

JAVA PROGRAMMING

Chap 8 : GUI Design using
Java FX

Java FX

- ▶ JavaFX is a Java library and a GUI toolkit designed to develop and facilitate Rich Internet applications, web applications, and desktop applications.
- ▶ Rich Internet Applications are those web applications that allow alike characteristics and expertise as that of desktop applications. These applications contribute more satisfying visual experience to the users when compared to the standard web applications.
- ▶ The most significant perk of using JavaFX is that the applications written using this library can run on multiple operating systems like Windows, Linux, iOS, Android, and several platforms like Desktops, Web, Mobile Phones, TVs, Tablets, etc.
- ▶ This characteristic of the JavaFX library makes it very versatile across operating systems and different platforms.
- ▶ After the arrival of JavaFX, Java developers and programmers were able to develop the GUI applications more effectively and more productively.
- ▶ JavaFX was introduced to supersede the Java Swing GUI framework.
- ▶ Nevertheless, the JavaFX provides more enhanced functionalities and features than the Java Swing.

Steps to configure Java FX on Netbeans & Eclipse :

https://www.tutorialspoint.com/javafx/javafx_environment.htm

Features of Java FX

Java Library – JavaFX is a Java library, which allows the user to gain the support of all the Java characteristics such as multithreading, generics, lambda expressions, and many more. The user can also use any of the Java editors or IDE's of their choice, such as Eclipse, NetBeans, to write, compile, run, debug, and package their JavaFX application.

Platform Independent – The rich internet applications made using JavaFX are platform-independent. The JavaFX library is open for all the scripting languages that can be administered on a JVM, which comprise – Java, Groovy, Scala, and JRuby.

FXML – JavaFX emphasizes an HTML-like declarative markup language known as FXML. FXML is based on extensible markup language (XML). The sole objective of this markup language is to specify a user interface (UI). In FXML, the programming can be done to accommodate the user with an improved GUI.

Scene Builder – JavaFX also implements a tool named Scene Builder, which is a visual editor for FXML. Scene Builder generates FXML mark-ups that can be transmitted to the IDE's like Eclipse and NetBeans, which further helps the user to combine the business logic to their applications. The users can also use the drag and drop design interface, which is used to design FXML applications.

Hardware-accelerated Graphics Pipeline – The graphics of the JavaFX applications are based on the hardware-accelerated graphics rendering pipeline, commonly known as Prism. The Prism engine offers smooth JavaFX graphics that can be rendered quickly when utilized with a supported graphics card or graphics processing unit (GPU). In the case where the system does not hold the graphic cards, then the prism engine defaults to the software rendering stack.

Swing Interoperability – In a JavaFX application, you can embed Swing content using the Swing Node class. Similarly, you can also update the existing Swing applications with JavaFX features.

Features of Java FX

WebView – JavaFX applications can also insert web pages. To embed web pages, Web View of JavaFX uses a new HTML rendering engine technology known as WebKitHTML. WebView is used to make it possible to insert web pages within a JavaFX application. JavaScript running in WebView can call Java APIs and vice-versa.

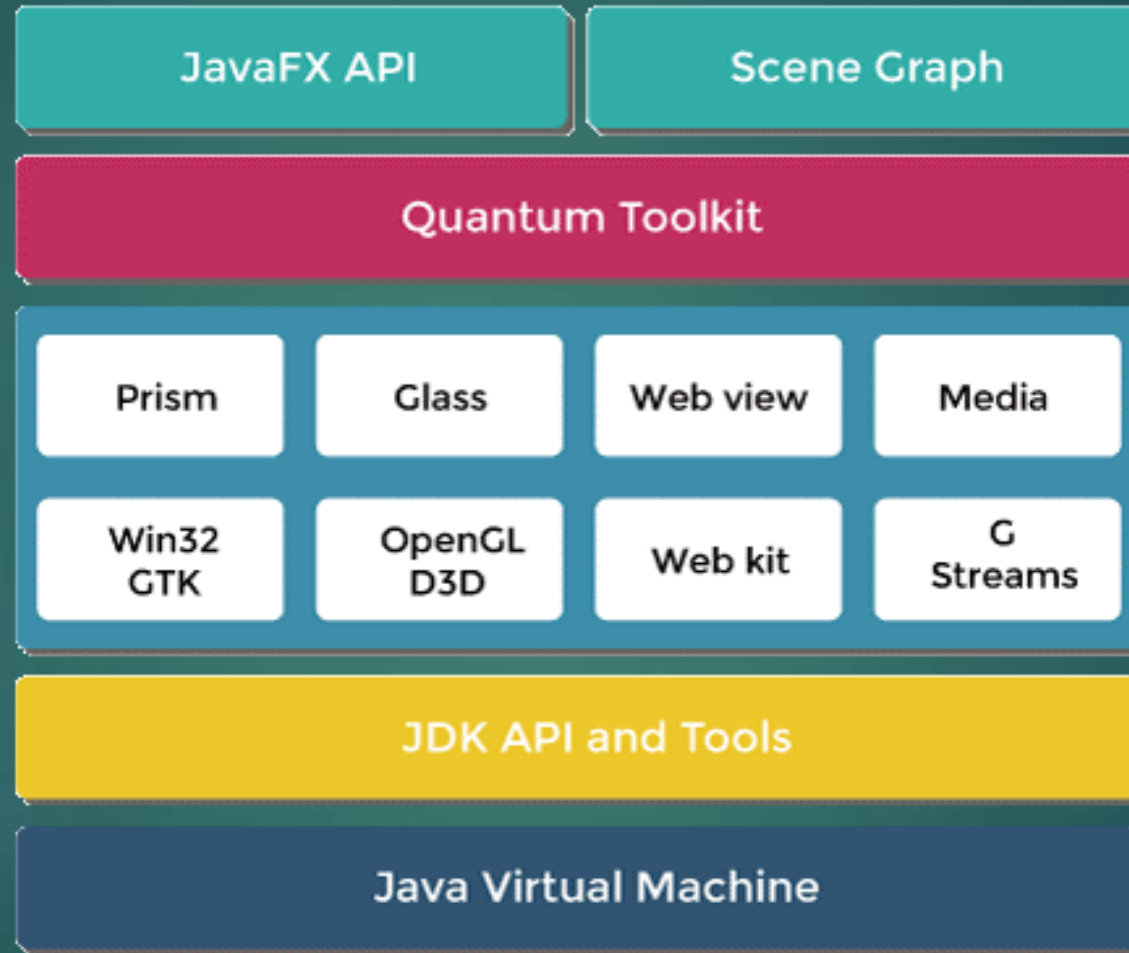
Built-in UI controls – JavaFX comprises all the major built-in UI controls that help in developing well-featured applications. These built-in UI components are not operating system-dependent. In simple words, these controls do not depend on any of the Operating systems like Windows, iOS, Android, etc. These built-in controls are single-handedly ample to perform a whole implementation of the applications.

CSS Styling – Just like websites use CSS for styling, JavaFX also provides the feature to integrate the application with CSS styling. The users can enhance the styling of their applications and can also improve the outlook of their implementation by having simple knowledge and understanding of CSS styling.

Rich set of APIs – JavaFX library also presents a valuable collection of APIs that helps in developing GUI applications, 2D and 3D graphics, and many more. This collection of APIs also includes all the characteristics of the Java platform. Hence, working with this API, a user can access the specialties of Java languages such as Generics, Annotations, Multithreading, and Lambda Expressions, and many other features as well. In JavaFX, the popular Java Collections library was also improved, and notions like lists and maps were introduced. Using these APIs, the users can witness the variations in the data models.

High-Performance media engine – Like the graphics pipeline, JavaFX also possesses a media pipeline that advances stable internet multimedia playback at low latency. This high-performance media engine or media pipeline is based on a multimedia framework known as Gstreamer.

Architecture of Java FX



Architecture of Java FX

- ▶ **JavaFX API** – The topmost layer of JavaFX architecture holds a JavaFX public API that implements all the required classes that are capable of producing a full-featured JavaFX application with rich graphics. The list of all the important packages of this API is as follows.
 - ▶ `javafx.animation`: It includes classes that are used to combine transition-based animations such as fill, fade, rotate, scale and translation, to the JavaFX nodes (collection of nodes makes a scene graph).
 - ▶ `javafx.css` – It comprises classes that are used to append CSS-like styling to the JavaFX GUI applications.
 - ▶ `javafx.geometry` – It contains classes that are used to represent 2D figures and execute methods on them.
 - ▶ `javafx.scene` – This package of JavaFX API implements classes and interfaces to establish the scene graph. In extension, it also renders sub-packages such as canvas, chart, control, effect, image, input, layout, media, paint, shape, text, transform, web, etc. These are the diverse elements that sustain this precious API of JavaFX.
 - ▶ `javafx.application` – This package includes a collection of classes that are responsible for the life cycle of the JavaFX application.
 - ▶ `javafx.event` – It includes classes and interfaces that are used to perform and manage JavaFX events.
 - ▶ `javafx.stage` – This package of JavaFX API accommodates the top-level container classes used for the JavaFX application.
- ▶ **Scene Graph** – A Scene Graph is the starting point of the development of any of the GUI Applications. In JavaFX, all the GUI Applications are made using a Scene Graph only. The Scene Graph includes the primitives of the rich internet applications that are known as nodes. In simple words, **a single component in a scene graph is known as a node**. In general, a scene graph is made up of a collection of nodes. All these nodes are organized in the form of a hierarchical tree that describes all of the visual components of the application's user interface (UI). A node instance can be appended to a scene graph only once.

Architecture of Java FX

- ▶ A node is a visual/graphical object and it may include –
 - ▶ Geometrical (Graphical) objects – (2D and 3D) such as circle, rectangle, polygon, etc.
 - ▶ UI controls – such as Button, Checkbox, Choice box, Text Area, etc.
 - ▶ Containers – (layout panes) such as Border Pane, Grid Pane, Flow Pane, etc.
 - ▶ Media elements – such as audio, video and image objects.
- ▶ The nodes are of three general types.
 - ▶ Root Node – A root node is a node that does not have any node as its parent.
 - ▶ Leaf Node – A leaf node is a node that does not contain any node as its children.
 - ▶ Branch Node – A branch node is a node that contains a node as its parent and also has a node as its children.
- ▶ **Quantum Toolkit** – Quantum Toolkit is used to connect prism and glass windowing tool kits collectively and prepares them for the above layers in the stack. In simple words, it ties Prism and GWT together and makes them available to JavaFX.
- ▶ **Prism** – The graphics of the JavaFX applications are based on the hardware-accelerated graphics rendering pipeline, commonly known as Prism. The Prism engine supports smooth JavaFX graphics that can be executed swiftly when utilized with a backed graphics card or graphics processing unit (GPU). In the situation where the system does not contain the graphic cards, then the prism engine defaults to the software rendering stack. To render graphics, a Prism uses –
 - ▶ DirectX 9 on Windows XP and Vista.
 - ▶ DirectX 11 on Windows 7.
 - ▶ OpenGL on Mac and Linux, Embedded Systems.

Architecture of Java FX

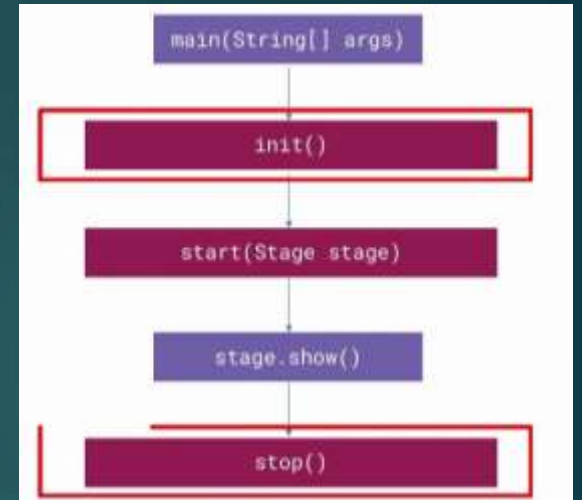
- ▶ **Glass Windowing Toolkit** – Glass Windowing Toolkit or simply Glass is a platform-dependent layer that assists in connecting the JavaFX platform to the primary operating system (OS). Glass Windowing Toolkit is very useful as it provides services such as controlling the windows, events, timers, and surfaces to the native operating system.
- ▶ **WebView** – JavaFX applications can also insert web pages. To embed web pages, Web View of JavaFX uses a new HTML rendering engine technology known as WebKitHTML. WebView is used to make it possible to insert web pages within a JavaFX application. JavaScript appearing in WebView can call Java APIs and vice-versa. This element promotes different web technologies like HTML5, CSS, JavaScript, DOM, and SVG. Using web view, we can execute the HTML content from the JavaFX application and can also implement some CSS styles to the user interface (UI) part of the application. Using WebView, you can –
 - ▶ Render HTML content from local or remote URL.
 - ▶ Support history and provide Back and Forward navigation.
 - ▶ Reload the content.
 - ▶ Apply effects to the web component.
 - ▶ Edit the HTML content.
 - ▶ Execute JavaScript commands.
 - ▶ Handle events.

Media Engine – Like the graphics pipeline, JavaFX also possesses a media pipeline that advances stable internet multimedia playback at low latency. This high-performance media engine or media pipeline is based on a multimedia framework known as Gstreamer. By applying the Media engine, the JavaFX application can support the playback of audio and video media files. The package `javafx.scene.media` covers all the classes and interfaces that can provide media functionalities to JavaFX applications. It is provided in the form of three components, which are – Media Object – This represents a media file, Media Player – To play media content, Media View – To display media.

LifeCycle of Java FX Application

- ▶ We need to import `javafx.application.Application` class in every JavaFX application.
- ▶ This provides the following life cycle methods for JavaFX application.

```
public void init()
public abstract void start(Stage primaryStage)
public void stop()
```
- ▶ **init()** – The `init()` method is an empty method that can be overridden. In this method, the user cannot create a stage or a scene.
- ▶ **start()** – The `start()` method is the entry point method of the JavaFX application where all the graphics code of JavaFX is to be written.
- ▶ **stop()** – The `stop()` method is an empty method that can also be overridden, just like the `init()` method. In this method, the user can write the code to halt the application.
- ▶ Other than these methods, the JavaFX application also implements a static method known as **launch()**. This `launch()` method is used to launch the JavaFX application. As stated earlier, the `launch()` method is static, the user should call it from a static method only. Generally, that static method, which calls the `launch()` method, is the `main()` method only.



LifeCycle of Java FX Application

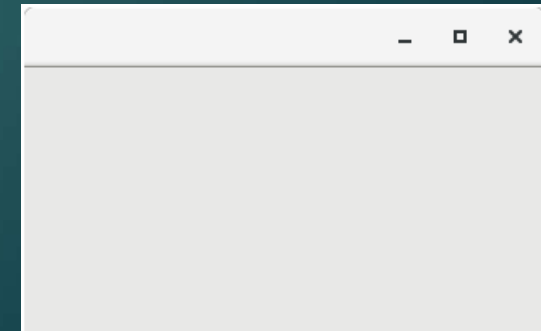
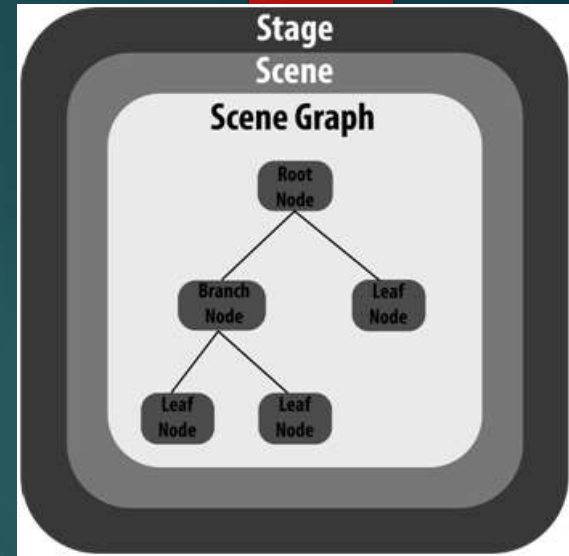
- ▶ Whenever a user launches a JavaFX application, there are few actions that are carried out in a particular manner only.
- ▶ The following is the order given in which a JavaFX application is launched.
 - ▶ Firstly, an instance of the application class is created.
 - ▶ After that, the `init()` method is called.
 - ▶ After the `init()` method, the `start()` method is called.
 - ▶ After calling the `start()` method, the launcher waits for the JavaFX application to end and then calls the `stop()` method.
- ▶ in order to create a basic JavaFX application, we need to:
 - Import `javafx.application.Application` into our code.
 - Inherit `Application` into our class.
 - Override `start()` method of `Application` class.

Application Structure of Java FX

- ▶ JavaFX application is divided hierarchically into three main components known as Stage, Scene and nodes.

Stage

- ▶ Stage in a JavaFX application is similar to the Frame in a Swing Application.
- ▶ It acts like a container for all the JavaFX objects.
- ▶ Primary Stage is created internally by the platform.
- ▶ Other stages can further be created by the application.
- ▶ The object of primary stage is passed to start method.
- ▶ We need to call show method on the primary stage object in order to show our primary stage.
- ▶ Initially, the primary Stage looks like following.
- ▶ However, we can add various objects to this primary stage.
- ▶ The objects can only be added in a hierarchical way i.e. first, scene graph will be added to this primaryStage and then that scene graph may contain the nodes.
- ▶ A node may be any object of the user's interface like text area, buttons, shapes, media, etc.
- ▶ It is represented by Stage class of the package javafx.stage.
- ▶ A stage has two parameters determining its position namely Width and Height. It is divided as Content Area and Decorations (Title Bar and Borders).
- ▶ You have to call the show() method to display the contents of a stage.



Application Structure of Java FX

Scene

- ▶ A scene represents the physical contents of a JavaFX application.
- ▶ It contains all the contents of a scene graph.
- ▶ The class Scene of the package javafx.scene represents the scene object. **javafx.scene.Scene** class provides all the methods to deal with a scene object.
- ▶ At an instance, the scene object is added to only one stage.
- ▶ You can create a scene by instantiating the Scene Class.
- ▶ You can opt for the size of the scene by passing its dimensions (height and width) along with the root node to its constructor.
- ▶ Creating scene is necessary in order to visualize the contents on the stage.
- ▶ In order to implement Scene in our JavaFX application, we must import javafx.scene package in our code.

Application Structure of Java FX

Scene Graph

- ▶ A scene graph is a tree-like data structure (hierarchical) representing the contents of a scene.
- ▶ In contrast, a node is a visual/graphical object of a scene graph. A node is the element which is visualized on the stage. It can be any button, text box, layout, image, radio button, check box, etc.
- ▶ A node may include –
 - ▶ Geometrical (Graphical) objects (2D and 3D) such as – Circle, Rectangle, Polygon, etc.
 - ▶ UI Controls such as – Button, Checkbox, Choice Box, Text Area, etc.
 - ▶ Containers (Layout Panes) such as Border Pane, Grid Pane, Flow Pane, etc.
 - ▶ Media elements such as Audio, Video and Image Objects. It can be seen as the collection of various nodes.
- ▶ The nodes are implemented in a tree kind of structure.
- ▶ There is always one root in the scene graph. This will act as a parent node for all the other nodes present in the scene graph. However, this node may be any of the layouts available in the JavaFX system.
- ▶ The leaf nodes exist at the lowest level in the tree hierarchy.
- ▶ Each of the node present in the scene graphs represents classes of javafx.scene package therefore we need to import the package into our application in order to create a full featured javafx application.

Application Structure of Java FX

Scene Graph

- ▶ As discussed earlier a node is of three types –
- ▶ **Root Node** – The first Scene Graph is known as the Root node.
- ▶ **Branch Node/Parent Node** – The node with child nodes are known as branch/parent nodes. The abstract class named Parent of the package javafx.scene is the base class of all the parent nodes, and those parent nodes will be of the following types –
 - ▶ Group – A group node is a collective node that contains a list of children nodes. Whenever the group node is rendered, all its child nodes are rendered in order. Any transformation, effect state applied on the group will be applied to all the child nodes.
 - ▶ Region – It is the base class of all the JavaFX Node based UI Controls, such as Chart, Pane and Control.
 - ▶ WebView – This node manages the web engine and displays its contents.
- ▶ **Leaf Node** – The node without child nodes is known as the leaf node. For example, Rectangle, Ellipse, Box, ImageView, MediaView are examples of leaf nodes.
- ▶ It is mandatory to pass the root node to the scene graph.

First JavaFX Application

Create a simple JavaFX application which prints **hello world** on the console on clicking the button shown on the stage.

Step-1 : Extend `javafx.application.Application` and override `start()`

- ▶ As we have studied earlier that `start()` method is the starting point of constructing a JavaFX application therefore we need to first override start method of `javafx.application.Application` class.
- ▶ Object of the class `javafx.stage.Stage` is passed into the `start()` method therefore import this class and pass its object into start method.
- ▶ `JavaFX.application.Application` needs to be imported in order to override start method.

```
package application;                                // creating source package. Created while making project in netbeans
import javafx.application.Application;               // importing Application class
import javafx.stage.Stage;
public class Hello_World extends Application{        // inheriting Application class

    @Override
    public void start(Stage primaryStage) throws Exception { // primaryStage is created. Object of primaryStage is
// JavaFX objects can be added here
    }
}
```

First JavaFX Application

Step-2 : Create a Button

- ▶ A button can be created by instantiating the `javafx.scene.control.Button` class.
- ▶ For this, we have to import this class into our code.
- ▶ Pass the button label text in Button class constructor.
- ▶ The code will look like following.

```
package application;                                // creating source package. Created while making project in netbeans
import javafx.application.Application;
import javafx.scene.control.Button;
import javafx.stage.Stage;
public class Hello_World extends Application{        // inheriting Application class

    @Override
    public void start(Stage primaryStage) throws Exception {    // Object of the Stage class is passed to abstract
method start() of Application class

Button btn1=new Button("Hello World");    // create a button and pass the caption on the button as parameter to
the constructor
}
}
```


First JavaFX Application

Step 3: Create a layout and add button to it

- ▶ JavaFX provides the number of layouts.
- ▶ We need to implement one of them in order to visualize the widgets properly.
- ▶ It exists at the top level of the scene graph and can be seen as a root node.
- ▶ All the other nodes (buttons, texts, etc.) need to be added to this layout.
- ▶ In this application, we have implemented StackPane layout.
- ▶ It can be implemented by instantiating javafx.scene.layout.StackPane class.
- ▶ The code will now look like following.

```
package application;                                // creating source package. Created while making project in netbeans
import javafx.application.Application;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class Hello_World extends Application{        // inheriting Application class
    @Override
    public void start(Stage primaryStage) throws Exception { // Object of the Stage class is passed to abstract
method start() of Application class
    Button btn1=new Button("Hello World"); // create a button and pass the caption on the button as parameter to
the constructor
    StackPane root=new StackPane(); // create layout object
    root.getChildren().add(btn1);
} }
```

First JavaFX Application

Step 4: Create a Scene

- ▶ The layout needs to be added to a scene.
- ▶ Scene remains at the higher level in the hierarchy of application structure.
- ▶ It can be created by instantiating javafx.scene.Scene class.
- ▶ We need to pass the layout object to the scene class constructor. we can also pass the width and height of the required stage for the scene in the Scene class constructor.
- ▶ Our application code will now look like following.

```
package application;                                // creating source package. Created while making project in netbeans
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class Hello_World extends Application{        // inheriting Application class
    @Override
    public void start(Stage primaryStage) throws Exception { // Object of the Stage class is passed to abstract method
start() of Application class
    Button btn1=new Button("Hello World"); // create a button and pass the caption on the button as parameter to the
constructor
    StackPane root=new StackPane();
    root.getChildren().add(btn1);
    Scene scene=new Scene(root, 300, 250); // pass layout object, width and height of the stage to Scene class constructor
} }
```

First JavaFX Application

Step 5: Prepare the Stage

- ▶ javafx.stage.Stage class provides some important methods which are required to be called to set some attributes for the stage.
- ▶ We can set the title of the stage.
- ▶ We also need to call show() method without which, the stage won't be shown.
- ▶ Lets look at the code which describes how can we prepare the stage for the application.

```
package application;                                // creating source package. Created while making project in netbeans
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class Hello_World extends Application{        // inheriting Application class
    @Override
    public void start(Stage primaryStage) throws Exception { // Object of the Stage class is passed to abstract method
start() of Application class
    Button btn1=new Button("Hello World"); // create a button and pass the caption on the button as parameter to the
constructor
    StackPane root=new StackPane();
    root.getChildren().add(btn1);
    Scene scene=new Scene(root, 300, 250); // pass layout object, width and height of the stage to Scene class constructor
    primaryStage.setScene(scene);
    primaryStage.setTitle("First JavaFX Application");
    primaryStage.show(); } }
```

First JavaFX Application

Step 6: Create an event for the button

- ▶ As our application prints hello world for an event on the button, we need to create an event for the button.
- ▶ For this purpose, call `setOnAction()` on the button and define a anonymous class Event Handler as a parameter to the method.
- ▶ Inside this anonymous class, define a method `handle()` which contains the code for how the event is handled.
- ▶ In our case, it is printing hello world on the console.

```
package application;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class Hello_World extends Application{
    @Override
    public void start(Stage primaryStage) throws Exception {

        // TODO Auto-generated method stub
        Button btn1=new Button("Hello World");
        btn1.setOnAction(new EventHandler<ActionEvent>() {
            @Override
```

```
        public void handle(ActionEvent arg0) {
            // TODO Auto-generated method stub
            System.out.println("Hello World !!!");
        }
    });
    StackPane root=new StackPane();
    root.getChildren().add(btn1);
    Scene scene=new Scene(root,600,400);
    primaryStage.setScene(scene);
    primaryStage.setTitle("First JavaFX Application");
    primaryStage.show();
}
```


First JavaFX Application

Step 7: Create a main method

- ▶ Till now, we have configured all the necessary things which are required to develop a basic JavaFX application but this application is still incomplete.
- ▶ We have not created main method yet.
- ▶ Hence, at the last, we need to create a main method in which we will launch the application i.e. will call launch() method and pass the command line arguments (args) to it.
- ▶ The code will now look like following.

```
package application;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class Hello_World extends Application{
    @Override
    public void start(Stage primaryStage) throws Exception {

        // TODO Auto-generated method stub
        Button btn1=new Button("Hello World");
        btn1.setOnAction(new EventHandler<ActionEvent>() {
            @Override
```

```
        public void handle(ActionEvent arg0) {
            // TODO Auto-generated method stub
            System.out.println("Hello World !!!");
        }
    });
    StackPane root=new StackPane();
    root.getChildren().add(btn1);
    Scene scene=new Scene(root,600,400);
    primaryStage.setScene(scene);
    primaryStage.setTitle("First JavaFX Application");
    primaryStage.show();
} }
```

```
public static void main (String[] args)
{
    launch(args);
}
}
```

```
StackPane root = new StackPane();
```

First JavaFX Application

Hello World

```
les\Java\jdk1.8.0_65\jre\..\lib\ant-javafx.jar  
t attribute should be used to restrict JAR repurposing.  
se property to override the current default non-secure value '*'.  
Files\Java\jdk1.8.0_65\jre\..\lib\ant-javafx.jar
```

```
\Documents\NetBeansProjects\JavaFXApplication1\dist\run589408307
```

```
Executing C:\Users\mpstme.student\Documents\NetBeansProjects\JavaFXApplication1\dist\run589408307\JavaFXApplicatio  
Hello World !!!
```

2D SHAPES

The background is a blue gradient, transitioning from a lighter blue at the top to a darker blue at the bottom. On the right side, there are several thin, white, parallel lines that run diagonally from the bottom-left towards the top-right, creating a sense of movement and depth.

2D Shapes

- ▶ In some of the applications, we need to show two dimensional shapes to the user.
- ▶ However, JavaFX provides the flexibility to create our own 2D shapes on the screen .
- ▶ There are various classes which can be used to implement 2D shapes in our application.
- ▶ All these classes resides in **javafx.scene.shape package**.
- ▶ This package contains the classes which represents different types of 2D shapes.
- ▶ There are several methods in the classes which deals with the coordinates regarding 2D shape creation.
- ▶ **What are 2D shapes?**
- ▶ In general, a two dimensional shape can be defined as the geometrical figure that can be drawn on the coordinate system consist of X and Y planes.
- ▶ Using JavaFX, we can create 2D shapes such as Line, Rectangle, Circle, Ellipse, Polygon, Cubic Curve, quad curve, Arc, etc. The class **javafx.scene.shape.Shape is the base class for all the shape classes**.
- ▶ Using the JavaFX library and Shape Class, you can draw –
 - ▶ Predefined shapes such as Line, Rectangle, Circle, Ellipse, Polygon, Polyline, Cubic Curve, Quad Curve, Arc.
 - ▶ Path elements such as MoveTO Path Element, Line, Horizontal Line, Vertical Line, Cubic Curve, Quadratic Curve, Arc.
 - ▶ In addition to these, you can also draw a 2D shape by parsing SVG path.
- ▶ To create a shape, you need to –
 - ▶ Instantiate the respective class of the required shape.
 - ▶ Set the properties of the shape.
 - ▶ Add the shape object to the group.

2D Shapes

Shape	Description
Line	In general, Line is the geometrical figure which joins two (X,Y) points on 2D coordinate system. In JavaFX, javafx.scene.shape.Line class needs to be instantiated in order to create lines.
Rectangle	In general, Rectangle is the geometrical figure with two pairs of two equal sides and four right angles at their joint. In JavaFX, javafx.scene.shape.Rectangle class needs to be instantiated in order to create Rectangles.
Ellipse	In general, ellipse can be defined as a curve with two focal points. The sum of the distances to the focal points are constant from each point of the ellipse. In JavaFX, javafx.scene.shape.Ellipse class needs to be instantiated in order to create Ellipse.
Arc	Arc can be defined as the part of the circumference of the circle of ellipse. In JavaFX, javafx.scene.shape.Arc class needs to be instantiated in order to create Arcs.
Circle	A circle is the special type of Ellipse having both the focal points at the same location. In JavaFX, Circle can be created by instantiating javafx.scene.shape.Circle class.
Polygon	Polygon is a geometrical figure that can be created by joining the multiple Co-planner line segments. In JavaFX, javafx.scene.shape.Polygon class needs to be instantiated in order to create polygon.
Cubic Curve	A Cubic curve is a curve of degree 3 in the XY plane. In Javafx, javafx.scene.shape.CubicCurve class needs to be instantiated in order to create Cubic Curves.
Quad Curve	A Quad Curve is a curve of degree 2 in the XY plane. In JavaFX, javafx.scene.shape.QuadCurve class needs to be instantiated in order to create QuadCurve.

Line

- ▶ Line can be defined as the geometrical structure which joins two points (X1,Y1) and (X2,Y2) in a X-Y coordinate plane.
- ▶ JavaFX allows the developers to create the line on the GUI of a JavaFX application.
- ▶ JavaFX library provides the class Line which is the part of javafx.scene.shape package.
- ▶ **How to create a Line?**
 - ▶ Instantiate the class `javafx.scene.shape.Line`.
 - ▶ set the required properties of the class object.
 - ▶ Add class object to the group
- ▶ Properties : Line class contains various properties described below.

Property	Description	Setter Methods
endX	The X coordinate of the end point of the line	setEndX(Double)
endY	The y coordinate of the end point of the line	setEndY(Double)
startX	The x coordinate of the starting point of the line	setStartX(Double)
startY	The y coordinate of the starting point of the line	setStartY(Double)

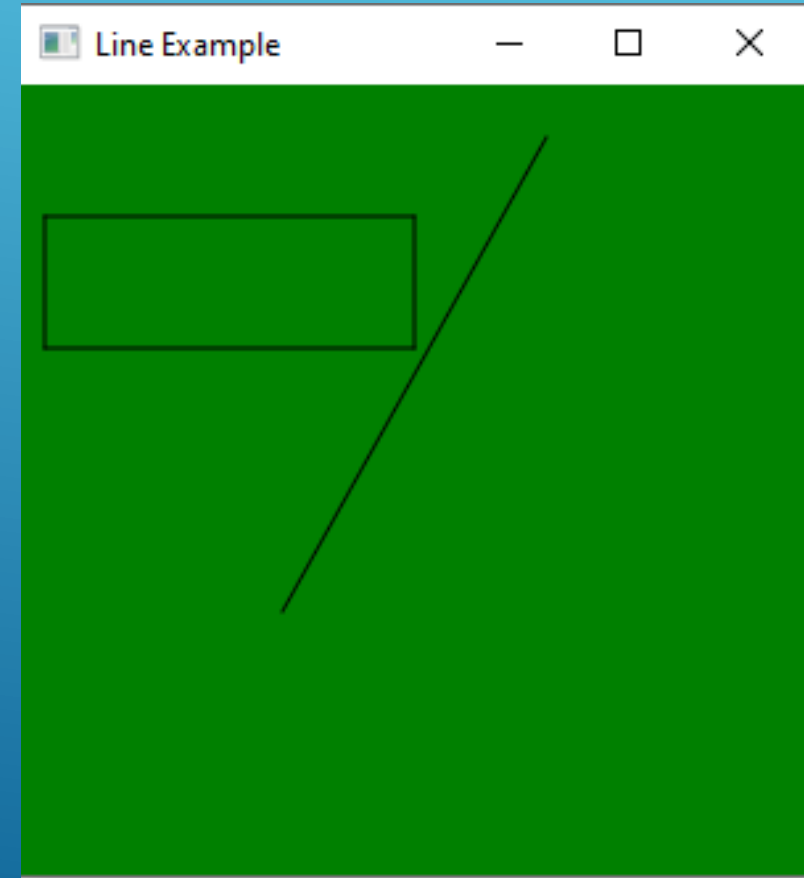
Line

```
package javafxapplication1;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.shape.Line;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
public class DrawLine extends Application{

    @Override
    public void start(Stage primaryStage) throws Exception {
        // TODO Auto-generated method stub
        Line line = new Line(); //instantiating Line class
        line.setStartX(200); //setting starting X point of Line
        line.setStartY(20); //setting starting Y point of Line
        line.setEndX(100); //setting ending X point of Line
        line.setEndY(200); //setting ending Y point of Line
        //adding multiple lines
        Line line1 = new Line(10,50,150,50);
        Line line2 = new Line(10,100,150,100);
        Line line3 = new Line(10,50,10,100);
        Line line4 = new Line(150,50,150,100);
        Group root = new Group(line, line1, line2, line3,
line4); //Creating a Group and adding line object to the
group
```

```
Scene scene = new Scene(root,300,300,Color.GREEN); //
set background color
        primaryStage.setScene(scene);
        primaryStage.setTitle("Line Example");
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    } }
```



Rectangle

- ▶ In general, Rectangles can be defined as the geometrical figure consists of four sides, out of which, the opposite sides are always equal and the angle between the two adjacent sides is 90 degree.
- ▶ A Rectangle with four equal sides is called square.
- ▶ JavaFX library allows the developers to create a rectangle by instantiating `javafx.scene.shape.Rectangle` class
- ▶ Properties of Rectangle Class -

Property	Description	Setter Method
ArcHeight	Vertical diameter of the arc at the four corners of rectangle	<code>setArcHeight(Double height)</code>
ArcWidth	Horizontal diameter of the arc at the four corners of the rectangle	<code>setArcWidth(Double Width)</code>
Height	Defines the height of the rectangle	<code>setHeight(Double height)</code>
Width	Defines the width of the rectangle	<code>setWidth(Double width)</code>
X	X coordinate of the upper left corner	<code>setX(Double X-value)</code>
Y	Y coordinate of the upper left corner	<code>setY(Double(Y-value))</code>

Rectangle

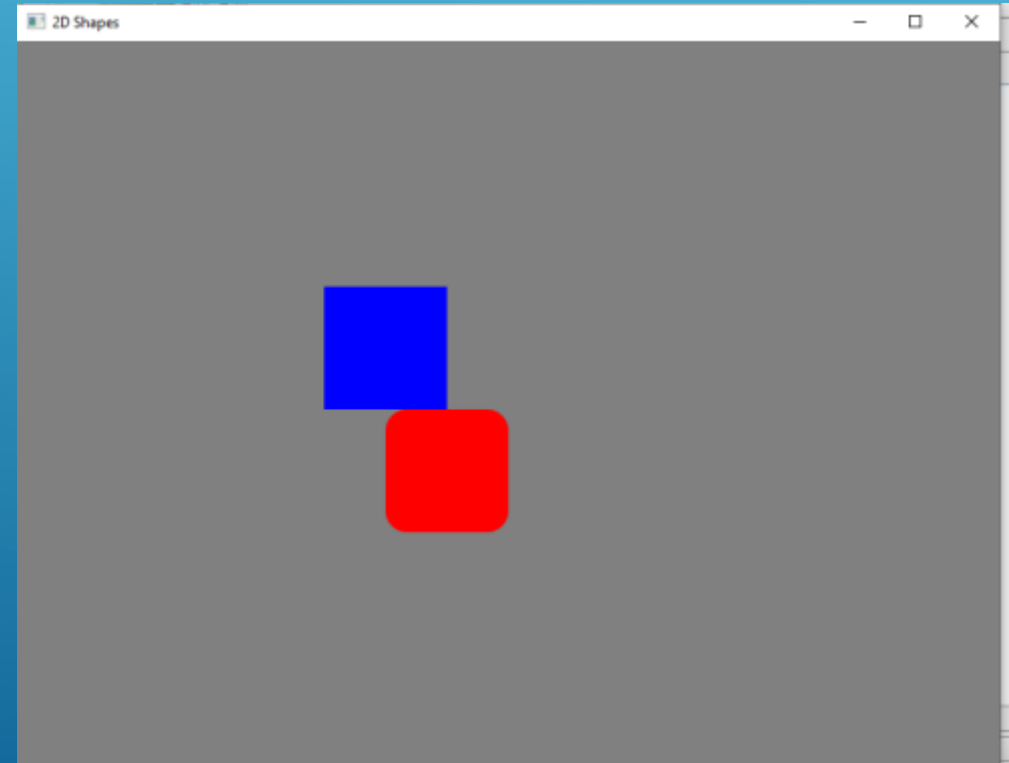
```
package javafxapplication1;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
public class Drawrect extends Application {

    @Override
    public void start(Stage primaryStage)throws Exception {
        Rectangle rect=new Rectangle();
        rect.setX(250);
        rect.setY(200);
        rect.setWidth(100);
        rect.setHeight(100);
        rect.setFill(Color.BLUE);
        // Rounded Rectangle
        Rectangle rect1=new Rectangle();
        rect1.setX(300);
        rect1.setY(300);
        rect1.setWidth(100);
        rect1.setHeight(100);
        rect1.setArcHeight(35.0);
        rect1.setArcWidth(35.0);
        rect1.setFill(Color.RED);
```

```
        Group root = new Group(rect, rect1);
        Scene scene = new Scene(root,800,800,Color.GRAY);

        primaryStage.setTitle("2D Shapes");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```



Ellipse

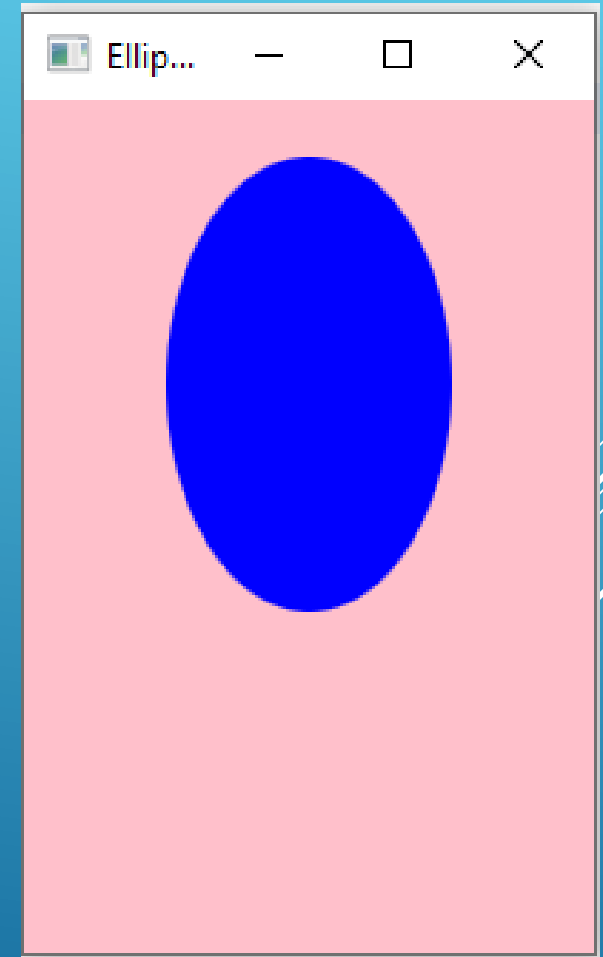
- ▶ In general, ellipse can be defined as the geometrical structure with the two focal points.
- ▶ The focal points in the ellipse are chosen so that the sum of the distance to the focal points is constant from every point of the ellipse.
- ▶ In JavaFX, the class `javafx.scene.shape.Ellipse` represents Ellipse.
- ▶ This class needs to be instantiated in order to create ellipse.
- ▶ This class contains various properties which needs to be set in order to render ellipse on a XY place.

Property	Description	Setter Methods
CenterX	Horizontal position of the centre of eclipse	<code>setCenterX(Double X-value)</code>
CenterY	Vertical position of the centre of eclipse	<code>setCenterY(Double Y-value)</code>
RadiusX	Width of Eclipse	<code>setRadiusX(Double X-Radius Vaue)</code>
RadiusY	Height of Eclipse	<code>setRadiusY(Double Y-Radius Value)</code>

Ellipse

```
package javafxapplication1;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.shape.Ellipse;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
public class DrawEllipse extends Application {

    @Override
    public void start(Stage primaryStage)throws Exception {
        Ellipse e = new Ellipse();
        e.setCenterX(100);
        e.setCenterY(100);
        e.setRadiusX(50);
        e.setRadiusY(80);
        e.setFill(Color.BLUE);
        Group root = new Group(e);
        Scene scene = new Scene(root,200,300,Color.PINK);
        primaryStage.setTitle("Ellipse Example");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```



Arc

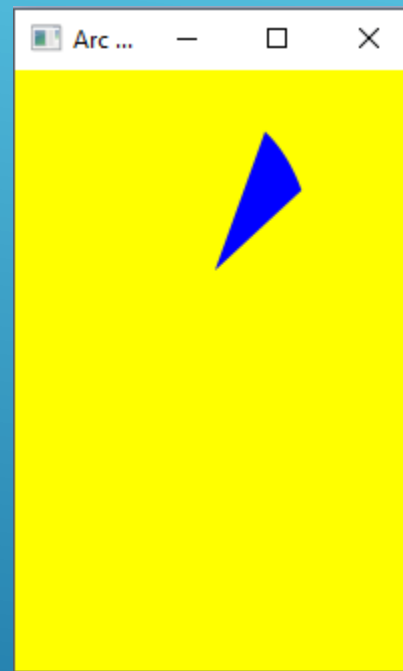
- ▶ In general, Arc is the part of the circumference of a circle or ellipse.
- ▶ It needs to be created in some of the JavaFX applications wherever required.
- ▶ JavaFX allows us to create the Arc on GUI by just instantiating `javafx.scene.shape.Arc` class.
- ▶ Just set the properties of the class to the appropriate values to show arc as required by the Application.

Property	Description	Method
CenterX	X coordinate of the centre point	<code>setCenterX(Double value)</code>
CenterY	Y coordinate of the centre point	<code>setCenterY(Double value)</code>
Length	Angular extent of the arc in degrees	<code>setLength(Double value)</code>
RadiusX	Full width of the ellipse of which, Arc is a part.	<code>setRadiusX(Double value)</code>
RadiusY	Full height of the ellipse of which, Arc is a part	<code>setRadiusY(Double value)</code>
StartAngle	Angle of the arc in degrees	<code>setStartAngle(Double value)</code>
type	Type of Arc : OPEN, CHORD, ROUND	<code>setType(Double value)</code>

Arc

```
package javafxapplication1;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Arc;
import javafx.scene.shape.ArcType;
import javafx.stage.Stage;
public class DrawArc extends Application{
    @Override
    public void start(Stage primaryStage) throws Exception {
Arc arc = new Arc();
    arc.setCenterX(100);
    arc.setCenterY(100);
    arc.setRadiusX(50);
    arc.setRadiusY(80);
    arc.setStartAngle(30);
    arc.setLength(30);
    arc.setType(ArcType.ROUND);
    arc.setFill(Color.BLUE);
    Group root = new Group(arc);
    Scene scene = new Scene(root,200,300,Color.YELLOW);
    primaryStage.setTitle("Arc Example");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

```
public static void main(String[] args) {
    launch(args);
} }
```



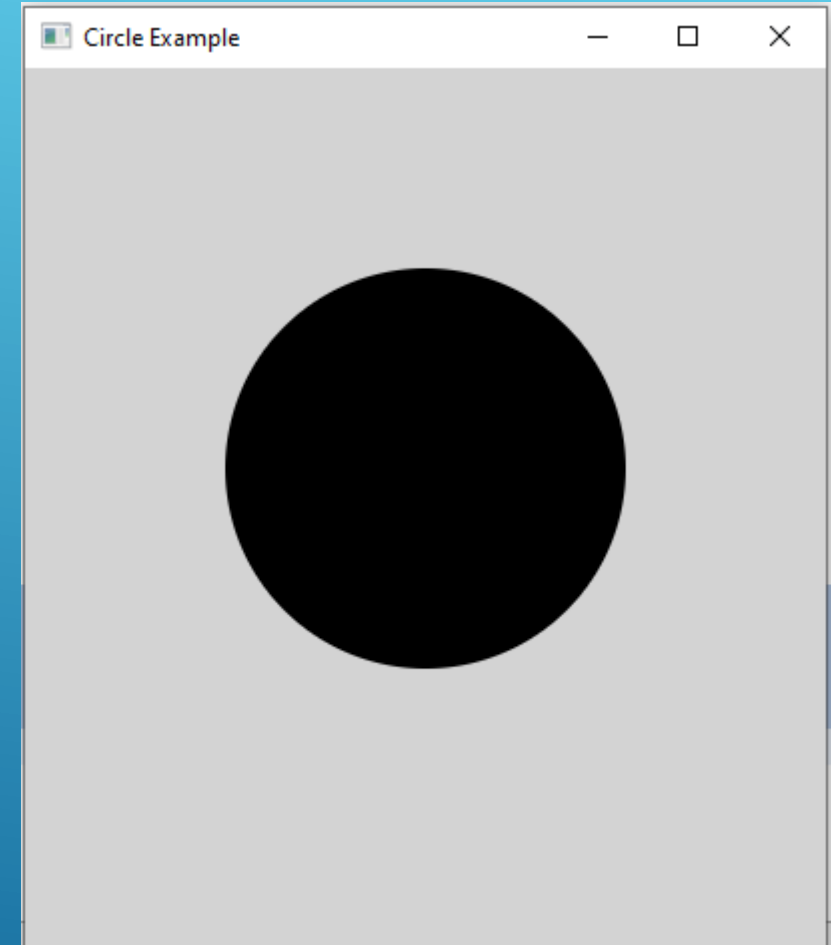
Circle

- ▶ A circle is a special type of ellipse with both of the focal points at the same position.
- ▶ Its horizontal radius is equal to its vertical radius.
- ▶ JavaFX allows us to create Circle on the GUI of any application by just instantiating `javafx.scene.shape.Circle` class.
- ▶ Just set the class properties by using the instance setter methods and add the class object to the Group.

Property	Description	Setter Methods
centerX	X coordinate of the centre of circle	setCenterX(Double value)
centerY	Y coordinate of the centre of circle	setCenterY(Double value)
radius	Radius of the circle	setRadius(Double value)

Circle

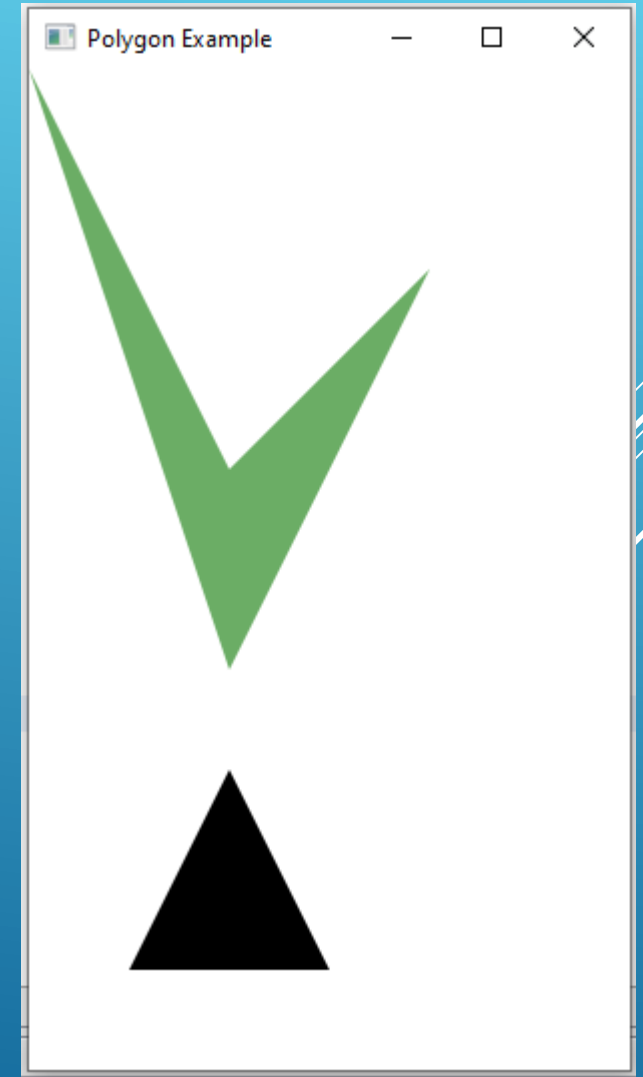
```
package javafxapplication1;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Arc;
import javafx.scene.shape.ArcType;
import javafx.stage.Stage;
public class DrawCircle extends Application{
    @Override
    public void start(Stage primaryStage) throws Exception {
Circle c = new Circle();
    c.setCenterX(200);
    c.setCenterY(200);
    c.setRadius(100);
    Group root = new Group(c);
    Scene scene = new Scene(root,400,500,Color.LIGHTGRAY);
    primaryStage.setTitle("Circle Example");
    primaryStage.setScene(scene);
    primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    } }
```



Polygons

```
package javafxapplication1;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.shape.Polygon;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
public class DrawPolygon extends Application {
    @Override
    public void start(Stage primarystage) {
        Polygon poly = new Polygon(0.0,0.0,100.0, 200.0, 200.0, 100.0, 100.0,300.0);
        Polygon poly1 =new Polygon(100.0, 350.0, 50.0, 450.0, 150.0, 450.0);
        // setting color based on rgb values
        int red=20, green=125, blue=10;
        poly.setFill(Color.rgb(red, green, blue,0.63));
        Group root = new Group(poly, poly1);
        Scene scene = new Scene(root,500,500);
        primarystage.setTitle("Polygon Example");
        primarystage.setScene(scene);
        primarystage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```



Text

- ▶ In some of the cases, we need to provide the text based information on the interface of our application.
- ▶ JavaFX library provides a class named `javafx.scene.text.Text` for this purpose.
- ▶ This class provides various methods to alter various properties of the text.
- ▶ We just need to instantiate this class to implement text in our application.
- ▶ Use the setter method `setText(string)` to set the string as a text for the text class object.
- ▶ Follow the syntax given below to instantiate the Text class.

```
Text <text_Object> = new Text();  
text.setText(<string-text>);
```
- ▶ By default, the text will be displayed to the center of the screen.
- ▶ **Font and position of the Text**
- ▶ JavaFX enables us to apply various fonts to the text nodes.
- ▶ We just need to set the property font of the Text class by using the setter method `setFont()`.
- ▶ This method accepts the object of Font class.
- ▶ The class Font belongs to the package `javafx.scene.text`.
- ▶ It contains a static method named `font()`.
- ▶ This returns an object of Font type which will be passed as an argument into the `setFont()` method of Text class.

Text

- ▶ The method `Font.font()` accepts the following parameters.
- ▶ **Family:** it represents the family of the font. It is of string type and should be an appropriate font family present in the system.
- ▶ **Weight:** this Font class property is for the weight of the font. There are 9 values which can be used as the font weight. The values are `FontWeight`. `BLACK`, `BOLD`, `EXTRA_BOLD`, `EXTRA_LIGHT`, `LIGHT`, `MEDIUM`, `NORMAL`, `SEMI_BOLD`, `THIN`.
- ▶ **Posture:** this Font class property represents the posture of the font. It can be either `FontPosture.ITALIC` or `FontPosture.REGULAR`.
- ▶ **Size:** this is a double type property. It is used to set the size of the font.
- ▶ The Syntax of the method `setFont()` is given below.
`<text_object>.setFont(Font.font(<String font_family>, <FontWeight>, <FontPosture>, <FontSize>))`

- ▶ **Applying Stroke and Color to Text**

- ▶ Stroke means the padding at the boundary of the text.
- ▶ JavaFX allows us to apply stroke and colors to the text.
- ▶ `javafx.scene.text.Text` class provides a method named `setStroke()` which accepts the `Paint` class object as an argument. Just pass the color which will be painted on the stroke.
- ▶ We can also set the width of the stroke by passing a width value of double type into `setStrokeWidth()` method.
- ▶ To set the color of the Text, `javafx.scene.text.Text` class provides another method named `setFill()`. We just need to pass the color which is to be filled in the text.

Text

- ▶ **Text Decoration**

- ▶ We can apply the decorations to the text by setting the properties `strikethrough` and `underline` of `javafx.scene.text.Text` class.
- ▶ The syntax of both the methods is given below.

`<TextObject>.setStrikeThrough(Boolean value) //pass true to put a line across the text`

`<TextObject>.setUnderLine(Boolean value) //pass true to underline the text`

Text

Property	Description	Setter Methods
boundstype	This property is of object type. It determines the way in which the bounds of the text is being calculated.	setBoundsType(TextBoundsType value)
font	Font of the text.	setFont(Font value)
fontsmoothingType	Defines the requested smoothing type for the font.	setFontSmoothingType(FontSmoothingType value)
linespacing	Vertical space in pixels between the lines. It is double type property.	setLineSpacing(double spacing)
strikethrough	This is a boolean type property. We can put a line through the text by setting this property to true.	setStrikeThrough(boolean value)
textalignment	Horizontal Text alignment	setTextAlignment(TextAlignment value)
textorigin	Origin of text coordinate system in local coordinate system.	setTextOrigin(VPos value)
text	It is a string type property. It defines the text string which is to be displayed.	setText(String value)
underline	It is a boolean type property. We can underline the text by setting this property to true.	setUnderLine(boolean value)
wrappingwidth	Width limit for the text from where the text is to be wrapped. It is a double type property.	setWidth(double value)
x	X coordinate of the text	setX(double value)
y	Y coordinate of the text	setY(double value)

Text

```
package javafxapplication1;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
import javafx.scene.text.TextAlignment;
//import javafx.scene.layout.StackPane;
public class JavaFXText extends Application{
    @Override
    public void start(Stage primaryStage) throws Exception {
        Text text = new Text("Hello !! Welcome to Java FX");
        text.setX(50);
        text.setY(100);

        text.setFont(Font.font("Arial",FontWeight.EXTRA_BOLD,Font
        Posture.ITALIC,50)); // font formatting
        text.setFill(Color.YELLOW); // setting colour of the text
        text.setStroke(Color.BLACK); // setting the border for the
        text
        text.setStrokeWidth(3); // setting border width to 3
        text.setUnderline(true); // underline the text
```

Text text1 = new Text("The line spacing property of the javafx.scene.text.The text class specifies the line spacing between the lines of the text (node) vertically.You can set the value to this property using the setLineSpacing() method. This method accepts a boolean value as a parameter and sets the specified space between the lines (vertically).");

```
        text1.setX(50);
        text1.setY(300);
        text1.setStrikethrough(true); // strike the text
        text1.setTextAlignment(TextAlignment.CENTER);
//Aligning text
        text1.setLineSpacing(10); // Line Spacing
        text1.setWrappingWidth(300); // text wrapping
        Group root = new Group(text, text1);
        Scene scene = new Scene(root,800,400);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Text Example");
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    } }
```

Text



Java FX Layouts

Java FX Layouts

- ▶ Layouts are the top level container classes that define the UI styles for scene graph objects.
- ▶ Layout can be seen as the parent node to all the other nodes.
- ▶ JavaFX provides various layout panes that support different styles of layouts.
- ▶ In JavaFX, Layout defines the way in which the components are to be seen on the stage.
- ▶ It basically organizes the scene-graph nodes.
- ▶ We have several built-in layout panes in JavaFX that are HBox, VBox, StackPane, FlowBox, AnchorPane, etc.
- ▶ Each Built-in layout is represented by a separate class which needs to be instantiated in order to implement that particular layout pane.
- ▶ All these classes belong to `javafx.scene.layout` package. `javafx.scene.layout.Pane` class is the base class for all the built-in layout classes in JavaFX..
- ▶ **Steps to create layout**
- ▶ In order to create the layouts, we need to follow the following steps.
 - ▶ Instantiate the respective layout class, for example, `HBox root = new HBox();`
 - ▶ Setting the properties for the layout, for example, `root.setSpacing(20);`
 - ▶ Adding nodes to the layout object, for example, `root.getChildren().addAll(<NodeObjects>);`

Java FX Layouts

Class	Description
BorderPane	Organizes nodes in top, left, right, centre and the bottom of the screen.
FlowPane	Organizes the nodes in the horizontal rows according to the available horizontal spaces. Wraps the nodes to the next line if the horizontal space is less than the total width of the nodes
GridPane	Organizes the nodes in the form of rows and columns.
HBox	Organizes the nodes in a single row.
Pane	It is the base class for all the layout classes.
StackPane	Organizes nodes in the form of a stack i.e. one onto another
VBox	Organizes nodes in a vertical column.

BorderPane

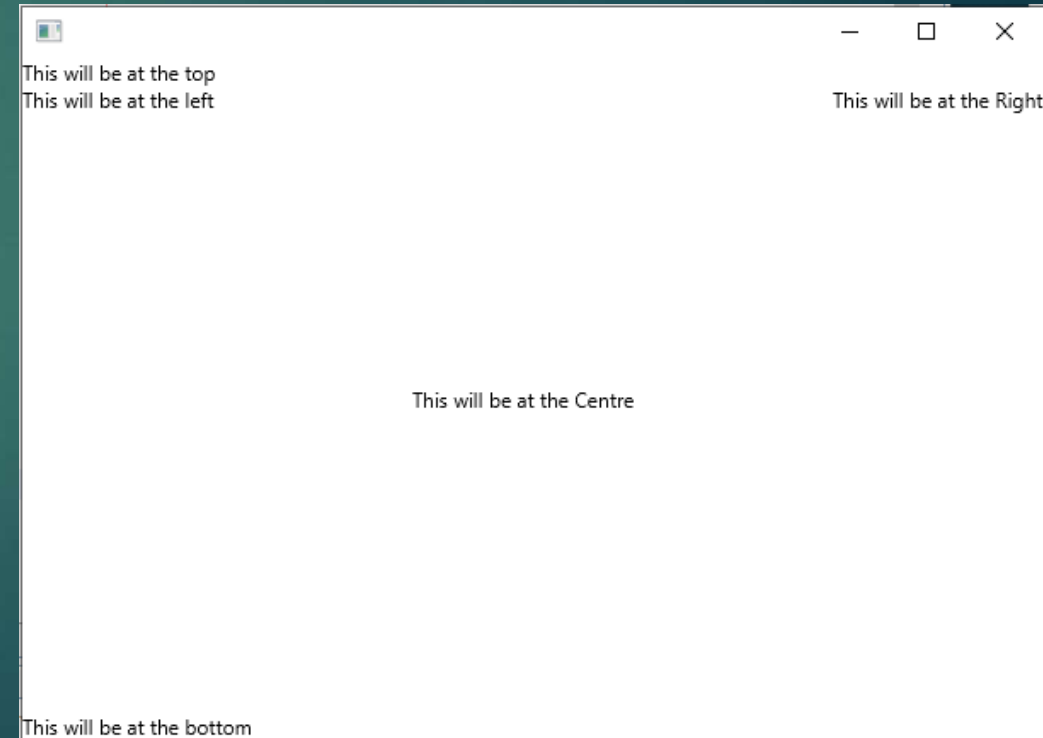
- ▶ BorderPane arranges the nodes at the left, right, centre, top and bottom of the screen.
- ▶ It is represented by `javafx.scene.layout.BorderPane` class.
- ▶ This class provides various methods like `setRight()`, `setLeft()`, `setCenter()`, `setBottom()` and `setTop()` which are used to set the position for the specified nodes.
- ▶ We need to instantiate `BorderPane` class to create the `BorderPane` layout.

Type	Property	Setter Methods	Description
Node	Bottom	<code>setBottom()</code>	Add the node to the bottom of the screen
Node	Centre	<code>setCenter()</code>	Add the node to the center of the screen
Node	Left	<code>setLeft()</code>	Add the node to the left of the screen
Node	Right	<code>setRight()</code>	Add the node to the right of the screen
Node	Top	<code>setTop()</code>	Add the node to the top of the screen

- ▶ There are the following constructors in the class.
 - ▶ `BorderPane()` : create the empty layout
 - ▶ `BorderPane(Node Center)` : create the layout with the center node
 - ▶ `BorderPane(Node Center, Node top, Node right, Node bottom, Node left)` : create the layout with all the nodes

BorderPane

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.*;
import javafx.stage.Stage;
import javafx.scene.text.Text;
public class BorderPaneDemo extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        BorderPane BPane = new BorderPane();
        BPane.setTop(new Text("This will be at the top"));
        BPane.setLeft(new Text("This will be at the left"));
        BPane.setRight(new Text("This will be at the Right"));
        BPane.setCenter(new Text("This will be at the Centre"));
        BPane.setBottom(new Text("This will be at the bottom"));
        Scene scene = new Scene(BPane,600,400);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    } }
```



HBox

- ▶ HBox layout pane arranges the nodes in a single row.
- ▶ It is represented by `javafx.scene.layout.HBox` class.
- ▶ We just need to instantiate HBox class in order to create HBox layout.

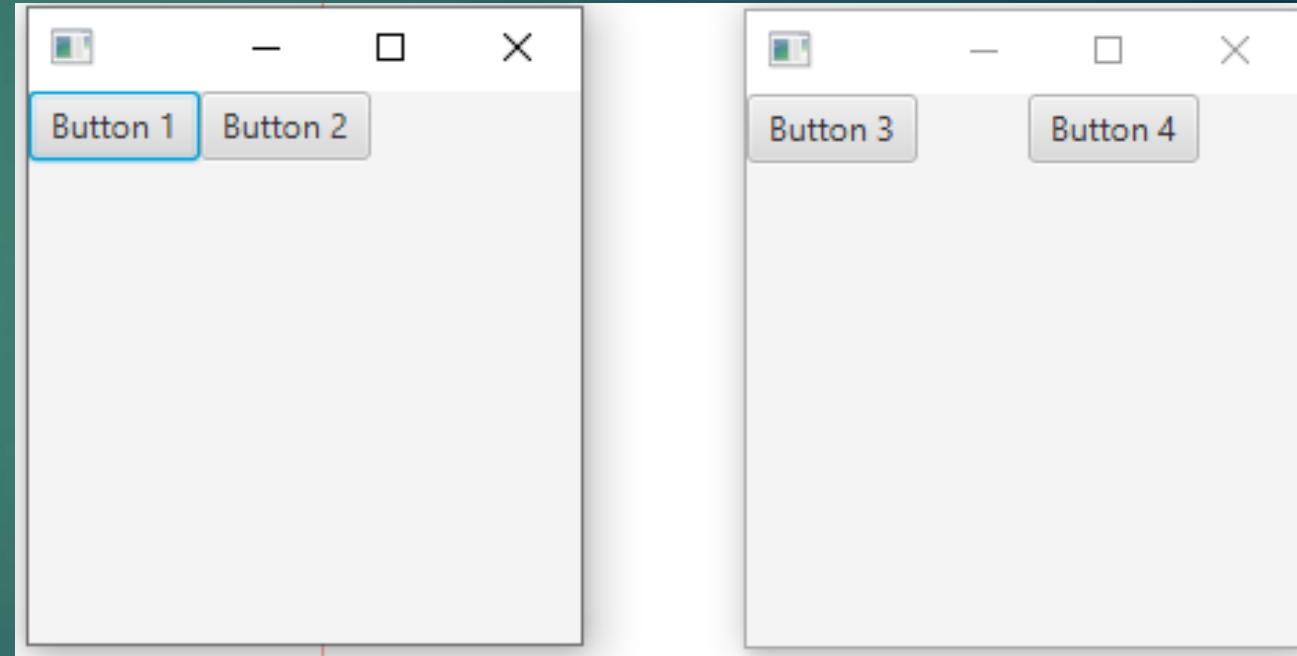
Property	Description	Setter Methods
alignment	This represents the alignment of the nodes.	<code>setAlignment(Double)</code>
fillHeight	This is a boolean property. If you set this property to true the height of the nodes will become equal to the height of the HBox.	<code>setFillHeight(Double)</code>
spacing	This represents the space between the nodes in the HBox. It is of double type.	<code>setSpacing(Double)</code>

- ▶ The HBox class contains two constructors that are given below.
 - ▶ `new HBox()` : create HBox layout with 0 spacing
 - ▶ `new HBox(Double spacing)` : create HBox layout with a spacing value

HBox

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class HBoxDemo extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        Button btn1 = new Button("Button 1");
        Button btn2 = new Button("Button 2");
        Button btn3 = new Button("Button 3");
        Button btn4 = new Button("Button 4");
        HBox root = new HBox(btn1,btn2);
        Scene scene = new Scene(root,200,200);
        primaryStage.setScene(scene);
        primaryStage.show();

        HBox root1 = new HBox(btn3,btn4);
        root1.setSpacing(40);
        Scene scene1 = new Scene(root1,200,200);
        Stage newStage=new Stage();
        newStage.setScene(scene1);
        newStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    } }
```



VBox

- ▶ Instead of arranging the nodes in horizontal row, VBox Layout Pane arranges the nodes in a single vertical column.
- ▶ It is represented by `javafx.scene.layout.VBox` class which provides all the methods to deal with the styling and the distance among the nodes.
- ▶ This class needs to be instantiated in order to implement VBox layout in our application.

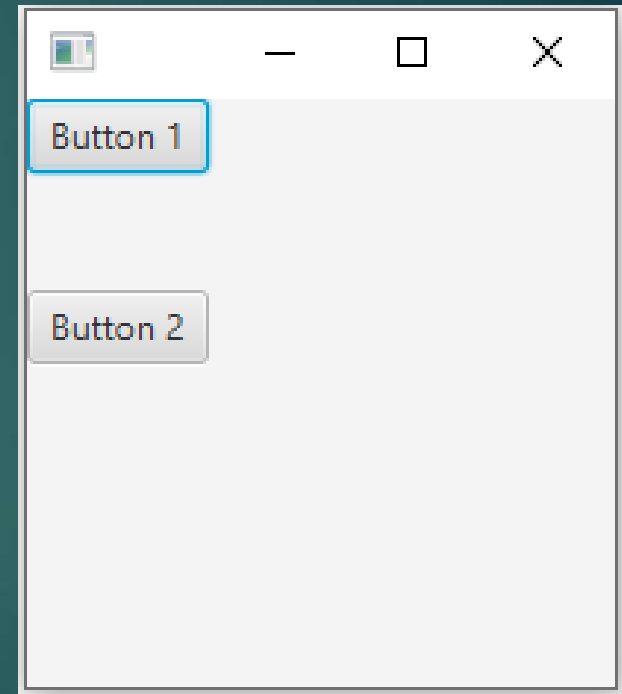
Property	Description	Setter Methods
Alignment	This property is for the alignment of the nodes.	<code>setAlignment(Double)</code>
FillWidth	This property is of the boolean type. The Width of resizable nodes can be made equal to the Width of the VBox by setting this property to true.	<code>setFillWidth(boolean)</code>
Spacing	This property is to set some spacing among the nodes of VBox.	<code>setSpacing(Double)</code>

- ▶ The VBox class contains constructors that are given below.
 - ▶ `VBox()` : creates layout with 0 spacing
 - ▶ `Vbox(Double spacing)` : creates layout with a spacing value of double type
 - ▶ `Vbox(Double spacing, Node? children)` : creates a layout with the specified spacing among the specified child nodes
 - ▶ `Vbox(Node? children)` : creates a layout with the specified nodes having 0 spacing among them

VBox

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
public class VBoxDemo extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Button btn1 = new Button("Button 1");
        Button btn2 = new Button("Button 2");
        VBox root = new VBox(btn1,btn2);
        root.setSpacing(40);
        Scene scene = new Scene(root,200,200);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    } }
```



StackPane

- ▶ The StackPane layout pane places all the nodes into a single stack where every new node gets placed on the top of the previous node.
- ▶ It is represented by `javafx.scene.layout.StackPane` class.
- ▶ We just need to instantiate this class to implement StackPane layout into our application.

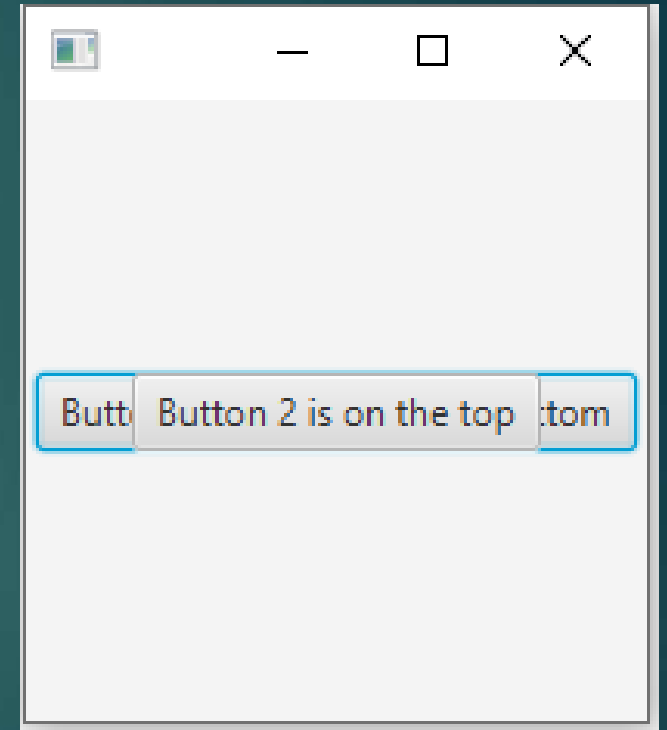
Property	Description	Setter Method
alignment	It represents the default alignment of children within the StackPane's width and height	<code>setAlignment(Node child, Pos value)</code> <code>setAlignment(Pos value)</code>

- ▶ The class contains two constructors that are given below.
 - ▶ `StackPane()`
 - ▶ `StackPane(Node? Children)`

StackPane

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class StackPaneDemo extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Button btn1 = new Button("Button 1 is stacked to the bottom");
        Button btn2 = new Button("Button 2 is on the top");
        StackPane root = new StackPane(btn1,btn2);
        Scene scene = new Scene(root,200,200);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```



GridPane

- ▶ GridPane Layout pane allows us to add the multiple nodes in multiple rows and columns.
- ▶ It is seen as a flexible grid of rows and columns where nodes can be placed in any cell of the grid.
- ▶ It is represented by `javafx.scene.layout.GridPane` class. We just need to instantiate this class to implement GridPane.



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

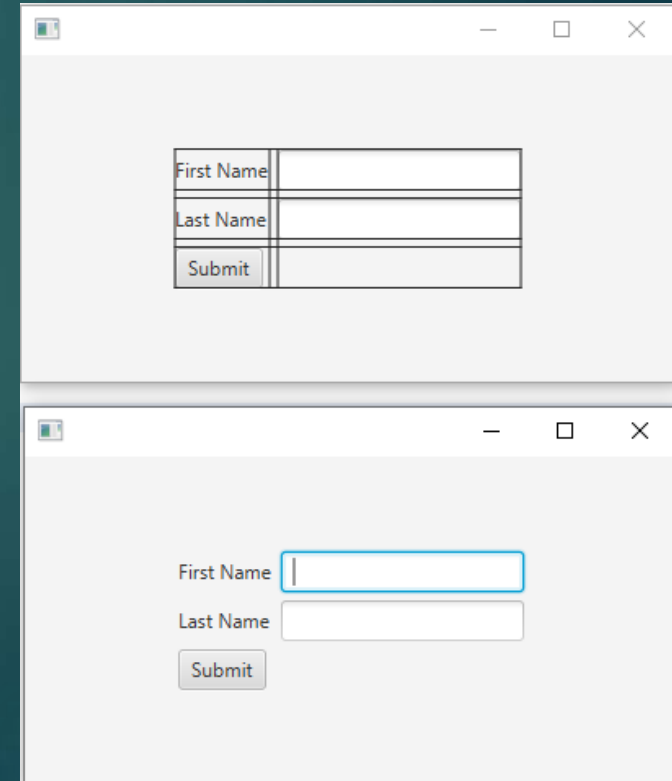
- ▶ The class contains only one constructor that is given below.
 - ▶ `Public GridPane()`: creates a gridpane with 0 hgap/vgap.

GridPane

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class GridPaneDemo extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        Label first_name=new Label("First Name");
        Label last_name=new Label("Last Name");
        TextField tf1=new TextField();
        TextField tf2=new TextField();
        Button Submit=new Button ("Submit");
        GridPane root=new GridPane();
        Scene scene = new Scene(root,400,200);
        root.add(first_name,0,0);
        root.add(tf1,1,0);
        root.addRow(1, last_name,tf2);
        root.addRow(2, Submit);
        root.setAlignment(Pos.CENTER);
```

```
//root.setGridLinesVisible(false);
        root.setGridLinesVisible(true);
        root.setHgap(5);
        root.setVgap(5);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    } }
```



FlowPane

- ▶ FlowPane layout pane organizes the nodes in a flow that are wrapped at the flowpane's boundary.
- ▶ The horizontal flowpane arranges the nodes in a row and wrap them according to the flowpane's width.
- ▶ The vertical flowpane arranges the nodes in a column and wrap them according to the flowpane's height.
- ▶ FlowPane layout is represented by `javafx.scene.layout.FlowPane` class.

Property	Description	Setter Methods
alignment	The overall alignment of the flowpane's content.	<code>setAlignment(Pos value)</code>
columnHalignment	The horizontal alignment of nodes within the columns.	<code>setColumnHalignment(HPos Value)</code>
hgap	Horizontal gap between the columns.	<code>setHgap(Double value)</code>
orientation	Orientation of the flowpane	<code>setOrientation(Orientation value)</code>
prefWrapLength	The preferred height or width where content should wrap in the horizontal or vertical flowpane.	<code>setPrefWrapLength(double value)</code>
rowValignment	The vertical alignment of the nodes within the rows.	<code>setRowValignment(VPos value)</code>
vgap	The vertical gap among the rows	<code>setVgap(Double value)</code>

FlowPane

- ▶ There are 8 constructors in the class that are given below.
 - ▶ FlowPane()
 - ▶ FlowPane(Double Hgap, Double Vgap)
 - ▶ FlowPane(Double Hgap, Double Vgap, Node? children)
 - ▶ FlowPane(Node... Children)
 - ▶ FlowPane(Orientation orientation)
 - ▶ FlowPane(Orientation orientation, double Hgap, Double Vgap)
 - ▶ FlowPane(Orientation orientation, double Hgap, Double Vgap, Node? children)
 - ▶ FlowPane(Orientation orientation, Node... Children)

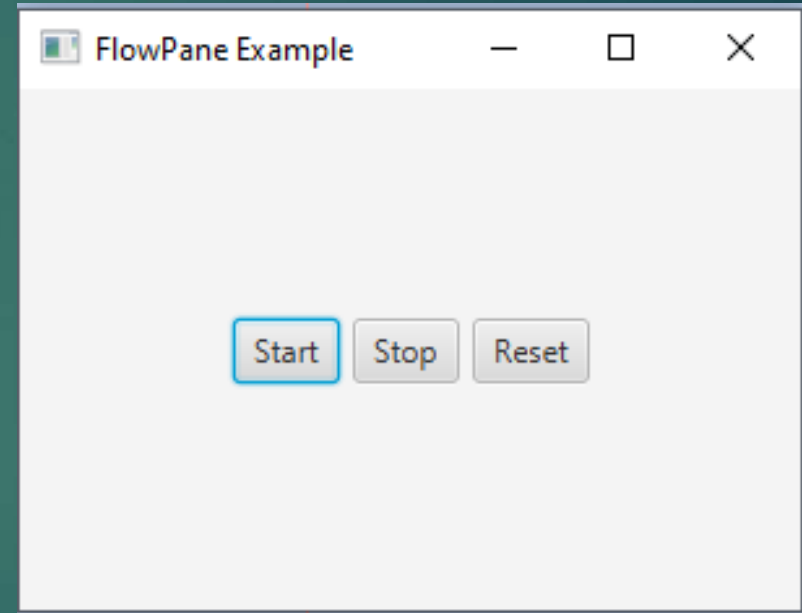
FlowPane

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;
public class FlowPaneDemo extends Application {
```

@Override

```
public void start(Stage primaryStage) {
    Button btn1=new Button("Start");
    Button btn2=new Button("Stop");
    Button btn3=new Button("Reset");
    FlowPane root = new FlowPane(btn1, btn2, btn3);
    root.setVgap(6);
    root.setHgap(5);
    root.setPrefWrapLength(250);
    root.setAlignment(Pos.CENTER);
    Scene scene = new Scene(root,300,200);
    primaryStage.setTitle("FlowPane Example");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

```
public static void main(String[] args) {
    launch(args);
}
```



UI Controls

Java FX UI Controls

- ▶ The graphical user interface of every desktop application mainly considers UI elements, layouts and behaviour.
- ▶ The UI elements are the one which are actually shown to the user for interaction or information exchange.
- ▶ Layout defines the organization of the UI elements on the screen.
- ▶ Behaviour is the reaction of the UI element when some event is occurred on it.
- ▶ However, the package `javafx.scene.control` provides all the necessary classes for the UI components like Button, Label, etc.
- ▶ Every class represents a specific UI control and defines some methods for their styling.

Java FX UI Controls

SN	Control	Description
1	Label	Label is a component that is used to define a simple text on the screen. Typically, a label is placed with the node, it describes.
2	Button	Button is a component that controls the function of the application. Button class is used to create a labelled button.
3	RadioButton	The Radio Button is used to provide various options to the user. The user can only choose one option among all. A radio button is either selected or deselected.
4	CheckBox	Check Box is used to get the kind of information from the user which contains various choices. User marked the checkbox either on (true) or off (false).
5	TextField	Text Field is basically used to get the input from the user in the form of text. <code>javafx.scene.control.TextField</code> represents TextField
6	PasswordField	PasswordField is used to get the user's password. Whatever is typed in the password field is not shown on the screen to anyone.
7	HyperLink	HyperLink are used to refer any of the webpage through your application. It is represented by the class <code>javafx.scene.control.HyperLink</code>
8	Slider	Slider is used to provide a pane of options to the user in a graphical form where the user needs to move a slider over the range of values to select one of them.
9	ProgressBar	Progress Bar is used to show the work progress to the user. It is represented by the class <code>javafx.scene.control.ProgressBar</code> .
10	ProgressIndicator	Instead of showing the analogue progress to the user, it shows the digital progress so that the user may know the amount of work done in percentage.
11	ScrollBar	JavaFX Scroll Bar is used to provide a scroll bar to the user so that the user can scroll down the application pages.
12	Menu	JavaFX provides a Menu class to implement menus. Menu is the main component of any application.
13	ToolTip	JavaFX ToolTip is used to provide hint to the user about any component. It is mainly used to provide hints about the text fields or password fields being used in the application.

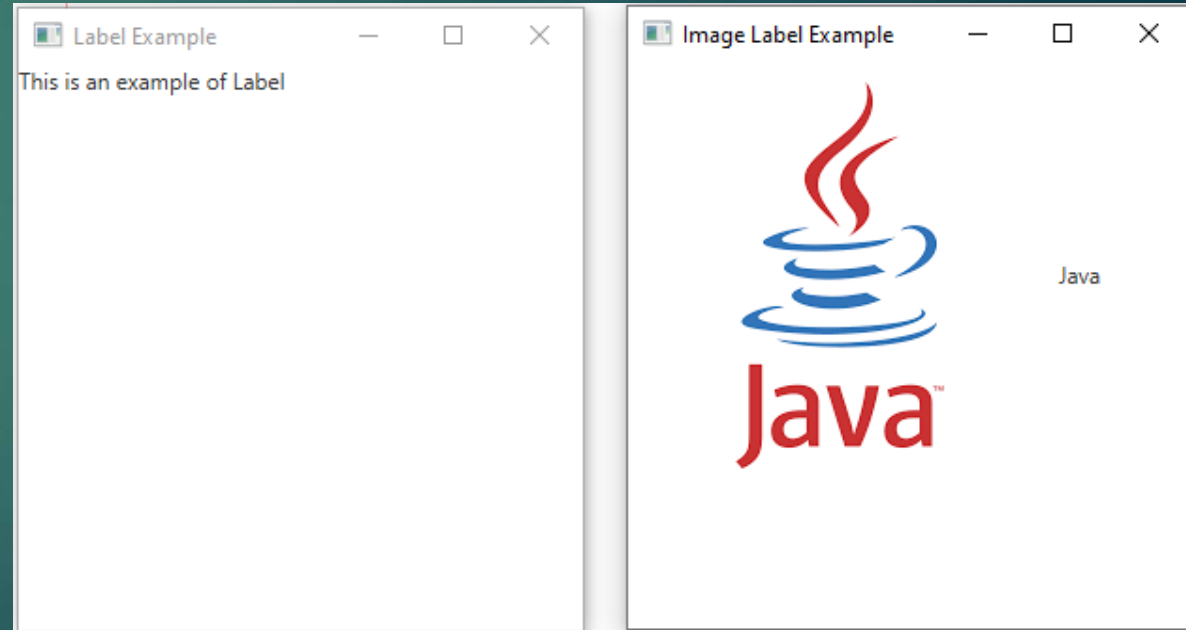
Label

- ▶ `javafx.scene.control.Label` class represents label control.
- ▶ As the name suggests, the label is the component that is used to place any text information on the screen.
- ▶ It is mainly used to describe the purpose of the other components to the user.
- ▶ You can not set a focus on the label using the Tab key.
- ▶ Package: `javafx.scene.control`
- ▶ Constructors:
 - ▶ `Label()`: creates an empty Label
 - ▶ `Label(String text)`: creates Label with the supplied text
 - ▶ `Label(String text, Node graphics)`: creates Label with the supplied text and graphics

Label

```
package javafxapplication1.UIControls;
import java.io.FileInputStream;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;
import javafx.scene.Group;
public class LabelDemo extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        // TODO Auto-generated method stub
        Label my_label=new Label("This is an example of Label");
        FileInputStream input= new
FileInputStream("C:/Users/mpstme.student/Documents/NetBeansPr
ojects/JavaFXApplication1/src/javafxapplication1/UIControls/javai
mg.png");
        Image img = new Image(input);
        ImageView imageview=new ImageView(img);
        Label my_label1=new Label("Java",imageview);
        Group root = new Group(my_label);
        Group root1 = new Group(my_label1);
        Scene scene=new Scene(root,300,300);
        Scene scene1=new Scene(root1,300,300);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Label Example");
        primaryStage.show();
```

```
// creating another stage to add another scene
    Stage newstage = new Stage();
    newstage.setScene(scene1);
    newstage.setTitle("Image Label Example");
    newstage.show();
}
public static void main(String[] args) {
    launch(args);
} }
```



Button

- ▶ JavaFX button control is represented by `javafx.scene.control.Button` class. A button is a component that can control the behaviour of the Application. An event is generated whenever the button gets clicked.
- ▶ Button can be created by instantiating Button class. Use the following line to create button object.

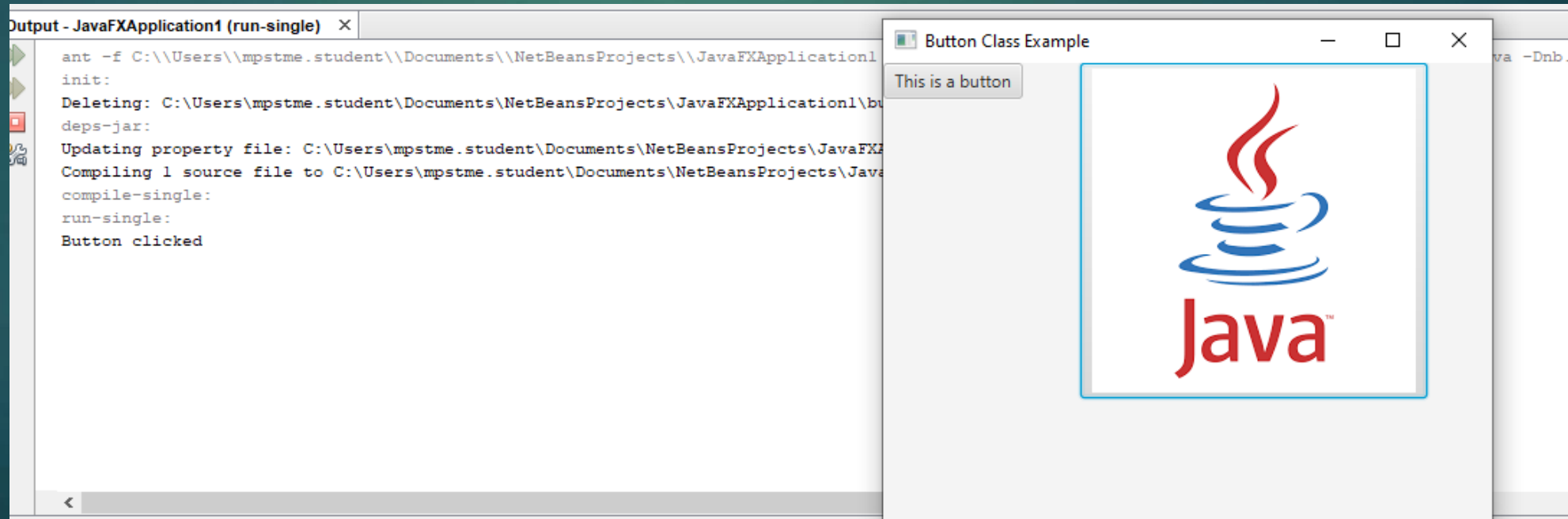
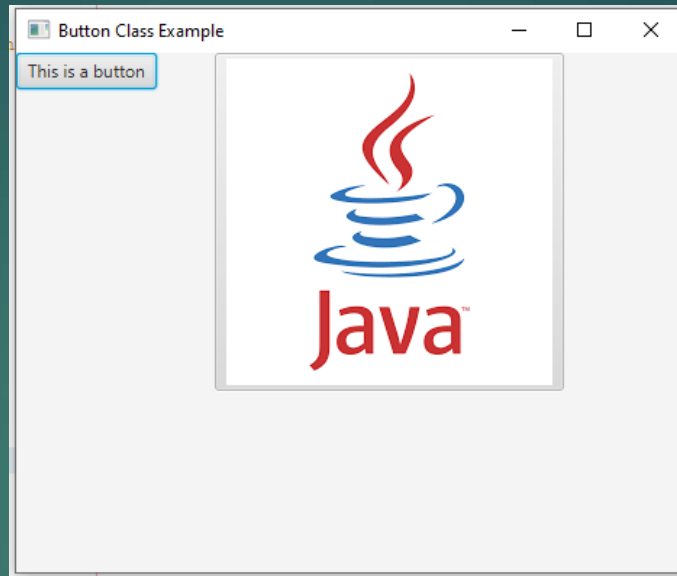
```
Button btn = new Button("My Button");
```
- ▶ **Button Action**
- ▶ Button class provides `setOnAction()` method which is used to set the action for the button click event.
- ▶ An object of the anonymous class implementing the `handle()` method, is passed in this method as a parameter.
- ▶ We can also pass lambda expressions to handle the events.

Button

```
package javafxapplication1.UIControls;
import java.io.FileInputStream;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;
import javafx.scene.layout.HBox;
public class ButtonDemo extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception
    {
        Button btn=new Button("This is a button");
        // Image Button
        FileInputStream input=new
FileInputStream("C:/Users/mpstme.student/Documents/
NetBeansProjects/JavaFXApplication1/src/javafxapplicat
ion1/UIControls/javaimg.png");
        Image image = new Image(input);
        ImageView img=new ImageView(image);
        Button btn1=new Button();
        btn1.setGraphic(img);
        //button event handling
        btn1.setOnAction(new EventHandler<ActionEvent>()
```

```
@Override
        public void handle(ActionEvent args) {
            // TODO Auto-generated method stub
            System.out.println("Button clicked");
        }
    });
    HBox root = new HBox(btn,btn1);
    root.setSpacing(40);
    Scene scene=new Scene(root,300,300);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Button Class Example");
    primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
}
}
```

Button



Radio Button

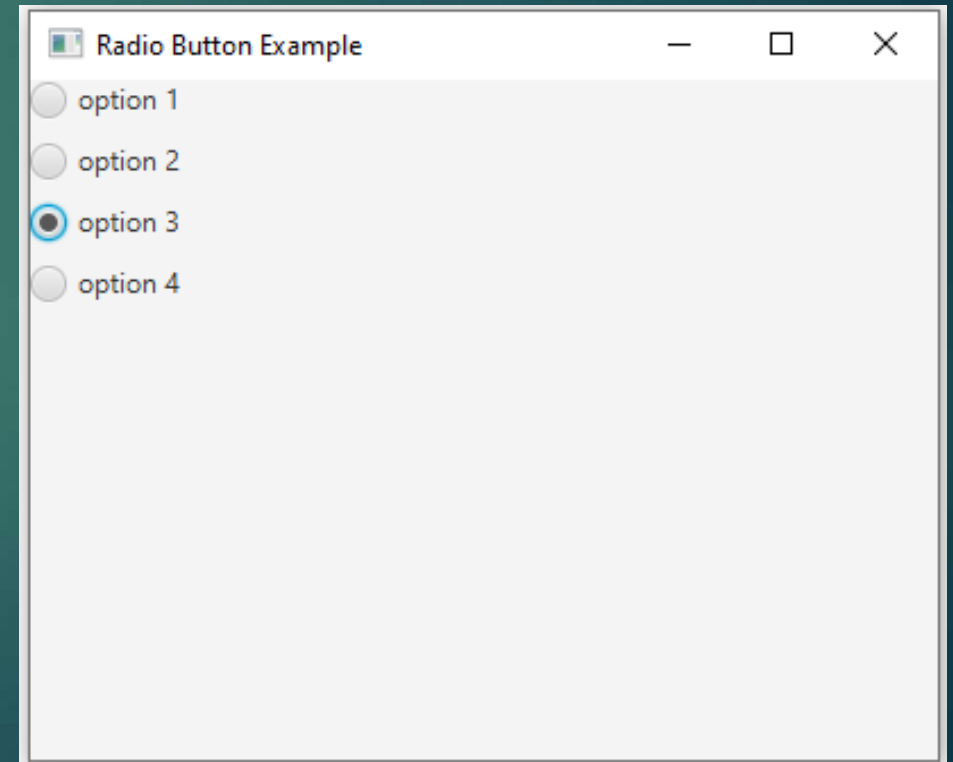
- ▶ The Radio Button is used to provide various options to the user.
- ▶ The user can only choose one option among all.
- ▶ A radio button is either selected or deselected.
- ▶ It can be used in a scenario of multiple choice questions in the quiz where only one option needs to be chosen by the student.

Radio Button

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
public class RadioButtonDemo extends Application {
@Override
public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    //ToggleGroup is used to define group of radiobuttons.
    Only one radiobutton in each group can be selected at a
    time.
```

```
    ToggleGroup group = new ToggleGroup();
    RadioButton button1 = new RadioButton("option 1");
    RadioButton button2 = new RadioButton("option 2");
    RadioButton button3 = new RadioButton("option 3");
    RadioButton button4 = new RadioButton("option 4");
    button1.setToggleGroup(group);
    button2.setToggleGroup(group);
    button3.setToggleGroup(group);
    button4.setToggleGroup(group);
    VBox root=new VBox(button1,button2,button3,button4);
    root.setSpacing(10);
```

```
    Scene scene=new Scene(root,400,300);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Radio Button Example");
    primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
} }
```

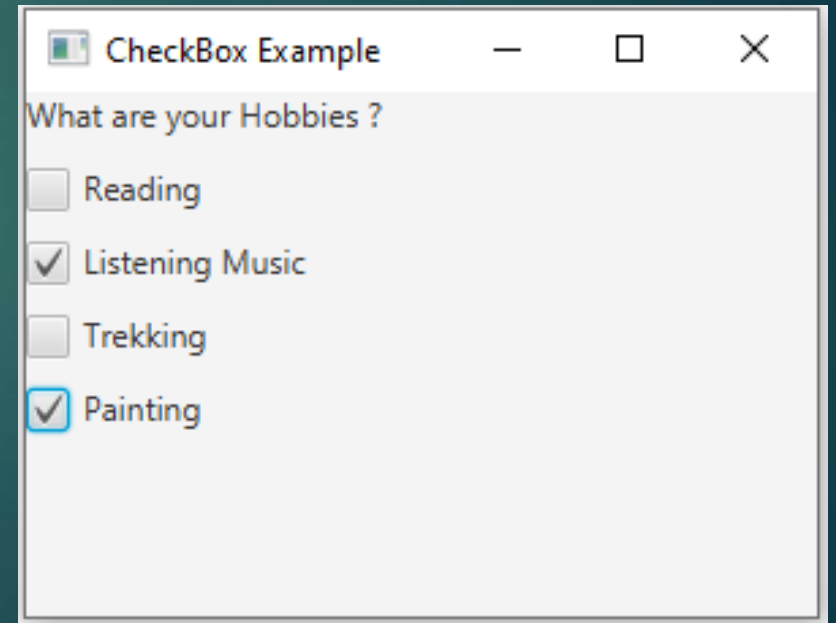


Checkbox

- ▶ The Check Box is used to provide more than one choices to the user.
- ▶ It can be used in a scenario where the user is prompted to select more than one option or the user wants to select multiple options.
- ▶ It is different from the radio button in the sense that, we can select more than one checkboxes in a scenerio.
- ▶ Instantiate `javafx.scene.control.CheckBox` class to implement `CheckBox`.
`CheckBox checkbox = new CheckBox();`
- ▶ Use the following line to attach a label with the checkbox.
`CheckBox checkbox = new CheckBox("Label Name");`
- ▶ We can change the `CheckBox` Label at any time by calling an instance method `setText("text")`.
- ▶ We can make it selected by calling `setSelected("true")`

Checkbox

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
public class CheckboxDemo extends Application {
@Override
public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    Label lbl = new Label("What are your Hobbies ?");
    CheckBox c1 = new CheckBox("Reading");
    CheckBox c2 = new CheckBox("Listening Music");
    CheckBox c3 = new CheckBox("Trekking");
    CheckBox c4 = new CheckBox("Painting");
    VBox root = new VBox(lbl,c1,c2,c3,c4);
    root.setSpacing(10);
    Scene scene=new Scene(root,800,200);
    primaryStage.setScene(scene);
    primaryStage.setTitle("CheckBox Example");
    primaryStage.show();
}
public static void main(String[] args) {
launch(args);    } }
```



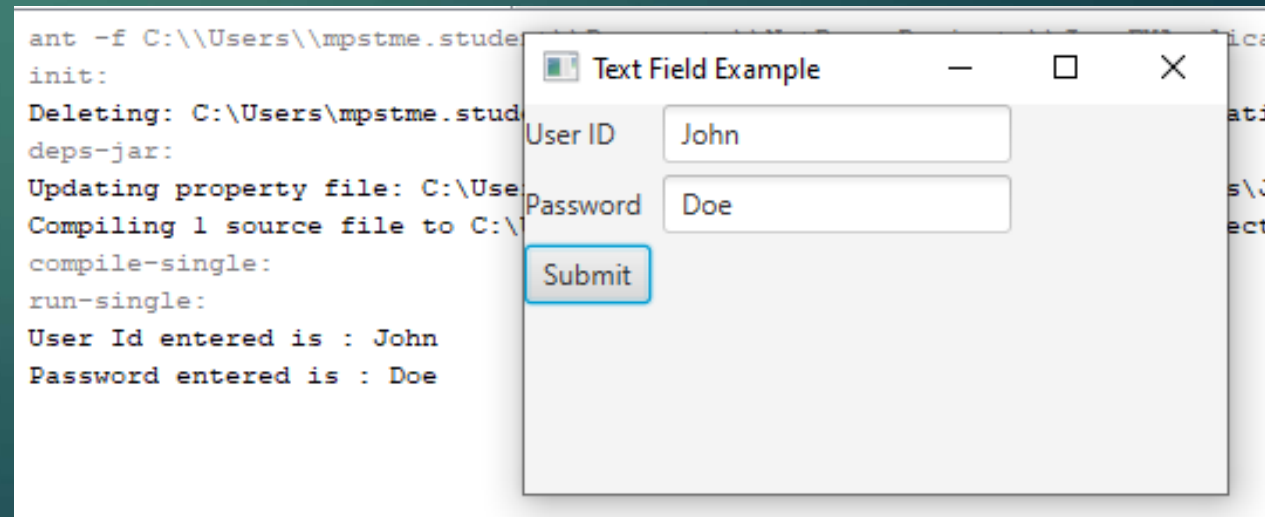
Text Field

- ▶ Text Field is basically used to get the input from the user in the form of text.
- ▶ `javafx.scene.control.TextField` represents `TextField`.
- ▶ It provides various methods to deal with textfields in JavaFX.
- ▶ `TextField` can be created by instantiating `TextField` class.
- ▶ `TextField` class provides an instance method `getText()` to retrieve the textfield data.
- ▶ It returns `String` object which can be used to save the user details in database.

Text Field

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
public class TextFieldDemo extends Application {
@Override
public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    Label user_id=new Label("User ID");
    Label password = new Label("Password");
    TextField tf1=new TextField();
    TextField tf2=new TextField();
    Button b = new Button("Submit");
    GridPane root = new GridPane();
    root.addRow(0, user_id, tf1);
    root.addRow(1, password, tf2);
    root.addRow(2, b);
    root.setHgap(5);
    root.setVgap(5);
    b.setOnAction(new EventHandler<ActionEvent>() {
```

```
@Override
    public void handle(ActionEvent args) {
        // TODO Auto-generated method stub
        System.out.println("User Id entered is : " + tf1.getText());
        System.out.println("Password entered is : " + tf2.getText());
    }
});
Scene scene=new Scene(root,300,300);
primaryStage.setScene(scene);
primaryStage.setTitle("Text Field Example");
primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
} }
```



Password Field

- ▶ Typing a password in a text field is not secure for the user.
- ▶ The Application must use a specific component to get the password from the user.
- ▶ Password Field can be created by instantiating `javafx.scene.control.PasswordField` class.
- ▶ `PasswordField` class contains a method named as `setPromptText()` for showing a prompt text to the user in password field.
- ▶ The data written in the password field is retrieved by `getText()` method.

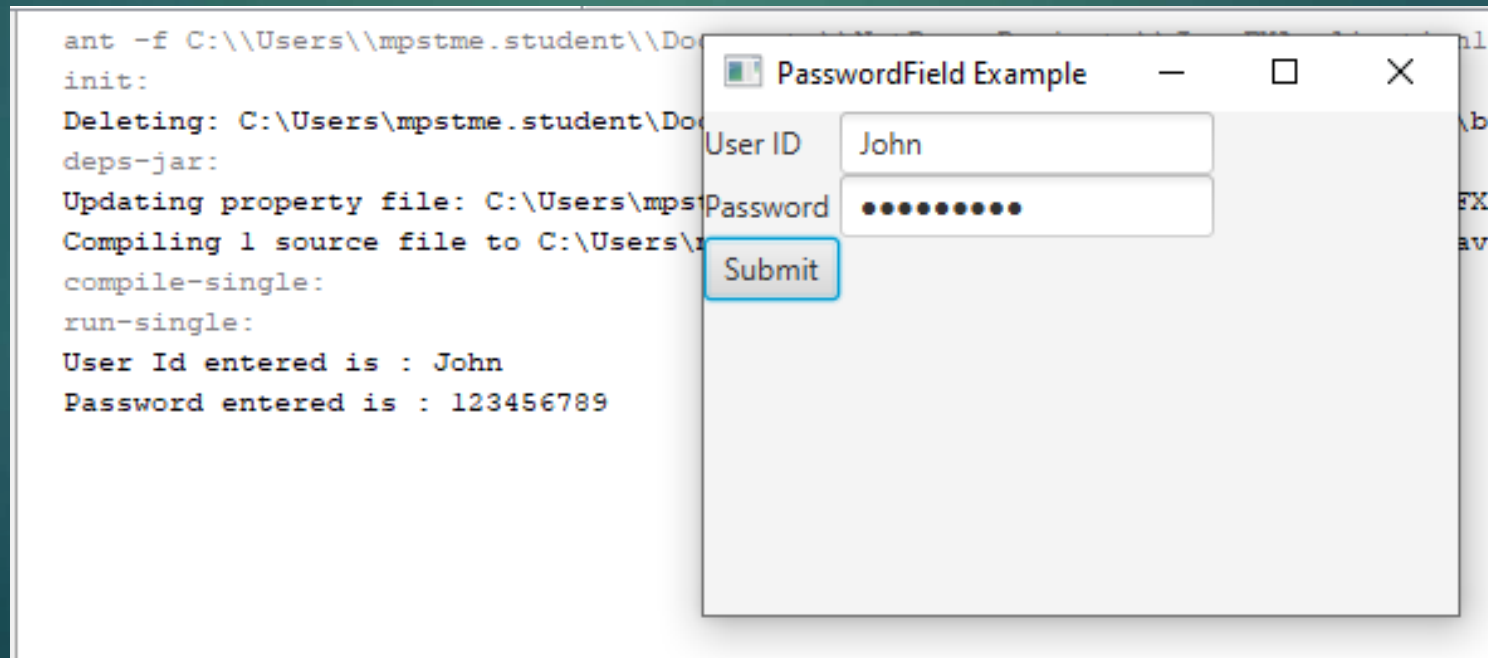
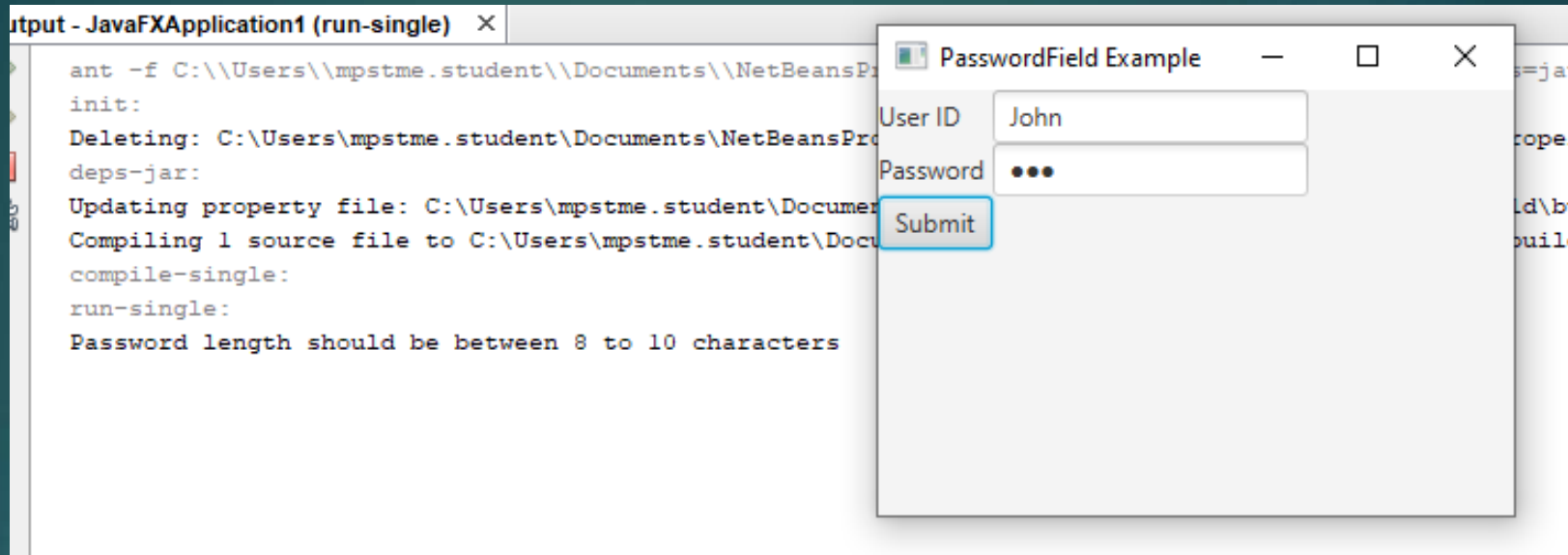
Password Field

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
public class PasswordFieldDemo extends Application {
public void start(Stage primaryStage) throws Exception {
    Label user_id=new Label("User ID");
    Label password = new Label("Password");
    TextField tf=new TextField();
    PasswordField pf=new PasswordField();
    tf.setPromptText("Enter U_Id");
    pf.setPromptText("Enter Password");
    Button b = new Button("Submit");
    GridPane root = new GridPane();
    root.addRow(0, user_id, tf);
    root.addRow(1, password, pf);
    root.addRow(5, b);
    b.setOnAction(new EventHandler<ActionEvent>() {
```

```
@Override
    public void handle(ActionEvent args) {
        String pwd=pf.getText();
        if(pwd.length()<8 || pwd.length()>10)
        {
            System.out.println("Password length should be between
8 to 10 characters");
        }
        else
        {
            System.out.println("User Id entered is : " + tf.getText());
            System.out.println("Password entered is : " + pwd);
        }
    }
});

Scene scene=new Scene(root,300,200);
primaryStage.setScene(scene);
primaryStage.setTitle("PasswordField Example");
primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
} }
```

Password Field



File Chooser

- ▶ JavaFX File chooser enables users to browse the files from the file system.
- ▶ `javafx.stage.FileChooser` class represents `FileChooser`.
- ▶ It can be created by instantiating `FileChooser` class.
- ▶ As we see in the modern day applications, there are two types of dialogues shown to the user, one is for opening the file and the other is for saving the files.
- ▶ In each case, the user needs to browse a location for the file and give the name to the file.
- ▶ The `FileChooser` class provides two types of methods,
 - ▶ `showOpenDialog()`
 - ▶ `showSaveDialog()`

File Chooser

```
package javafxapplication1.UIControls;
import java.io.File;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
public class FileChooserDemo extends Application{

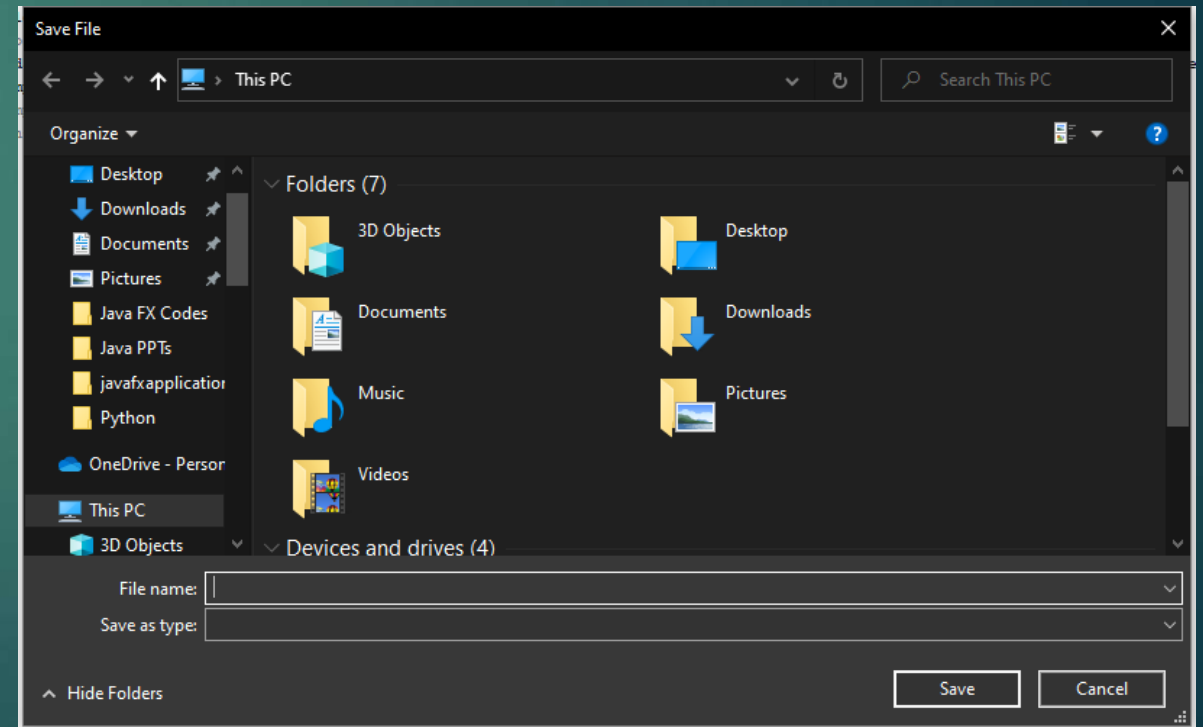
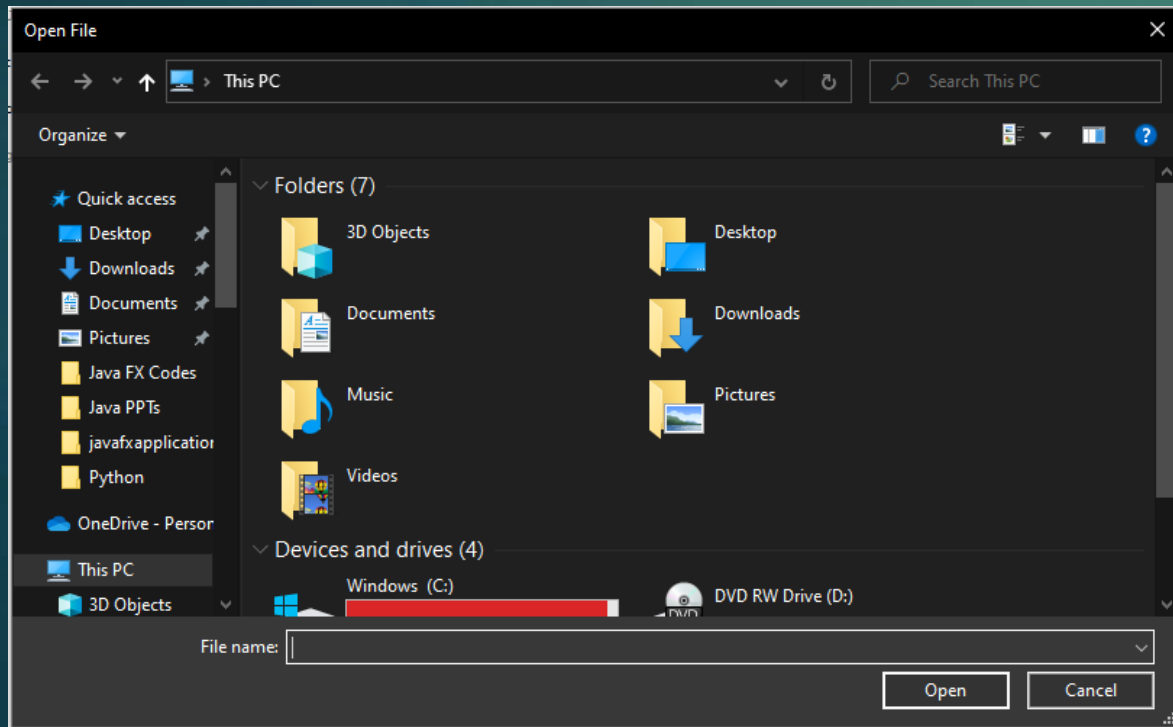
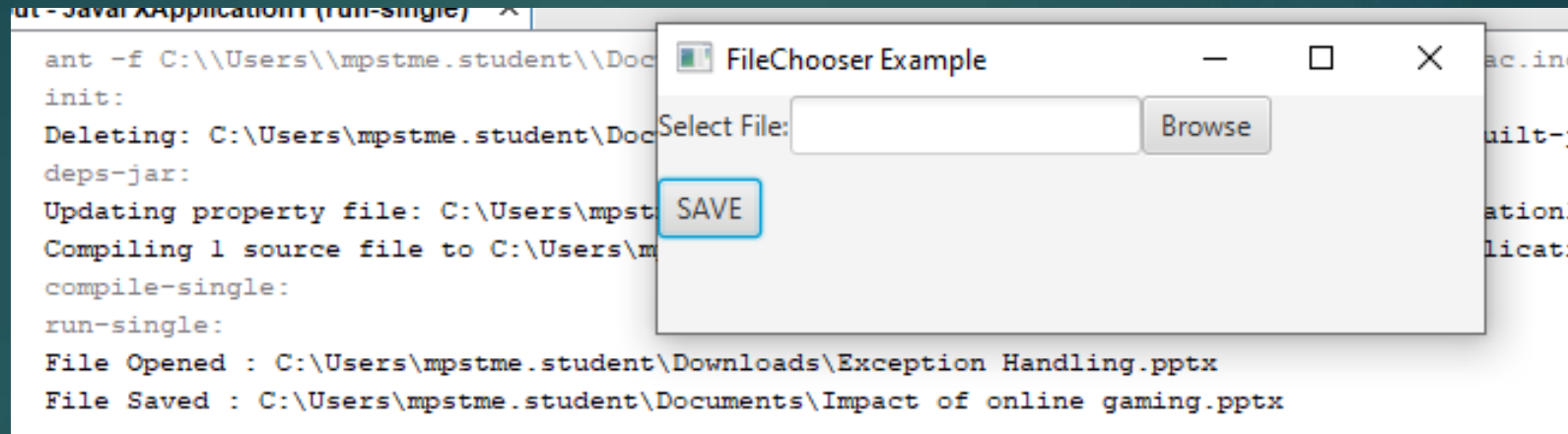
    @Override
    public void start(Stage primaryStage) throws Exception {
        // TODO Auto-generated method stub
        // showOpenDialog() Demo
        Label label = new Label("Select File:");
        TextField tf= new TextField();
        Button btn = new Button("Browse");
        btn.setOnAction(e->
        {
            FileChooser file = new FileChooser();
            file.setTitle("Open File");
            File file3=file.showOpenDialog(primaryStage);
            System.out.println("File Opened : "+file3);
        });
    }
}
```

```
// showSaveDialog() Demo
Button btn1 = new Button("SAVE");
btn1.setOnAction(e->
{
    FileChooser file1 = new FileChooser();
    file1.setTitle("Save File");
    File file2 = file1.showSaveDialog(primaryStage);
    System.out.println("File Saved : "+file2);
});

GridPane root = new GridPane();
root.addRow(0,label,tf,btn);
root.addRow(1,btn1);
root.setVgap(10);
Scene scene = new Scene(root,350,100);
primaryStage.setScene(scene);
primaryStage.setTitle("FileChooser Example");
primaryStage.show();
}

public static void main(String[] args) {
    launch(args);
} }
```

File Chooser



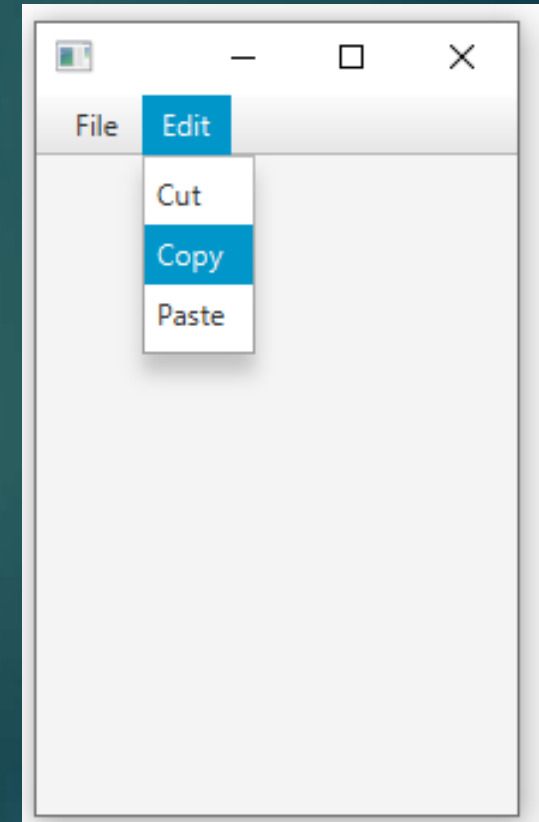
Menu

- ▶ JavaFX provides a Menu class to implement menus.
- ▶ Menu is the main component of any application.
- ▶ In JavaFX, `javafx.scene.control.Menu` class provides all the methods to deal with menus.
- ▶ This class needs to be instantiated to create a Menu.

Menu

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
public class MenuDemo extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        BorderPane root = new BorderPane();
        Scene scene = new Scene(root,200,300);
        MenuBar menubar = new MenuBar();
        Menu FileMenu = new Menu("File");
        MenuItem filemenuitem1=new MenuItem("New");
        MenuItem filemenuitem2=new MenuItem("Save");
        MenuItem filemenuitem3=new MenuItem("Exit");
        Menu EditMenu=new Menu("Edit");
        MenuItem editmenuitem1=new MenuItem("Cut");
        MenuItem editmenuitem2=new MenuItem("Copy");
        MenuItem editmenuitem3=new MenuItem("Paste");
        EditMenu.getItems().addAll(editmenuitem1,editmenuitem2,editmenuitem3);
        FileMenu.getItems().addAll(filemenuitem1,filemenuitem2,filemenuitem3);
        menubar.getMenus().addAll(FileMenu,EditMenu);
        root.setTop(menubar);
```

```
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    } }
```



Programming Practice Questions

- ▶ Create a simple GUI-based calculator using JavaFX. The calculator should perform basic operations like addition, subtraction, multiplication, and division. The GUI should include buttons for digits (0-9) and operations, as well as a display area to show the input and result.
- ▶ Create a JavaFX application with a button and a label. Each time the button is clicked, the label should update to show the current count of clicks.
- ▶ Build a simple JavaFX application with a text field and a button. When the button is clicked, display "Hello, [input text] !" in a label, where [input text] is what the user entered. Also format the text using various javafx text formatting classes and methods.
- ▶ Create a basic quiz application that presents a single multiple-choice question. When the user selects an answer and clicks a button, display whether their answer was correct or incorrect.
- ▶ Create a number guessing game where the application randomly selects a number between 1 and 100. The user should input their guess, and upon clicking a button, the app will indicate whether the guess is too high, too low, or correct.
- ▶ Design the following UI's using Java FX -

