PCA502L Java Programming

Thilagavathi M, AP(Sr.), SCORE

Module 3
JavaFX

Thilagavathi M, AP(Sr.), SCORE

What is JavaFX?

- JavaFX is a Java library used to build Rich Internet Applications. The applications written using this library can run consistently across multiple platforms.
- The applications developed using JavaFX can run on various devices such as Desktop Computers, Mobile Phones, TVs, Tablets, etc.
- To develop **GUI Applications** using Java programming language, the programmers rely on libraries such as **Advanced Windowing Toolkit** and **Swing**.
- After the advent of JavaFX, Java programmers can now develop GUI applications effectively with rich content.

Features

- Written in Java
- FXML
- Scene Builder
- Swing Interoperability
- Built-in UI Controls
- CSS like styling
- Integrated Graphics Library

Features of JavaFX

- **Written in Java** – The JavaFX library is written in Java and is available for the languages that can be executed on a JVM, which include – **Java, Groovy and JRuby**. These JavaFX applications are also platform independent.
- **FXML** – JavaFX features a language known as FXML, which is a HTML like declarative markup language. The sole purpose of this language is to define a user Interface.
- **Scene Builder** – JavaFX provides an application named Scene Builder. On integrating this application in IDE's such as Eclipse and NetBeans, the users can access a drag and drop design interface, which is used to develop FXML applications (just like Swing Drag & Drop and DreamWeaver Applications).
- **Swing Interoperability** – In a JavaFX application, you can embed Swing content using the **Swing Node** class. Similarly, you can update the existing Swing applications with JavaFX features like embedded web content and rich graphics media.
- **Built-in UI controls** – JavaFX library caters UI controls using which we can develop a full-featured application.
- **CSS like Styling** – JavaFX provides a CSS like styling. By using this, you can improve the design of your application with a simple knowledge of CSS.
- **Integrated Graphics library** – JavaFX provides classes for **2D** and **3D** graphics.

Packages of JavaFX

JavaFX provides a complete API with a rich set of classes and interfaces to build GUI applications with rich graphics. The important packages of this API are –

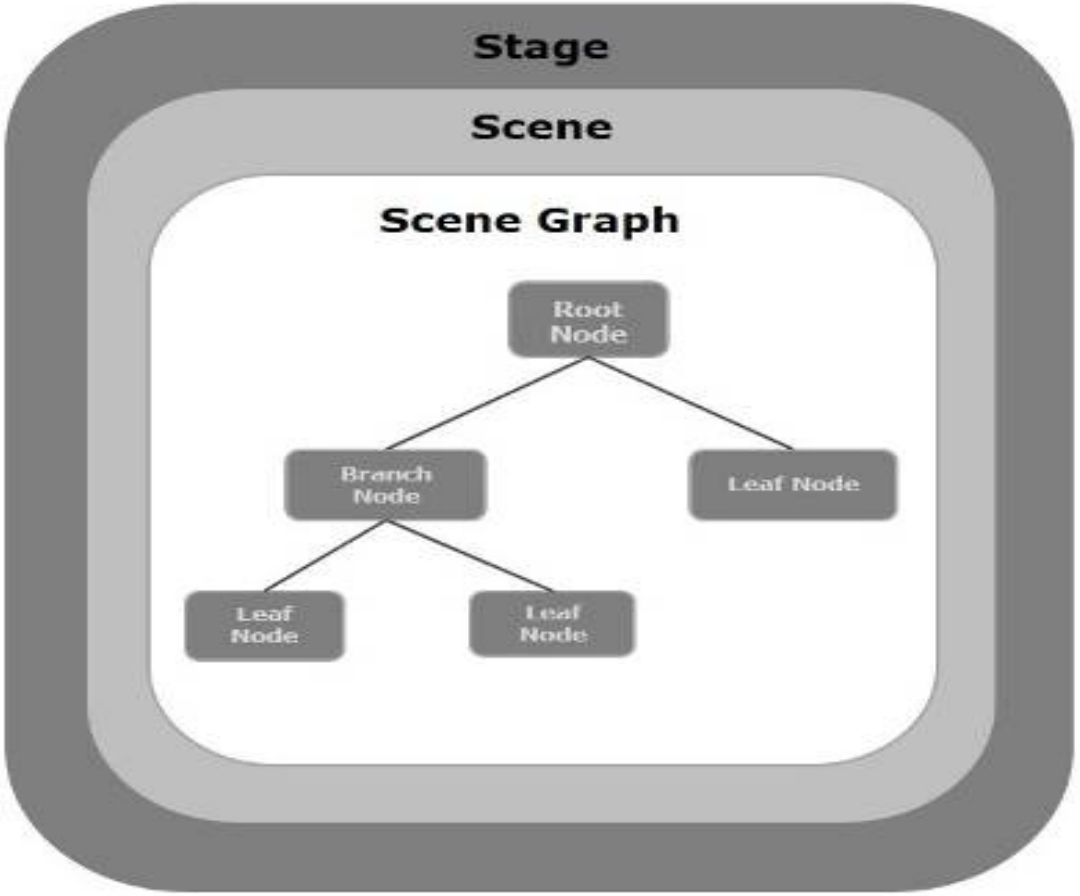
- **javafx.animation**
- **javafx.application**
- **javafx.css**
- **javafx.event**
- **javafx.geometry**
- **javafx.stage**
- **javafx.scene**

Packages of JavaFX

JavaFX provides a complete API with a rich set of classes and interfaces to build GUI applications with rich graphics. The important packages of this API are –

- **javafx.animation** – Contains classes to add transition based animations such as fill, fade, rotate, scale and translation, to the JavaFX nodes.
- **javafx.application** – Contains a set of classes responsible for the JavaFX application life cycle.
- **javafx.css** – Contains classes to add CSS–like styling to JavaFX GUI applications.
- **javafx.event** – Contains classes and interfaces to deliver and handle JavaFX events.
- **javafx.geometry** – Contains classes to define 2D objects and perform operations on them.
- **javafx.stage** – This package holds the top level container classes for JavaFX application.
- **javafx.scene** – This package provides classes and interfaces to support the scene graph. In addition, it also provides sub-packages such as canvas, chart, control, effect, image, input, layout, media, paint, shape, text, transform, web, etc. There are several components that support this rich API of JavaFX.

JAVA Application structure



Structure of the program

Thilagavathi M, AP(Sr.), SCORE

Stage

- A stage (a window) contains all the objects of a JavaFX application. It is represented by **Stage** class of the package **javafx.stage**.
- The primary stage is created by the platform itself. The created stage object is passed as an argument to the **start()** method of the **Application** class.
- The **show()** method is used to display the contents of a stage.

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. At an instance, the scene object is added to only one stage.
- we can create a scene by instantiating the Scene Class. we can opt for the size of the scene by passing its dimensions (height and width) along with the **root node** to its constructor.

- Eg.,: StackPane root = new StackPane();

```
root.getChildren().add(btn);

Scene scene = new Scene(root, 300, 250);
```

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 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.

To create a JavaFX Application

- To create a JavaFX application, you need to inherit **Application** class and implement its abstract method **start()**. In this method, you need to write the entire code for the JavaFX graphics
- In the **main** method, launch the application using the **launch()** method. This method internally calls the **start()** method of the Application class as shown in the following program.

```
public class JavafxSample extends Application {
   @Override
   public void start(Stage primaryStage) throws Exception {
      /*
      Code for JavaFX application.
      (Stage, scene, scene graph)
      */
   }
   public static void main(String args[]){
      launch(args);
   }
}
```

To create a JavaFX Application

- Within the **start()** method, in order to create a typical JavaFX application, you need to follow the steps given below –
  - Prepare a scene graph with the required nodes.
  - Prepare a Scene with the required dimensions and add the scene graph (root node of the scene graph) to it.
  - Prepare a stage and add the scene to the stage and display the contents of the stage.

First example

```
import javafx.application.Application;
import javafx.stage.Stage;
public class firstfx extends Application
{
    public void start(Stage primaryStage) {
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Life cycle of JavaFX

- The JavaFX Application class has three life cycle methods, which are
  - **start()** – JavaFX graphics code is written.
  - **stop()** – An empty method which can be overridden, here you can write the logic to stop the application.
  - **init()** – An empty method which can be overridden, but you cannot create stage or scene in this method.
- In addition to these, it provides a static method named **launch()** to launch JavaFX application.
- Since the **launch()** method is static, you need to call it from a static context (main generally). Whenever a JavaFX application is launched, the following actions will be carried out (in the same order).
  - **Init()** method is called.
  - The **start()** method is called.
  - The launcher waits for the application to finish and calls the **stop()** method.

Steps to create JAVAFX application

Step 1: Creating a Class

Create a Java class and inherit the **Application** class of the package **javafx.application** and implement the start() method of this class as follows.

```
public class Samplefx extends Application {

    public void start(Stage primaryStage) throws Exception { }

}
```

Step 2: Creating a Group Object

In the **start()** method create a group object by instantiating the class named Group, which belongs to the package **javafx.scene**, as follows.

```
Group root = new Group();
```

Step 3: Creating a Scene Object

Create a Scene by instantiating the class named **Scene** which belongs to the package **javafx.scene**. To this class, pass the Group object **(root)**, created in the previous step.

In addition to the root object, you can also pass two double parameters representing width and height of the screen along with the object of the Group class as follows.

```
Scene scene = new Scene(root, width, height);

Scene scene = new Scene(root,600, 300);
```

Step 4: Setting the Title of the Stage

You can set the title to the stage using the **setTitle()** method of the **Stage** class. The **primaryStage** is a Stage object which is passed to the start method of the scene class, as a parameter.

Using the **primaryStage** object, set the title of the scene as **Sample Application** as shown below.

```
primaryStage.setTitle("Sample Application");
```

Step 5: Adding Scene to the Stage

You can add a Scene object to the stage using the method **setScene()** of the class named **Stage**. Add the Scene object prepared in the previous steps using this method as shown below.

```
primaryStage.setScene(scene);
```

Step 6: Displaying the Contents of the Stage

Display the contents of the scene using the method named **show()** of the **Stage** class as follows.

```
primaryStage.show();
```

Step 7: Launching the Application

Launch the JavaFX application by calling the static method **launch()** of the **Application** class from the main method as follows.

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

Label using FX

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
public class labelfx extends Application
{
    public void start(Stage primaryStage)
    {
        Label label1 = new Label("JAVA FX"); //show text
        StackPane root = new StackPane(); //create a layout
        root.getChildren().add(label1); //add the Label to the layout
        Scene scene = new Scene(root,100,100); //add the StackPane to the scene and set's the width to be 100 pixels and the height to be 100 pixels
        primaryStage.setScene(scene); //set the Scene
        primaryStage.show(); //show the Stage
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Layouts

There are many different layouts.

The following are some of the more common layouts:

- BorderPane - A basic layout that gives you five areas: top, left, right, bottom, and center.
- HBox - A basic layout that orders the GUI controls in a horizontal (thus the “H") line.
- VBox - A basic layout that orders the GUI controls in a vertical (thus the “V") line.
- GridPane - A basic layout that orders the GUI controls in a grid. For example, a grid might be 2 row by 2 columns.
- StackPane - A basic layout that put all the GUI controls in a stack, in other words, right on top of each other.

Structure of the program

Thilagavathi M, AP(Sr.), SCORE

2D shapes

- Using the JavaFX library, you can draw –
  - Predefined shapes such as Line, Rectangle, Circle, Ellipse, Polygon, Polyline, Cubic Curve, Quad Curve, Arc.
- The package **javafx.scene.shape**

Line object

```
//Creating a line object
    Line line = new Line();

    //Setting the properties to a line
    line.setStartX(100.0);
    line.setStartY(150.0);
    line.setEndX(500.0);
    line.setEndY(150.0);

    //Creating a Group
    Group root = new Group(line);
```

To create a rectangle

```
Rectangle rectangle = new Rectangle();

    //Setting the properties of the rectangle
    rectangle.setX(150.0f);
    rectangle.setY(75.0f);
    rectangle.setWidth(300.0f);
    rectangle.setHeight(150.0f);

    //Creating a Group object
    Group root = new Group(rectangle);
```

Rounded Rectangle

```
//Drawing a Rectangle
    Rectangle rectangle = new Rectangle();

    //Setting the properties of the rectangle
    rectangle.setX(150.0f);
    rectangle.setY(75.0f);
    rectangle.setWidth(300.0f);
    rectangle.setHeight(150.0f);

    //Setting the height and width of the arc
    rectangle.setArcWidth(30.0);
    rectangle.setArcHeight(20.0);

    //Creating a Group object
    Group root = new Group(rectangle);
```

Drawing a Circle

```
//Drawing a Circle
    Circle circle = new Circle();

    //Setting the properties of the circle
    circle.setCenterX(300.0f);
    circle.setCenterY(135.0f);
    circle.setRadius(100.0f);

    //Creating a Group object
    Group root = new Group(circle);
```

Drawing a Polygon

- You can pass the double array as a parameter to the constructor of this class as shown below –

  Polygon polygon = new Polygon(doubleArray);

- Or, by using the getPoints() method as follows –

  polygon.getPoints().addAll(new Double[]{ List of XY coordinates separated by commas });

```
//Creating a Polygon
    Polygon polygon = new Polygon(array);

    //Adding coordinates to the polygon
    polygon.getPoints().addAll(new Double[]{
       300.0, 50.0,
       450.0, 150.0,
       300.0, 250.0,
       150.0, 150.0,
    });
```

UI Controls

- JavaFX provides several classes in the package **javafx.scene.control** to create various GUI components (controls). JavaFX supports several controls such as date picker, button text field, etc.
- Each control is represented by a class; you can create a control by instantiating its respective class.
- List of commonly used controls
  - Label
  - Button
  - ColorPicker
  - CheckBox
  - RadioButton
  - ListView
  - ChoiceList
  - TextField
  - PasswordField
  - ScrollBar
  - FileChooser
  - ProgressBar
  - Slider

UI Controls

- **JavaFX 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 gr aphics

Adding a Node to the scene graph

```
 import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
 public class LabelTest extends Application {
    public void start(Stage primaryStage) throws Exception  {
        Label my_label=new Label("This is an example of Label");
        StackPane root = new StackPane();
        root.getChildren().add(my_label);
        Scene scene=new Scene(root,300,300);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Label Class Example");
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Adding Image

- You can load and modify images using the classes provided by JavaFX in the package **javafx.scene.image**.
- JavaFX supports the image formats like **Bmp, Gif, Jpeg, Png**.
- Loading an Image - You can load an image in JavaFX by instantiating the class named **Image** of the package **javafx.scene.image**.
  - To the constructor of the class, you have to pass either of the following –
    - A File**InputStream** object of the image to be loaded or,
    - A string variable holding the URL for the image.
    - Example

```
//Passing FileInputStream object as a parameter
FileInputStream inputstream = new FileInputStream("C:\\images\\image.jpg");
Image image = new Image(inputstream);

//Loading image from URL
//Image image = new Image(new FileInputStream("url for the image));
```

Adding Image

- After loading the image, you can set the view for the image by instantiating the **ImageView** class and passing the image to its constructor as follows –

```
ImageView imageView = new ImageView(image);
```

Adding Image

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;

public class ImageExample extends Application {
  @Override
  public void start(Stage stage) throws FileNotFoundException {
    //Creating an image
    Image image = new Image(new FileInputStream("path of the image"));

    //Setting the image view
    ImageView imageView = new ImageView(image);

    //Setting the position of the image
    imageView.setX(50);
    imageView.setY(25);

    //setting the fit height and width of the image view
    imageView.setFitHeight(455);
    imageView.setFitWidth(500);

 //Setting the preserve ratio of the image view
    imageView.setPreserveRatio(true);

    //Creating a Group object
    Group root = new Group(imageView);

    //Creating a scene object
    Scene scene = new Scene(root, 600, 500);

    //Setting title to the Stage
    stage.setTitle("Loading an image");

    //Adding scene to the stage
    stage.setScene(scene);

    //Displaying the contents of the stage
    stage.show();
  }
  public static void main(String args[]) {
    launch(args);
  }
}
```

JavaFX label

- The JavaFX Label control can display a text or image label inside a JavaFX GUI.

```
Label label = new Label("My Label", imageView);

Scene scene = new Scene(label, 200, 100);
```

Creating a Button

- Package: javafx.scene.control.Button
- Constructors
  - Button b1 = new Button();
  - Button b2 = new Button("String");
- There are two ways of setting the text on the button.
  - Passing the text into the class constructor
  - By calling setText("text") method
- Wrapping Button Text
  - Btn.setWrapText(**true**);

Creating a TextField

- Package: javafx.scene.control.TextField
- Constructors
  - TextField t1 = new TextField();
  - TextField t2 = new TextField(“Default String");
- Methods
  - t1.setText("text") // set the text in the text field
  - String str = t1.getText();//retrieves the value of the text field and returns it as a String
  - t1.setVisible(boolean); // to make the control visible or to hide it
  - t1.setEditable(boolean);// to make the field editable or not
  - t1.setDisable(boolean); // to make the field unable to be clicked etc

Creating Buttons and TextField

```
Button b1=new Button("This is a button");
Button b2 = new Button("Exit");
TextField tf1=new TextField();
GridPane root = new GridPane();
root.addRow(0, tf1,b1);
root.addRow(1,b2);
Scene scene=new Scene(root,300,300);
primaryStage.setScene(scene);
primaryStage.setTitle("Button Class Example");
primaryStage.show();
```