields.

- ✓ copy()– transfers the currently selected range in the text to the clipboard, leaving the current selection.
- ✓ cut()– transfers the currently selected range in the text to the clipboard, removing the current selection.
- ✓ selectAll()- selects all text in the text input.
- ✓ paste()– transfers the contents in the clipboard into this text, replacing the current selection.

PasswordField

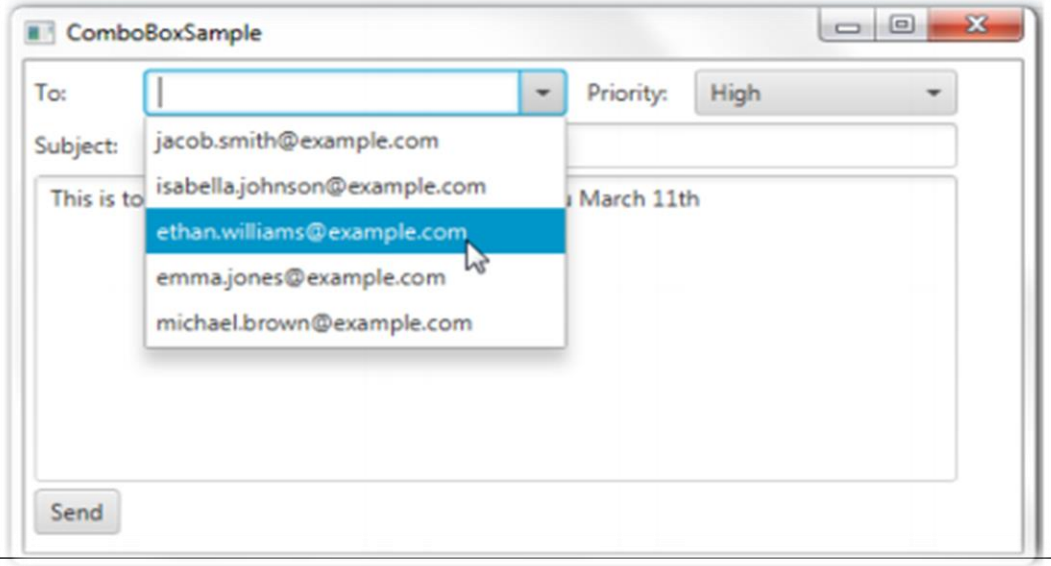
The PasswordField class implements a specialized textfield. The characters typed by a user are hidden by displaying an echo string.

```
//Creating a Password Field
PasswordField passwordField = new PasswordField();
passwordField.setPromptText("Your password");
```



ComboBox

- A combo box is a typical element of a user interface that enables users to choose one of several options.
- A combo box is helpful when the number of items to show exceeds some limit, because it can add scrolling to the drop down list, unlike a choice box.
- If the number of items does not exceed a certain limit, developers can decide whether a combo box or a choice box better suits their needs.



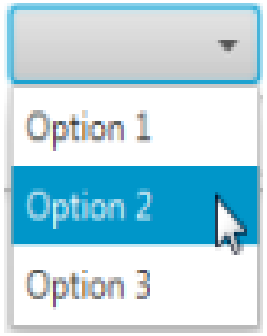
ComboBox

<u>Creating a Combo Box with an Observable List</u>

•ObservableList<String> options = FXCollections.observableArrayList(

"Option 1","Option 2", "Option 3");

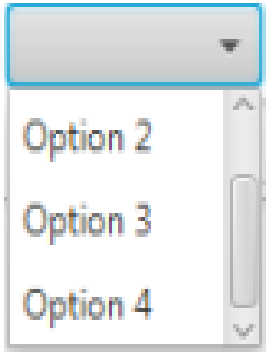
ComboBox comboBox = new ComboBox(options);



- At any time, you can supplement the list of items with new values.

comboBox.getItems().addAll("Option 4", "Option 5","Option 6");

- comboBox.getValue() used to take selected value

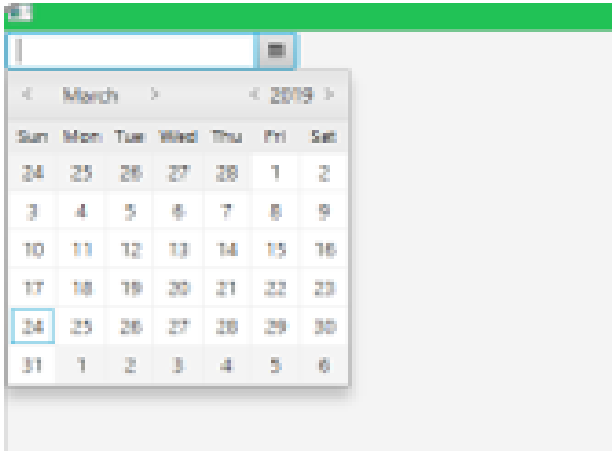


DatePicker

- Enables selection of a day from the given calendar
- Creating date Picker

DatePicker datepicker= new DatePicker();

- datepicker.getValue() to take the selected date



ListView

- A ListView is a component that allows users to display and interact with a scrollable list of items.
- It is highly versatile and supports both single and **multiple** selection modes.
- Can display any type of data. By default, it uses toString() to render items.
- Selection Modes
  - **SelectionMode**.SINGLE: Allows selection of a single item (default).
  - **SelectionMode**.MULTIPLE: Allows selection of multiple items.
- i.e ListView<String> listView = new ListView<>(items);

  items.add("JAva"); items.add("OS");

ScrollPane

- A layout container that provides a scrollable view of its content.
- It is useful when the content exceeds the visible area of the application window.
- Adds horizontal and/or vertical scrollbars when the content exceeds its viewport.
- i.e TextArea textArea = new TextArea();
  textArea.setPrefSize(300, 200); // Set preferred size
  textArea.setWrapText(true); // Enable text wrapping //
  ScrollPane scrollPane = new ScrollPane();
  scrollPane.setContent(textArea);
  scrollPane.setFitToWidth(true);

  scrollPane.setPannable(true); //
- Panning allows the user to drag the content of the ScrollPane using a mouse or touch gestures, rather than scrolling only with the scrollbars or a mouse wheel.

TabPane

- The TabPane is a container that organizes its content into tabs.
- Each tab can have its own content and can be navigated by selecting the corresponding tab.
- Tabs can contain various types of content (e.g., layouts, controls, or custom nodes).
- Tabs can be made closable, allowing users to remove them.

```
TabPane tabPane = new TabPane();
  Tab tab1 = new Tab("Home");
  tab1.setContent(new Label("Welcome to the Home tab!"));
  Tab tab2 = new Tab("Settings");
  tab2.setContent(new Label("Settings go here."));
   tabPane.getTabs().addAll(tab1, tab2);
 StackPane root = new StackPane(tabPane);
```

Menu

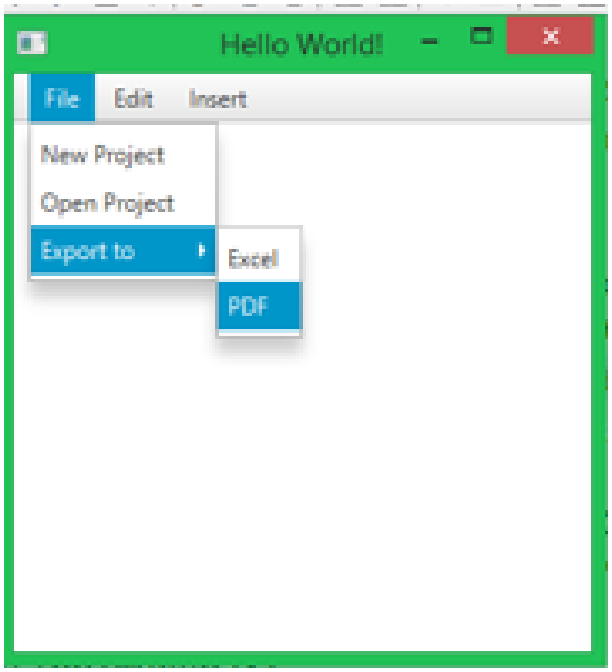
- Menus are a standard way for desktop application to select options

- **Creating menus and menu items**

- We must create a Menubar object to hold Menu object

- MenuBar menubar= new MenuBar();

- Menu object can hold menu and menuItem object



Creating menu

```
//menu
    Menu file= new Menu("File");
    Menu edit= new Menu("Edit");
    Menu insert= new Menu("Insert");
    menubar.getMenus().addAll(file, edit,insert); // add  menus on menu bar
    // submenus
    MenuItem newproject= new MenuItem("New Project");
    MenuItem open= new MenuItem("Open Project");
    Menu export= new Menu("Export to");
file.getItems().addAll(newproject, open, export);     //add submenus to menu
• // add menuItems to menu
    export.getItems().addAll( new MenuItem("Excel"), new MenuItem("PDF"));
   root.setTop(menubar); // add the menubar on the container
```

JavaFX 2D Shapes

- JavaFX provides the flexibility to create our own 2D shapes on the screen
- resides in **javafx.scene.shape** package.
- A two dimensional shape is geometrical figure that can be drawn on the coordinate system consist of X and Y planes
- 2D shapes such as Line, Rectangle, Circle, Ellipse, Polygon, Cubic Curve, quad curve, Arc, et
- Example 1, Rectangele

```
Rectangle rect = new Rectangle()
  rect.setX(20); //setting the X coordinate of upper left //corner of rectangle
  rect.setY(20); //setting the Y coordinate of upper left //corner of rectangle
  rect.setWidth(100); //setting the width of rectangle
  rect.setHeight(100); // setting the height of rectangle
```

Example 2

```
Circle circle = new Circle();
circle.setCenterX(200);
circle.setCenterY(200);
circle.setRadius(100);
circle.setFill(Color.RED);
```

Example 3 :

```
Line line = new Line(); //instantiating Line class
line.setStartX(0); //setting starting X point of Line
line.setStartY(0); //setting startingY point of Line
line.setEndX(100); //setting ending X point of Line
line.setEndY(200); //setting endingY point of Line
```

Group Assignment

1) **JavaFX Charts group -6**
2) **Media with JavaFX 8**
3) **JavaFX 3D Shapes 2**
4) **JavaFX 3D Shapes 3**
5) **JavaFX Effects 1**
6) **JavaFX Charts 5**
7) **JavaFX Transformation 7**
8) **JavaFX Animation 4**

Event Handling

- ✓ An event is an occurrence of a user interaction with the application.
- ✓ Clicking the mouse and pressing a key on the keyboard are examples of events
- ✓ An event in JavaFX is represented by an object of the **javafx.event.Event** class or any of its subclasses.
- ✓ Every event in JavaFX has three properties:
  - **An event source:-**The source from which the event is generated will be the source of the event. i.e. mouse is the source of the even
  - **An event target:** The node on which an event occurred. A target can be a window, scene, and a node.
  - **An event type: Type** of the occurred event; in case of mouse event – mouse **pressed**, **mouse** released are the type of events.

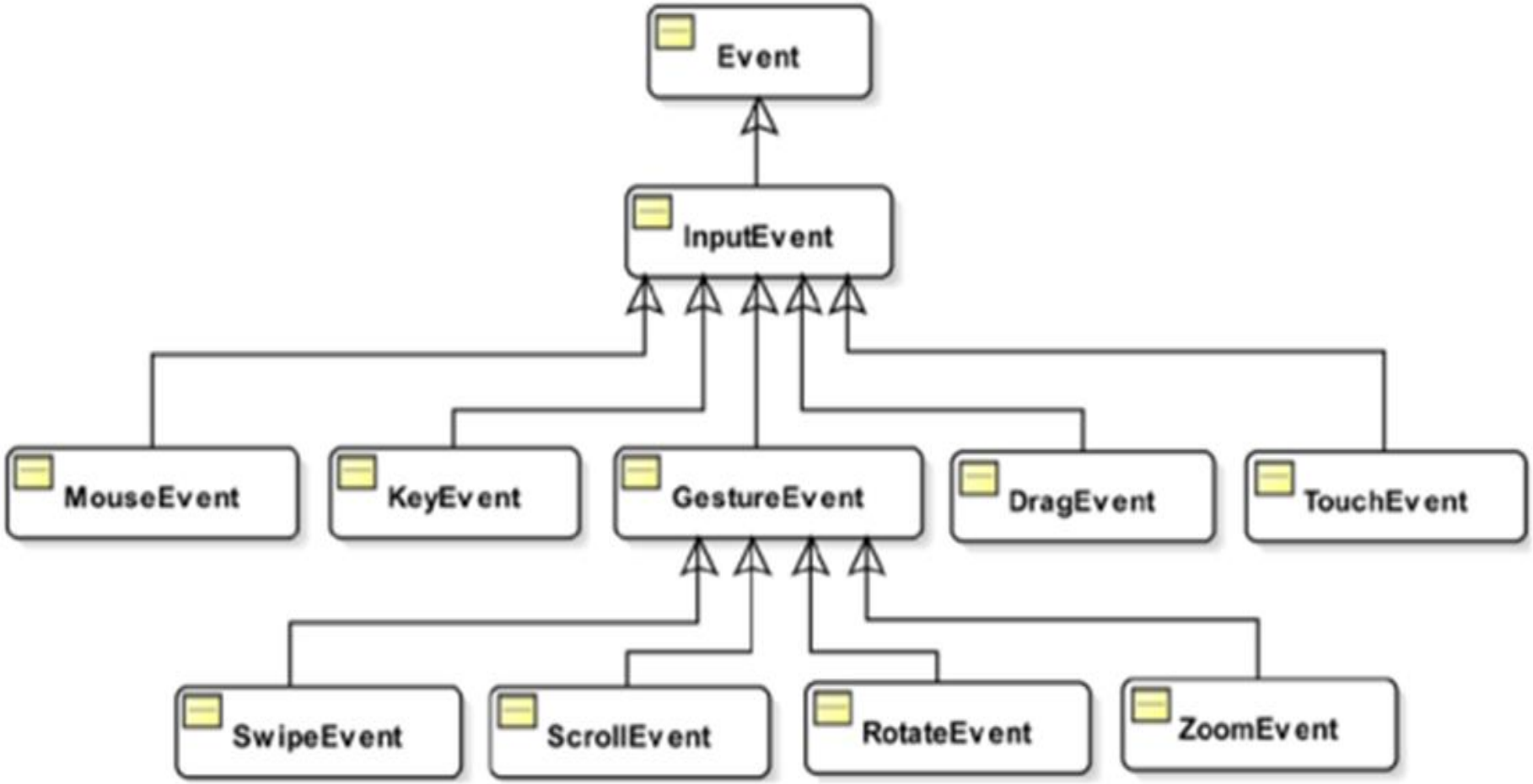
Contt..

- The piece of code that is executed in response to an event is known as an **event handler** or an **event filter**.
- The UI element that calls event handlers is the **source** of the event for those event handlers. When an event occurs, it passes through a chain of event dispatchers.
- The **event target** is the destination of an event. The event target determines the route through which the event travels during its processing. Suppose a mouse click occurs over a Circle node. In this case, the Circle node is the event target of the mouse-clicked event.

Contt..

- The event type describes the type of the event that occurs. for example, **KeyEvent, MouseEvent, DragEvent, WindowEvent**
- An input event indicates a user input (or a user action), for example, clicking the mouse, pressing a key, touching a touch screen, and so forth. JavaFX supports many types of input events.
- All input event–related classes are in the javafx.scene.input package.
- Below Figure shows the class diagram for some of the classes that represent input event.

Contt..


Event
InputEvent
MouseEvent
KeyEvent
GestureEvent
DragEvent
TouchEvent
SwipeEvent
ScrollEvent
RotateEvent
ZoomEvent

Register event handler

- To add an event handler to a node, use the method **addEventHandler()** of the Node class

```
EventHandler<MouseEvent> eventHandler =
  new EventHandler<MouseEvent>() {
    @Override
  public void handle(MouseEvent e) {
    System.out.println("Hello World");
    circle.setFill(Color.DARKSLATEBLUE);
  }
};
//Adding the event handler
btn.addEventHandler(MouseEvent.MOUSE_CLICKED, eventHandler);
```

- To remove an event use removeEventHandler();
  - btn.removeEventHandler(MouseEvent.MOUSE_CLICKED, eventHandler);

Register event Filter

- Use addEventFilter() of the Node class

//Creating the mouse event handler

Example:
```
EventHandler<MouseEvent> eventHandler = new EventHandler<MouseEvent>() {
  @Override
  public void handle(MouseEvent e) {
    System.out.println("Hello World");
    circle.setFill(Color.DARKSLATEBLUE);
  }
};
btn.addEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);
```

- Mouse event types can be MOUSE_CLICKED, MOUSE_ENTERED, MOUSE_PRESSED etc.

Styling nodes

- ✓ styling a node means decorating or make different from the default appearance, this my done using javaFx or CSS codes.
- ✓ A cascading style sheet (CSS) is a language used to describe the presentation of UI elements in a GUI application.
- ✓ JavaFX allows you to define the look (or the style) of JavaFX applications using CSS.
- ✓ To embed the CSS styling code in to the **javaFx** code have two methods.
  - Those are **Inline styling** and **External styling**

cont..

- using **external** styling you can add multiple style sheets to a JavaFX application.
- Style sheets are added to a scene or parents. Like
  - Scene scene = ...

```
scene.getStylesheets().add(javaFileName.class.getResource(“ss1.c
ss").toExternalForm());
```

//Add a style sheet, vbox.css, to a VBox (a Parent)

- VBox root = new VBox();

```
root.getStylesheets().add("vbox.css");
```

- ss1.css and vbox.css are css files that containe stayle of the node of javaFx application.

cont..

- .button {

```
-fx-background-color: red;
-fx-text-fill: white;
    }
```

- ✓ The Node class has a style property that is of String Property type.
- ✓ The style property holds the inline style for a node.
- ✓ can use the setStyle(String inlineStyle) and getStyle() methods to set and get the inline style of a node

JavaFX CSS properties

-fx-background-color

-fx-background-image

-fx-background-radius

-fx-background-repeat

-fx-font-size

-fx-background-radius: 50px;

-fx-font-weight

-fx-max-height

-fx-max-width

-fx-text-fill

-fx-border-color

-fx-padding

Cont..

Styling a Button

```
//Code added to the CSS file

.button1{

-fx-font: 22 arial;

-fx-base: #b6e7c9;

} //Code in the ButtonSample.java file

button1.getStyleClass().add("button1");
```

Example 1: File name **mystyle.css**

```
.button {
    -fx-text-fill: white;
    -fx-font-family: "Arial";
    -fx-font-size: 14px;
    -fx-padding: 10px 20px;
    -fx-border-color: #0056b3;
    -fx-border-width: 2px;
    -fx-border-radius: 5px;
    -fx-effect: dropshadow(three-pass-box, #000000, 10, 0.5, 0, 0);
    -fx-opacity: 0.9;
    -fx-cursor: hand;
}
```

```
.label {    -fx-font-size: 16px;}
.container
{    -fx-padding: 10px }
#myButton {    -fx-background-color: #007bff;}
#mytextField{
-fx-text-fill: red;
    -fx-min-height: 100;
    -fx-font-size: 50;
}
#lbl{
    -fx-text-fill: green;
    -fx-font-size: 50;
}
```

Example 1: File name **Main.java**

```
public class Main extends Application {
   public static void main(String[] args)
{
      launch(args);
   }
public void start(Stage stage)  {
      VBox vb= new VBox();
      Label name= new Label("First Name");
      name.setId("lbl");
      TextField tfName= new TextField();
      Button button = new Button("Click Me");
      button.setId("myButton");
      tfName.setId("mytextField");
 tfName.setStyle("-fx-background-color:yellow; -fx-border-color: blue");
vb.getChildren().addAll(name, tfName, button);
 Scene sc= new Scene(vb, 300, 300);
 sc.getStylesheets().add(getClass().getResource("mystyle.css").toExternalForm());
vb.getStyleClass().add("container");
vb.setSpacing(20);
stage.setScene(sc);
stage.setTitle("First example");
stage.show();
    }
}
```

Thanks ??

The End