UNIT 2 WORKING WITH UI CONTROLS

Structure

2.0 INTRODUCTION

The previous unit of this block gave you an introduction to Graphical User Interface(GUI) in Java and also introduced UI controls. The JavaFX provides a wide list of UI control elements such as label, textbox, checkbox, button and radio button. The UI (User Interface) elements are those elements that are actually shown to the user for information exchange. The package javafx.scene.control have all the necessary classes required for the UI elements. This unit explains you how to JavaFX UI controls work with CSS, FXML and event handling.

Cascading Style Sheets (CSS) are used for adding styles such as text alignments, font-size, colors, including background(s) images and page settings in HTML documents. Basically, CSS is used to control the presentation of one or more web pages in your application. This unit designates the usage of cascading style sheets with JavaFX application. You are already aware of the XML; one similar language such as FXML will be covered in this unit. You will learn how to use FXML with JavaFX application. If you talk about the events which are object and it describes a state change in a source and, when you interact with elements or nodes of an application. For example, whenever you click on the button, select an item from drop-down menu, an event generates. For handling these events in JavaFx application, you will learn four stages such as Target selection, Route construction, Event capturing and Event bubbling in this unit. In event handling mechanism, a source generates an event and sends to one or more listeners. Once the listeners received this event, processes the event and then returns.

This unit continues the study of effects, animations and media in JavaFX. This Unit explains you how to apply effects on UI elements such as Blend, DropShadow and Glow. You have seen many animated movies or images. In this unit you will use animation with user interface controls. You will also include media like audio or video in your application. For incorporating media in an application, JavaFX API provides necessary classes such as Media, MediaPlayer and MediaView. The main use of this package is media playback. This unit utmost describes concepts with examples. The previous unit already explained to you how to run JavaFX application with IDE. So, this unit is not describing this running procedure again. You can run examples of this unit whether with IDE (Eclipse/NetBeans/IntelliJ Idea, etc.) or through the command prompt.

2.1 OBJECTIVES

At the end of this unit, you will be able to:

- describe how to create CSS and how to apply this upon UI control elements in JavaFX application,
- build User Interface with FXML,
- handle events in JavaFX application,
- apply effects, animations on the UI controls, and
- incorporate audio or video media in JavaFX application.

2.2 SKIN APPLICATIONS WITH CSS

You are already aware of the Cascading Style Sheets (CSS), which are used for adding style such as fonts, colors, backgrounds and spacing between the paragraphs in HTML documents. It is used to control the presentation of one or more web pages. This section defines how to use cascading style sheets with JavaFX applications. You can use cascading style sheets to create a custom look and feel for your JavaFX application.

Before going in detail, you may recall the JavaFX application structure, which contains three keys components, namely Stage, Scene and Nodes as shown in the following figure-1. A **stage (**or a window) embraces all the objects of a JavaFX application and it is represented by the Stage class of the package **javafx.stage**. The show() method can be used to display the contents of a stage. A **scene** graph is a data structure that is a collection of nodes, and it is represented by package javafx.scene in a JavaFX application. The scene class holds all the contents of a scene graph. A **node** is a basic graphical object of a JavaFX application such as UI control objects (button, radio checkbox, etc.), 2D and 3D geometric objects (circle, rectangle, etc.),

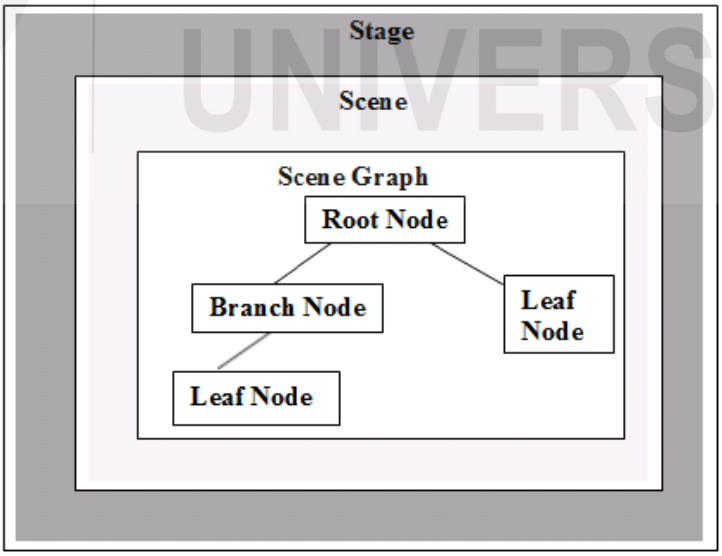

Figure 1: JavaFX application structure

Container/layout objects (Border Pane, Grid Pane, etc.), media element objects (audio, video etc.).

You can use CSS in JavaFX applications similar to use CSS in HTML web documents. JavaFX CSS are created on basis of W3C CSS specification (accessible at https://www.w3.org/Style/CSS/Overview.en.html) with some additional JavaFX features. You can refer JavaFX CSS reference guide at the link of Oracle web page (https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html).

JavaFX CSS

JavaFX provides a package javafx.css which contains the classes that are used to apply CSS in JavaFX applications. You can use CSS to enrich the look of the JavaFX application. A style sheet comprises rules that specify how formatting should be applied to the particular elements in your application. Each style rule contains two parts which are selector and declaration. The selector indicates which element(s) the declaration applies, and the declaration specifies the formatting properties of the element.

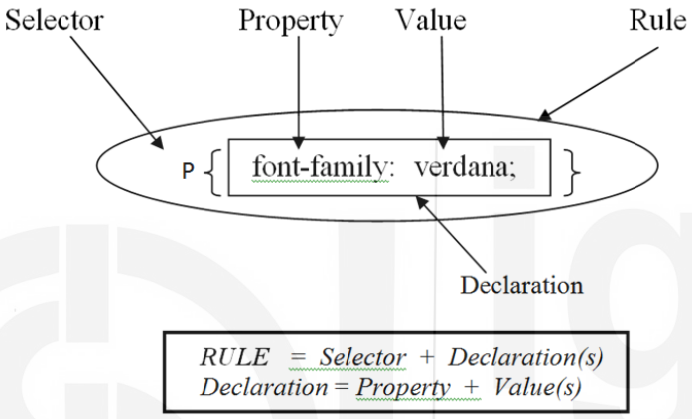

Figure 2: Format of CSS rules

JavaFX contains the new generation UI library, which is used to configure the theme of the application.

The JavaFX 8 application holds caspian.css as the default style sheet found in the JavaFX runtime JAR file (jfxrt.jar). The latest version of JavaFX application uses Medona.css as a style sheet that defines styles for the root node and the UI controls. There are several ways to apply a CSS style in a JavaFX UI controls such as JavaFX default CSS style sheet, Scene specific, Parent specific and Element specific style property. If you do not create your own style sheet, then JavaFX provides a default CSS style sheet that may be applicable to all JavaFX elements.

Creating Style Sheets

You can generate your own cascading style sheets which override the styles in the default style sheet. You can create style sheet(s) with .css extension, and it is placed in the same directory of the main class for your JavaFX application. You can create your external Style Sheet or inline Style Sheet for the JavaFX applications.

When you design either your external Style Sheet or inline Style Sheet, you may follow the rules for the selection of property name as :

- You must define "-fx-" as a prefix with all JavaFX property names.
- If more than one word exists in the property name, then use the hyphen (-) to separate them.
- Property name and their value are separated by colon (:)

... If more rules are defined for the Property name, then they are separated by a semicolon (;).

CSS class Selector

You can write a dot (.) character followed by the Style class selector name. It is used to identify more than one element in an application. In the following example, the font property name is preceded by prefix -fx- and Property name and their value are separated by colon, and rules are separated by semicolon for the JavaFX label element.

```
.label
{  -fx-font : bold 12pt "arial";   -fx-padding:10     }
```

For accessing the above-mentioned style class which are appropriate to label node, you can use the getStyleClass().add() method.

ID Styles

JavaFX provides us facility to create the style for the individual node. The style name can be given as the ID name preceded by the hash(#) symbol to select one unique element.

```
#submit
{
   -fx-font : bold 14pt "arial";
   -fx-background-color : #87ceeb;
}
```

You can use the above-mentioned style for the individual Submit button like the following way:

```
Button Submit=new Button ("Submit");   //submit button
Submit.setId("submit");    //set ID for the submit button
```

Create your own Style Sheet

You can add your own external style sheet to a scene in JavaFX application like the following:

```
Scene sc = new Scene(root, 400, 200);
sc.getStylesheets().add("path/name_of_ownstylesheet.css");
```

If you want to set a CSS style sheet on a parent layout element then style sheet will be applied to all elements inside that layout element or child element. It is similar to setting style sheet on a Scene object. You can add your own external style sheet to a Parent element in JavaFX application like the following:

```
Button b1 = new Button("Submit");
Button b2 = new Button("Reset");
VBox vbox = new VBox(b1, b2);
vbox.getStylesheets().add("button_specfic_styles.css");
```

Following example illustrate you how to create an own CSS for JavaFX application. This example displays Grade Card page which consists of controls such as label, text field and submit button. This example contains two files: OwnCSSExample.java and Examplestyle.css. The source code for both files is listed below:

Example-1: Source code for OwnCSSExample.java

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.text.*;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
public class OwnCSSExample extends Application
{
  public void start(Stage stage) throws Exception
  {
    Label enroll=new Label("Enroll No.");    //label creation for Enroll
    Label prg=new Label("Programme");  //label creation for programme

    //Creating Text Field for Enroll and programme
    TextField txtf1=new TextField();
    TextField txtf2=new TextField();

    Button Submit=new Button ("Submit");   //submit button
    Submit.setId("submit");    //set ID for the submit button

    GridPane gpane=new GridPane();    //creating grid pane

    //setting horizontal and vertical gaps between the rows
    gpane.setHgap(5);
    gpane.setVgap(5);

    Scene scene = new Scene(gpane,400,200);
    //adding the the nodes to the GridPane's rows
    gpane.addRow(0, enroll,txtf1);
    gpane.addRow(1, prg,txtf2);
    gpane.addRow(2, Submit);

    gpane.getStylesheets().add("Examplestyle.css"); //add CSS file to the grid pane
    stage.setTitle("Own CSS Example");  //set title to the Stage
    stage.setScene(scene);//set scene to the stage
    stage.show();  //showing contents of the stage
  }
  public static void main(String[] args)
  {
    launch(args);
  }
}
```

Source code for Examplestyle.css

```
.label
{
-fx-font : bold 12pt "arial";
-fx-padding:10
}

#submit
{
-fx-font : bold 14pt "arial";
-fx-background-color : #87ceeb;
```

```
-fx-text-fill:red;
-fx-padding: 8px 8px
}
```

Now, you can compile and execute the OwnCSSExample.java file from the command prompt by using the following commands:

```
javac OwnCSSExample.java
java OwnCSSExample
```

The output of the above example is shown in figure-3 like the following:

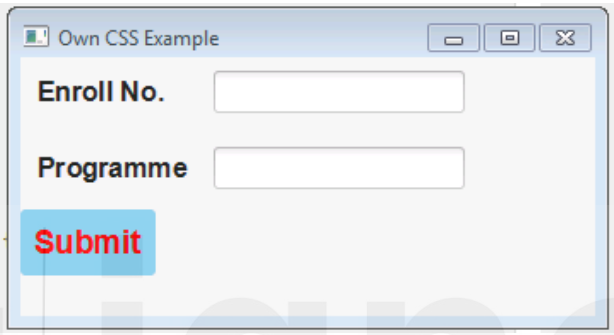

Figure 3: Output Screen for CSS creation example

Create Inline Style Sheets

JavaFX allows us to describe the style rules in the JavaFX application code itself. You can insert inline styles in JavaFX applications by using setStyle() method. The inline style provides the finest level of control. These styles are applicable only for those nodes on which they are set. The following example illustrates you how to create an inline style sheet to a button.

```
Button b1 = new Button("Submit");
b1.setStyle("-fx-background-color:yellow; -fx-text-fill: white;");
```

The following example illustrates inline style sheet rules in the source code file for some UI controls in JavaFX such as text field, password field and button. This example contains only one java file namely CssInStyleExample.java. The source code is listed below:

Example-2: Source code for CssInStyleExample.java

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.*;
import javafx.scene.text.*;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;

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

```
}
public void start(Stage stage)
{
 Text txt1 = new Text("Name");     // label for Student Name
 Text txt2 = new Text("Password");     // label for Password
 Text txt3 = new Text("Programme");  // label for Programme

 // Text Field for Student
 TextField txtField1 = new TextField();

 // Text Field for password
 PasswordField txtField2 = new PasswordField();

 // Text Field for programme
 TextField txtField3 = new TextField();

 //Creating Buttons
 Button bt1 = new Button("Submit");
 Button bt2 = new Button("Reset");

 //Creating a Grid Pane
 GridPane gPane = new GridPane();

 //Set size for the pane
 gPane.setMinSize(400, 200);

 //Set the padding
 gPane.setPadding(new Insets(10, 10, 10, 10));

 //Set the vertical and horizontal gaps between the columns
 gPane.setVgap(5);
 gPane.setHgap(5);

 //Set the Grid alignment
 gPane.setAlignment(Pos.CENTER);

 //arrange all the nodes in the grid
 gPane.add(txt1, 0, 0);
 gPane.add(txtField1, 1, 0);
 gPane.add(txt2, 0, 1);
 gPane.add(txtField2, 1, 1);
 gPane.add(txt3, 0, 2);
 gPane.add(txtField3, 1, 2);
 gPane.add(bt1, 0, 3);
 gPane.add(bt2, 1, 3);

 //inline style for nodes
 bt1.setStyle("-fx-background-color:blue; -fx-text-fill: white;");
 bt2.setStyle("-fx-background-color: blue; -fx-text-fill: white;");
 txt1.setStyle("-fx-font: normal 15px 'arial' ");
 txt2.setStyle("-fx-font: normal 15px 'arial' ");
 txt3.setStyle("-fx-font: normal 15px 'arial' ");
 gPane.setStyle("-fx-background-color: lightblue;");

 // creating a scene object
 Scene scene = new Scene(gPane);
```

```
        // Set Stage title
        stage.setTitle("CSS InStyle Example");

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

        //showing the contents of the stage
        stage.show();
    }
}
```

Now, you can compile and execute the CssInStyleExample.java file from the command prompt by using the following commands.

javac CssInStyleExample.java

java CssInStyleExample

The output of the above example is shown in following figure-4:

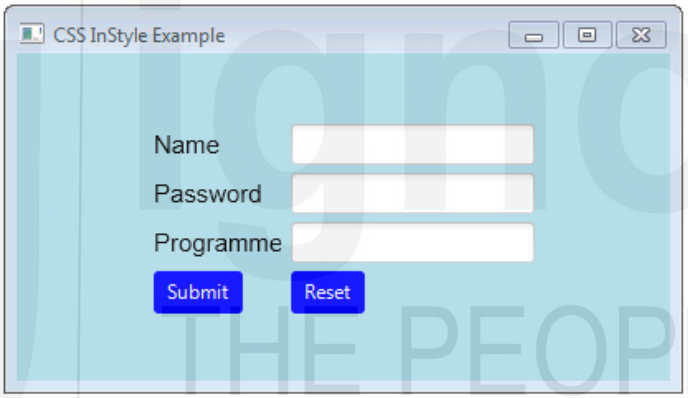

Figure 4: Output Screen for inline CSS creation example

2.3 BUILD UI WITH FXML

In the previous section, you have learned about the creation of cascading style sheets (CSS) and linking with JavaFX UI controls. This section describes to you how to create JavaFX user interfaces using FXML.

FXML is a XML-based user interface markup language for defining the user interface of a JavaFX application. It is created by Oracle Corporation and is another form of designing user interfaces by means of procedural code. Using FXML, you can write the UI controls code separate from the application logic.

JavaFX FXML Example

You can write very simplest way to JavaFX FXML. Here is a FXML example that comprises a simple JavaFX Graphical User Interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Label?>

<VBox>
```

```
    <children>
        <Label text="Welcome Students in the FXML world "/>
    </children>
</VBox>
```

The first line in the FXML document is similar to standard first line of XML documents. The next two lines are import statements which you want to import the classes to use in FXML. After the import statements you shall write the actual structure of the GUI. This example defines a VBox JavaFX layout component which contains a single Label as child element. The Label is used to display a text in the Graphical User Interface.

For the understanding of the concept of FXML through the below example which comprises three files:

- ApplicationFXML.java - It is the main JavaFX Application class which loads the FXML document using FXMLLoader.
- ControllerFXML.java – This file contains event handler code.
- FXMLfile.fxml - This FXML Document contains user interface for the application

Example - 3: Source code for ApplicationFXML.java

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class ApplicationFXML extends Application
{
    public void start(Stage stage) throws Exception
    {
        Parent root = FXMLLoader.load(getClass().getResource("FXMLfile.fxml"));
        Scene sc = new Scene(root);
        stage.setScene(sc);

        // Set the Stage Title
        stage.setTitle("FXML Example");
        stage.show();
    }
    public static void main(String[] args)
    {
        launch(args);
    }
}
```

Source code for FXMLfile.fxml

```
<?xml version="1.0" encoding="UTF-8"?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
```

```
<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="ControllerFXML">

<children>
      <Button layoutX="126" layoutY="90" text="Click here!"
                onAction="#ButtonEvent" fx:id="button" />
      <Label layoutX="126" layoutY="120" minHeight="16" minWidth="69"
              fx:id="label" />
 </children>
</AnchorPane>
```

Source code for ControllerFXML.java

```
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Label;

public class ControllerFXML
{
  @FXML
   private Label label;
   public void initialize() {   }
  @FXML
   private void ButtonEvent(ActionEvent event)
   {
     label.setText("Welcome in SOCIS!");
   }
}
```

Now, you can compile both java files. When you run this project, it shall show a small window with a "Click here" button. After clicking on the button, it will show a welcome message.

When your main application class named as ApplicationFXML will run, the FXMLLoader will load FXMLfile.fxml. During loading FXMLfile.fxml, loader will find the name of the controller class which is specified by fx:controller="ControllerFXML" in the FXML file. Then it will call the controller's initialize method, and the code that is registered as an action handler with the button will be executed.

☞ Check Your Progress-1

1. What is JavaFX CSS? What are different ways to apply styles to a javaFX application? Explain with examples.

.....

.....

.....

.....

2. Explain the differences between CSS class Selector and ID styles.

.....

.....

.....

.....

3. In JavaFX application, a key concept is the scene graph. Explain it with diagram.

.....

.....

.....

.....

4. What is FXML? How to use FXML in JavaFX application, explain it with example.

.....

.....

.....

.....

2.4 EVENT HANDLING IN JAVAFX

In the previous section 2.2, you have used UI control elements with CSS in JavaFX applications. This section describes events and the handling of events for these UI elements.

JavaFX API facilitates us to develop GUI based applications. Whenever you interact with such applications (nodes), an event is generated. For example, when a user clicks on a button, selects an item from a menu, then an event generates. You can say that events are notifications. In JavaFX, events are principally used to notify the application about the actions taken by the user. When you write GUI applications, then the program requires event handling mechanism for controlling the user's interaction with the GUI.

JavaFX provides a javafx.event package, which makes available for the basic framework of FX events. The Event class works as the base class for JavaFX events. Each event is associated with source, event target or listener and event type. In event handling mechanism, a source generates an event and sends it to one or more listeners. Once the listeners received this event, they process the event and then returns.

Event source is an object (e.g., a Button object) that generates an event (e.g., MouseEvent). Event is generated when the changes occurred in the internal state of that object. The source may create various types of events. The origin of the event implements the EventTarget interface.

Listener or targets object that listens for a particular type of event which is generated by the source. For the communication between listener and source, the listener must have been connected with the source and must implement methods to receive and process an interface like EventHandler.

There are various **events types** in JavaFX such as ActionEvent (e.g. button is pressed), MouseEvent (e.g. mouse activity), KeyEvent (e.g. keystroke has occurred), ScrollEvent (e.g. scrolling by mouse wheel) etc. You can define your own event by inheriting the class javafx.event.Event. These events are mainly categorised into two groups as Foreground Events and Background Events. **Foreground Events** are based on a user's direct interaction (e.g. button is pressed), while **Background Events** require end-user interaction such as hardware or software failure, timer expiry, etc. For more details, you can refer to the https://docs.oracle.com/javase/8/javafx/events-tutorial/processing.htm

Stages of Event Handling

When a source generates an event, the JavaFX application goes through the four stages (Target selection, Route construction, Event capturing, and Event bubbling) for event handling. Before knowing the stages of event handling, you should know two important terms; Event Filters and Event Handlers.

Event Filter

Event filter covers application logic to process an event. It can be used to process the events such as mouse actions, keyboard actions, scroll actions and other user interactions which are generated by your application. Event filters allow you to handle an event during the event capturing stage of event handling process. During the event capturing stage, a node must be registered with an event filter to process an event.

Event Handler

An event handler is a function or method executed in response to an event, such as keystrokes and button clicks. It is used during the bubbling stage of event handling process. Node which is participated in the dispatch chain path can register to more than one filter/handler. You can define a common filter/handler to the parent, which is treated as a default for all the child nodes.

Following are stages of event handling:

Target selection

When any user interacts with UI controls, then an action occurs. At that moment, the system decides which node gets focus. For example, you click somewhere on the screen for the mouse event, the target node is found at the location of the cursor. If more than one node is found at the cursor point, then the uppermost node will be considered as the target.

Route construction

Whenever an event is generated, an event dispatch route is created in order to handle the events.

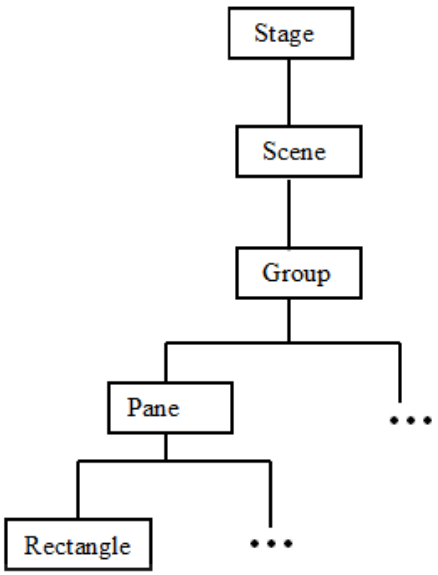

Figure 5: Event Dispatch Route

This dispatch route comprises the path from the stage to the node on which the event is generated. The Event route is regulated by the event dispatch chain which was formed in the operation of the buildEventDispatchChain() method of the selected event target.

Event capturing

Once the event dispatch route is created, the event is transmitted from the source node. The generated event is traversed all the nodes in the route from top to bottom. If the event filter is registered with any of these nodes which are in the dispatch route, then event will be executed otherwise, it will be transferred to the target node. The target node processes the event in that case. You can see in event dispatch route, which are shown in Figure 5, the event moves from the Stage node to the Rectangle node during the event capturing phase.

Event bubbling

When the event is captured and processed by the target node or registered filter, the event starts navigation all the nodes again from the bottom to the root node. If any of the node in the despatch chain has handler which is registered for the type of event happened then that handler is called, and it will get executed otherwise the process is returned to the next node up in the route and repeat the process. If a handler does not consume the event then the root node ultimately receives such event, and processing is finished.

You will find the example on event handling with animation feature in the next section.

2.5 EFFECTS, ANIMATION AND MEDIA

In the previous section, you have known about the event handling in javaFX application. This section describes you how to give effects and animations on the UI controls as well as about the integration of media in your application.

Effects in JavaFX

You may be aware of the creation of the effects on graphics which are any action or changes that heightens the presence of the graphics in an application. The meaning of effects is functionally similar to the JavaFX application, wherein it is an algorithm that applies on nodes to visually enrich their appearance. The JavaFX API provides effect property of the **Node** class to specify the effect which is used to set various effects such as **Blend, Bloom, BoxBlur, ColorAdjust, ColorInput, DropShadow, GaussianBlur, Glow, ImageInput, InnerShadow, Lighting, MotionBlur, PerspectiveTransform, Reflection, SepiaTone, and Shadow** to a node. Each of these effects is denoted by a class, and all these classes are accessible by a package named **javafx.scene.effect.Effect**. This section describes you the effects of using two DropShadow and Reflection with examples. Here is not possible to define each effect with examples.

Apply Effect to JavaFX Node

You can apply effects on any JavaFX UI control using setEffect() method, which needs to be called through a node object. The following example is showing DropShadow effect on Rectangle shape. Here, color code (FFFF00) representing the yellow color. JavaFX provides **javafx.scene.effect.DropShadow** class is used for providing dropshadow effect on UI controls. The parameters of DropShadow class are Radius, X Offset, Y Offset and Color like the following. These parameters are optional.

```
Rectangle.setEffect(new DropShadow(1, 20, 30, Color.web("#FFFF00")));
```

DropShadow Effects

The following example-4 demonstrates you how to apply DropShadow effects to a JavaFX UI control. For this you can create an application which has a Rectangle shape and put DropShadow effect on the shape. The source code is listed below:

Example- 4 : Source code for EffectsJavaFX.java

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.effect.DropShadow;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
public class EffectsJavaFX extends Application
{
  public static void main(String[] args)
  {
    launch(args);
  }
  public void start(Stage stage)
  {
    Rectangle r = new Rectangle(100,100, Color.RED);
    r.setEffect(new DropShadow(1, 20, 30, Color.web("#FFFF00")));
    Scene scene = new Scene(new Pane(r), 300, 250);
    stage.setScene(scene);
    stage.setTitle("Effects Example");
    stage.centerOnScreen();
    stage.show();
  }
}
```

Now, you can compile and execute the above program from the command prompt. The output of the program is display as shown in figure-6:

Figure 6: Example for DropShadow Effects

Reflection Effects

The JavaFX provides mirror-like reflection effect using javafx.scene.effect.Reflection class. It takes the parameters such as topOffset, topOpacity, bottomOpacity, fraction, which affect resulting reflection to a JavaFX Node.

The following example-5 demonstrates you how to apply Reflection effects to a JavaFX UI control.

Example-5: Source code for ReflectionEffects.java

```
import javafx.application.Application;
import javafx.geometry.VPos;
import javafx.scene.Scene;
import javafx.scene.effect.Reflection;
import javafx.scene.layout.Pane;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;
public class ReflectionEffects extends Application
{
  public static void main(String[] args)
  {
    launch(args);
  }
  public void start(Stage stage)
  {
    Text txt = new Text("Working with UI Controls");
    txt.setLayoutX(30);
    txt.setLayoutY(20);
    txt.setTextOrigin(VPos.TOP);
    txt.setFont(Font.font("Arial", FontWeight.BOLD, 40));
    Reflection ref = new Reflection();
    ref.setTopOffset(0);
    ref.setTopOpacity(0.75);
    ref.setBottomOpacity(0.10);
    ref.setFraction(0.7);
    txt.setEffect(ref);
    Scene scene = new Scene(new Pane(txt), 425, 175);
```

```
        stage.setScene(scene);
        stage.setTitle("Reflection Effects Example");
        stage.show();
    }
}
```

Now, you can compile and execute the above program from the command prompt. The output of the program is display as shown in figure-7:

Figure 7: Example for Reflection Effects

Animation in JavaFX

You have seen many animated graphics which have shown the illusion of its motion using the rapid display. You can create an animated node in JavaFX application by changing its property over time. You can apply animations or transitions such as **Fade Transition**, **Fill Transition**, **Rotate Transition**, **Scale Transition**, **Stroke Transition**, **Translate Transition**, **Pause Transition**, **Parallel Transition, Path Transition**, **Sequential Transition** etc. All these transitions are represented by individual classes in the package **javafx.animation**.

Translate transition is used to create movement/transition from one point to another point within a defined duration. It is done by keep changing the translateX and translateY properties of the node at the regular interval. In JavaFX, this animation is represented by the class javafx.animation.TranslateTransition. The example-6 is showing Translate transition feature of animation. Using **Rotate transition** feature**,** you can rotate an object. You can set rotation using 'toAngle' and 'byAngle' method. The 'toAngle' indicates what angle the node should rotate, and 'byAngle' indicates how much it should rotate from the current angle of rotation. For more details and example, you can refer to the Oracle JavaFX document at the link (https://docs.oracle.com/javafx/2/animations/basics.htm).

To incorporate animations in JavaFX application, you may follow the below steps

- Create a node using their class.
- Instantiate the animation class that is to be used
- Set the properties of the animation
- To complete this application, play the animation using the play() method of the Animation class.

The following example illustrates you how to apply '**Translate Transition**' animation on a Rectangle node with event handling. For this you can create an application which has a Rectangle shape and two Play and Pause buttons for change the running position of the shape. The source code is listed below:

Example-6: Source code for AnimationNEventHandler.java

```
import javafx.animation.TranslateTransition;
```

```
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.*;
import javafx.scene.control.Button;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;
public class AnimationNEventHandler extends Application
{
  public static void main(String[] args)
  {
    launch(args);
  }
  public void start(Stage stage) throws Exception
  {

    Rectangle r = new Rectangle(100,100);  //Create Rectangle
    r.setFill(Color.BLUE);  //set the color of  Rectangle

    //create play button and set coordinates
    Button btn1 = new Button("Play");
    btn1.setTranslateX(100);
    btn1.setTranslateY(150);

    // creating pause button and set coordinate
    Button btn2 = new Button("Pause");
    btn2.setTranslateX(145);
    btn2.setTranslateY(150);

    //Instantiating TranslateTransition class to create animation
    TranslateTransition trans = new TranslateTransition();

    //set attributes for the TranslateTransition
    trans.setAutoReverse(true);
    trans.setByX(200);
    trans.setCycleCount(100);
    trans.setDuration(Duration.millis(600));
    trans.setNode(r);

    //Create EventHandler
    EventHandler<MouseEvent> handler = new EventHandler<MouseEvent>()
    {
      public void handle(MouseEvent event)
      {
        if(event.getSource()==btn1)
        {
          trans.play(); //animation will play when click on the play button
        }
        if(event.getSource()==btn2)
        {
          trans.pause(); //animation will pause when click on the pause button
        }
        event.consume();
      }
```

```
        };

        //Add Handler for the play and pause button
        btn1.setOnMouseClicked(handler);
        btn2.setOnMouseClicked(handler);

        //Create Group and scene
        Group root = new Group();
        root.getChildren().addAll(r,btn1,btn2);
        Scene scene = new Scene(root,200,100,Color.TRANSPARENT);
        stage.setScene(scene);
        stage.setTitle("Animation with Event Handling");
        stage.centerOnScreen();
        stage.show();
    }
}
```

Now, you can compile and execute the above program from the command prompt. The output of the program is display as shown in figure-8

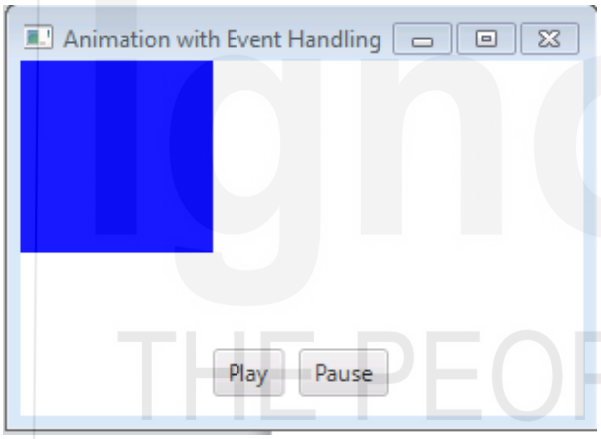

Figure 8: Output screen for the example of event handling with animation

Media in JavaFX

You have seen many rich internet applications (RIA) on the internet or even used such applications like Google Maps and Google Docs. JavaFX provides media API that enables you to integrate audio and video into rich internet applications. The package **javafx.scene.media** covers all the essential classes for making an attractive, interactive JavaFX application. It consists of three prime classes such as **Media, MediaPlayer** and **MediaView**. The package is used for media playback.

The Media class denotes a media resource which could be an audio or a video, whereas MediaPlayer class is used for playing media. The MediaPlayer class is used in association with the Media and MediaView classes to display and control media playback. MediaPlayer provides media-controlling methods such as pause(), play(), stop() and seek() which apply to all types of media. MediaView arranges a view of Media being played by a MediaPlayer.

It is very easy to incorporate video or audio in JavaFX application. For integrating media in JavaFX, you can use the following steps:

- You can instantiate **javafx.scene.media.Media** class by passing the path of the video or audio file in its constructor like the following:

```
Media media = new Media("source of video/audio");
```

- Now, you can pass the media class object to the new instance of javafx.scene.media.MediaPlayer object like the following:

```
Mediaplayer player = new MediaPlayer(media);
```

- Invoke the MediaPlayer object's by play() method

```
player.play();
```

- You can instantiate MediaView class and pass player object into its constructor like the following:

```
MediaView mediaView = new MediaView (mediaPlayer)
```

- Add the MediaView object and configure Scene.

```
Scene scene = new Scene(new Pane(mView), 500, 400);
stage.setScene(scene);
stage.setTitle("Video Example");
stage.show();
```

Following example shows you how to run video file in a javaFX application. Demonstrated Video in this example which is available on the YouTube (https://www.youtube.com/watch?v=CtwJ1Mgf_ic) or you can use your own video for running this example.

Example-7: Source Code for ExampleMedia.java

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import javafx.stage.Stage;
import java.io.File;
public class ExampleMedia extends Application
{
  public static void main(String[] args)
  {
    launch(args);
  }
  public void start(Stage stage) throws Exception
  {
    File mFile = new File("Computer.MP4");

    //Instantiating Media class
    Media media = new Media(mFile.toURI().toString());

    //Instantiating MediaPlayer class
    MediaPlayer player = new MediaPlayer(media);
```

```
        //Instantiating MediaView class
        MediaView mView = new MediaView(player);
        player.play();

         //configure scene
        Scene scene = new Scene(new Pane(mView), 500, 400);
        stage.setScene(scene);  // Setting the scene to stage
        stage.setTitle("Video Example");
        stage.show();
    }
}
```

Now, you can compile and execute the above program from the command prompt. The output of the program is display as shown in figure-9.

Figure 9: Example for video incorporating in javaFX application

The above example is for video including in JavaFX application. In a similar way, you can include your audio also.

☞ Check Your Progress-2

1. Explain the term event. What steps are needed to be followed in order to handle the events?

.....

.....

.....

.....

2. What is the difference between foreground and background events?

.....

.....

3. Give names of animation in JavaFX. Explain any two of them with example.

2.6 SUMMARY

This unit explained about the JavaFX UI Controls, which are used in different perspectives such as creating and linking with CSS, building UI with FXML, event handling, effects, animation and media incorporating with UI. Also, in this unit it is explained that how to create and apply CSS on UI control elements with examples. You have also learnt how to build javaFX UI with FXML. The FXML is a XML based language, and it is used for constructing Java object graphs. FXML offers an appropriate alternative to constructing such graphs in procedural code. You also got knowledge about events handling, and it is used when users interact with UI controls.

This unit explained about giving effects and animation to javaFX UI controls elements. At the end of this unit, you have learnt about the media integration in JavaFX application using API that enables you to integrate audio and video into the applications. The package javafx.scene.media covers all the essential classes for making JavaFX applications. This package contains three main classes such as Media, MediaPlayer and MediaView. The package is used for media playback. Now you are able to work with UI controls.

2.7 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRESS

☞ Check Your Progress 1

1) CSS stands for Cascading Style Sheets which are used for adding style such as fonts, colors, backgrounds and spacing between the paragraphs in HTML documents. It is used to control the presentation of one or more web pages. JavaFX CSS is based on the W3C CSS with some additional features. The objective of JavaFX CSS is to allow developers (who are already at ease with CSS for HTML) to use CSS for customizing JavaFX controls. There are two ways to add CSS to JavaFX application by your own Style Sheet and/or inline Style Sheet.

You can add your own external style sheet to a scene in JavaFX application like the following:

```
Scene sc = new Scene(root, 400, 200);
sc.getStylesheets().add("path/name_of_stylesheet.css");
```

Using inline style, you can set CSS styles for a specific element by setting the CSS properties directly on the element by using the setStyle() method, for example, inline style sheet for a button.

```
Button button = new Button("Reset");
button.setStyle("-fx-background-color: #FFbbbb");
```

For more details, you can refer section 2.2 in this Unit.

2) The difference between an ID and a class selector is that an ID is only used to identify a single element in javaFX application whereas class selector is used to identify more than one element. The IDs are only used when one element in the web page should have a particular style applied to it. The name of class selector is preceded by dot(.) character, while the name of ID is preceded by the hash(#) symbol to select the element.

3) A scene graph is a data structure that denotes the contents of a window. The scene graph consists of nodes that show graphical components in the window, such as buttons, radio buttons and checkbox. A scene graph is represented by package javafx.scene in a JavaFX application. The scene class holds all the contents of a scene graph. Some of the nodes behave as containers that contains other nodes. The GUI in a window is created by building a scene graph. For more details, you may refer section 2.2 of this unit.

4) FXML is an XML-based user interface markup language for defining the user interface of a JavaFX application. It is created by Oracle Corporation. It is another form of designing user interfaces using procedural code. You may refer to section 2.3 of this unit for viewing the example.

☞ Check Your Progress 2

1) An event is an object which describes a state change in a source. It generates whenever a user interacts with UI control elements. It is generated by clicking a mouse, pressing a button, entering a character using the keyboard. It can also generate by without human interaction, such as timer expires, a software or hardware failure etc.

The event handling consists of four stages: target selection, route construction, Event capturing, and bubbling. In the target selection phase, when any user interacts with UI controls then an action occurs, at that time the system decides which node gets focus; this happens in target selection phase. Whenever an event is generated, an event dispatch route is created to handle the events in the stage of route construction. This dispatch route comprises the path from the stage to the node on which the event is generated. The event route is regulated by the event dispatch chain which was formed in the operation of the buildEventDispatchChain() method of the selected event target.

In Event capturing stage, the event is transmitted from the source node. The generated event is traversed all the nodes in the route from top to bottom. If the event filter is registered with any of these nodes which are in the dispatch route then event will be executed otherwise, it will be transferred to the target node. The target node processes the event in that case. When the event is captured and processed by the target node or registered filter, the event starts navigationing all the nodes again from the bottom to the root node. If any of the node in the dispatch chain has handler which is registered for the type of event happened then that handler is called, and it will get executed otherwise the process is returned to the next node up in the route and repeat the process. If a handler does not consume the event, then the root node ultimately receives such event and processing is finished. These happen in the event bubbling phase.

2) Foreground Events are based on the direct interaction of a user (e.g. button is pressed), while Background Events involve the interaction of end-user such as hardware or software failure, timer expiry. Foreground Events are created by a person directly interacting with the graphical components in a GUI. For example, double-clicking on a button, moving the mouse, input character through a keyboard, selecting an item from the menu, scrolling the page, etc. The hardware or software failure, operation completion interruptions are the example of background events.

3) The names of animations are Fade Transition, Fill Transition, Rotate Transition, Scale Transition, Stroke Transition, Translate Transition, Path Transition, Sequential Transition, Pause Transition, Parallel Transition, etc. All these transitions are represented by individual classes in the package javafx.animation.

   Translate transition is used to create movement/transition from one point to another point within a defined duration. It is done by keep changing the translateX and translateY properties of the node at the regular interval. In JavaFX, this animation is represented by the class javafx.animation.TranslateTransition. Using **Rotate transition** feature**,** you can rotate an object. You can set rotation by using 'toAngle' and 'byAngle' method. The 'toAngle' indicates what angle the node should rotate, and 'byAngle' indicates how much it should rotate from the current angle of rotation.

2.8 FURTHER READINGS

- Carl Dea, Gerrit Grunwald, José Pereda, Sean Phillips and Mark Heckler "JavaFX 9 by Example", Apress,2017.
- Sharan, Kishori. Beginning Java 8 APIs, Extensions and Libraries: Swing, JavaFX, JavaScript, JDBC and Network Programming APIs. Apress, 2014.
- https://docs.oracle.com/javase/8/javafx/get-started-tutorial/index.htmlhttps://www.w3.org/Style/CSS/Overview.en.html
- https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html
- https://docs.oracle.com/javafx/2/css_tutorial/jfxpub-css_tutorial.htm
- https://docs.jboss.org/richfaces/latest_4_5_X/Developer_Guide/en-US/html/chap-Developer_Guide-Skinning_and_theming.html
- https://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm
- https://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction_to_fxml.html
- https://docs.oracle.com/javafx/2/events/processing.htm
- https://docs.oracle.com/javafx/2/events/filters.htm
- https://docs.oracle.com/javase/8/javafx/api/javafx/scene/effect/Effect.html
- https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm
- https://docs.oracle.com/javafx/2/api/javafx/scene/media/package-summary.html