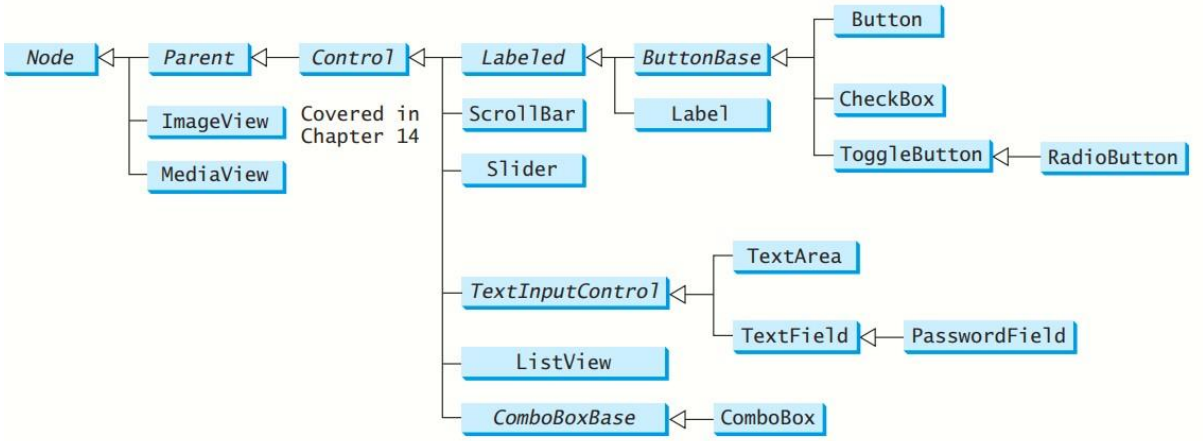
UNIT-8 JAVAFX UI controls and Multimedia

(Chapter-16) Labeled and Label, button, Checkbox, RadioButton, Textfield, TextArea, Combo Box, ListView, Scrollbar, Slider, Video and Audio.

UI Controls
JavaFX provides many UI controls as follows for developing a comprehensive user interface:

- **Label** is a control that displays simple text on the screen. It is generally used to describe the otheruser controls.
- **Button** component is used to control specific function of the application.
- **Checkbox** is used to provide various choices to the user. The check box can be checked(True) or unchecked(False).
- **Radio button** is used to provide the choices to the user. The radio button can be selected or deselected.
- **TextFie1d** is used for getting the input from the user.
- **TextArea** control allows the user to enter the multiple line text.
- **ComboBox** control displays the list of items out of which user can select at the most one item.
- **ListVieW** control displays the list of items out of which user can select one or multiple items from the list.
- **Slider** control is used to display a continuous or discrete range of valid numeric choices and allows the user to interact with the control.

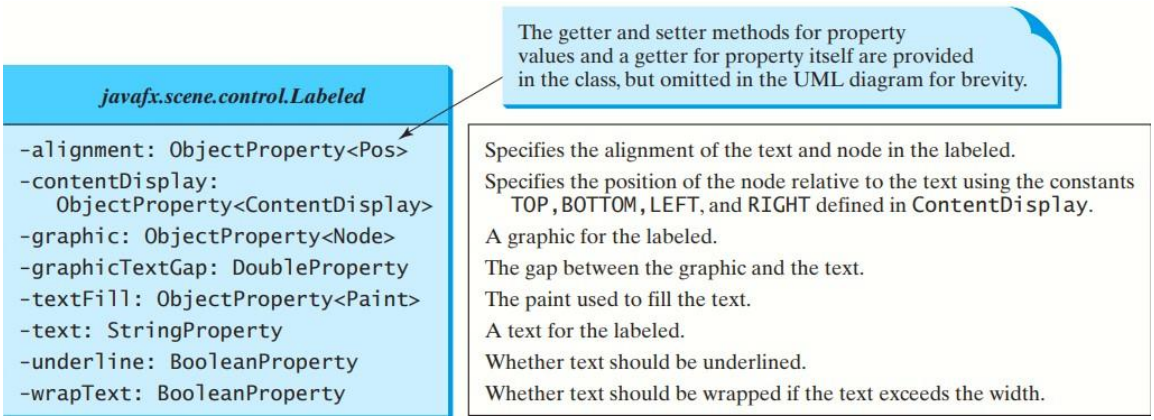


Let us now learn and understand how to create simple JavaFX programs that use these UI controls.

8.1 Labeled and Label

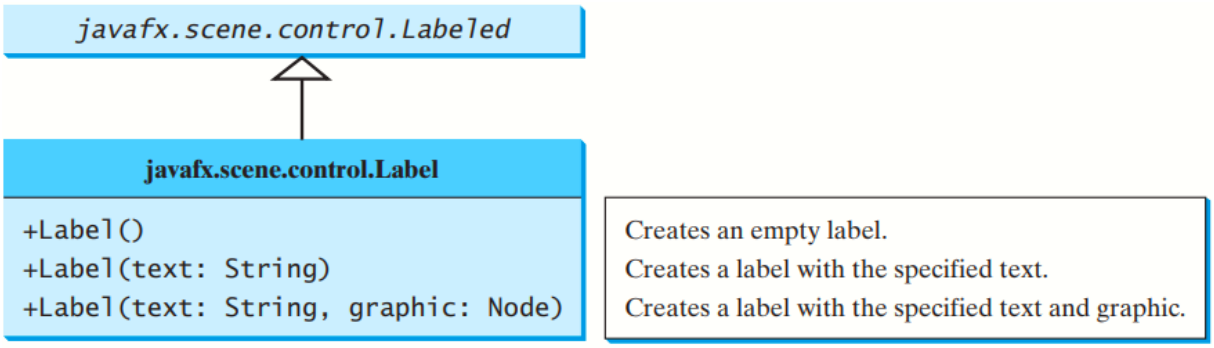
❖ Labels and buttons share many common properties. These common properties are defined in the **Labeled** class

• Labeled defines common properties for Label, Button, CheckBox, and RadioButton.



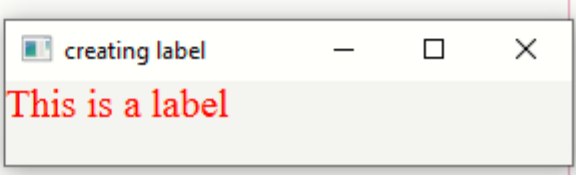
❖ The **Label** control displays simple text on the screen.

• Its main purpose is to describe other components such as textfield, textarea, radio button and so on.
• For using this control the package **javafx.scene.control.Label** need to be imported
• The constructors used for using **Label** control is



Following program shows the creation of label control in JavaFX

```
import javafx.scene.Scene;
import javafx.scene.control.ContentDisplay;
import javafx.scene.control.Label;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class LabelDemo extends Application{
    @Override
    public void start(Stage primaryStage) {
        // create a label
        Label l = new Label("This is a label");
        l.setContentDisplay(ContentDisplay.BOTTOM);
        l.setTextFill(Color.RED);
        l.setFont(Font.font("Times New Roman",20));
        HBox s = new HBox();
        s.getChildren().add(l);

        Scene sc = new Scene(s, 200, 200);
        primaryStage.setScene(sc);
        primaryStage.setTitle("creating label");
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```



8.2 Button

The button control is the most commonly used control in any GUI application. This UI component controls the behavior of the application. Some event gets generated when a button is clicked.

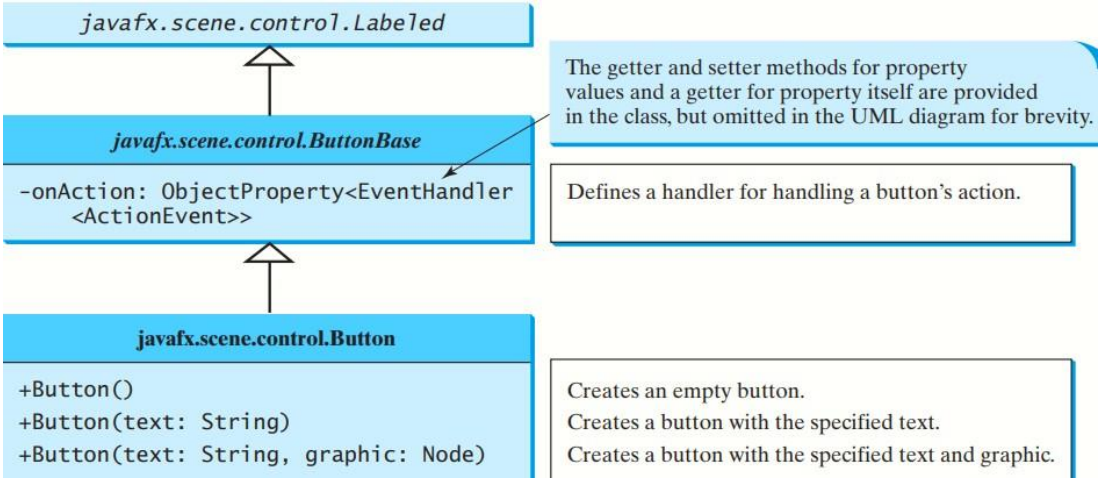
• The button control is created using following code

```
Button btn = new Button("Click Here");
```

• For using the button control we need to import the package **javafx.scene.control.Button**.

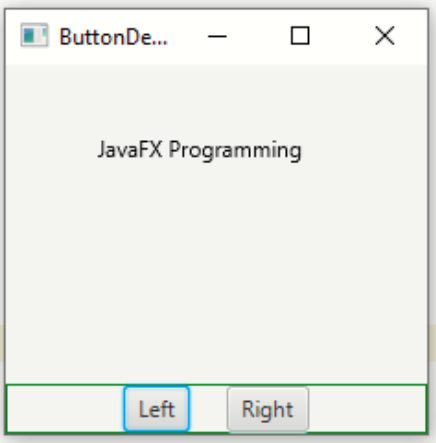
JavaFX provides regular buttons, toggle buttons, check box buttons, and radio buttons. The common features of these buttons are defined in ButtonBase and Labeled classes.

A button is just like a label except that the button has the onAction property defined in the ButtonBase Class.



ButtonBase extends Labeled and defines common features for all buttons.
Following program shows the creation of button control in JavaFX

```
package chapter16;
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.text.Text;
public class ButtonDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        HBox paneForButtons = new HBox(20);
        Button btLeft = new Button("Left");
        Button btRight = new Button("Right");
        paneForButtons.getChildren().addAll(btLeft, btRight);
        paneForButtons.setAlignment(Pos.CENTER);
        paneForButtons.setStyle("-fx-border-color: green");

        BorderPane pane = new BorderPane();
        pane.setBottom(paneForButtons);
```



```
        Text text = new Text(50, 50, "JavaFX Programming");
        Pane paneForText = new Pane();
        paneForText.getChildren().add(text);
        pane.setCenter(paneForText);

        btLeft.setOnAction(e -> text.setX(text.getX() - 10));
        btRight.setOnAction(e -> text.setX(text.getX() + 10));
        // Create a scene and place it in the stage
        Scene scene = new Scene(pane, 450, 200);
        primaryStage.setTitle("ButtonDemo"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

8.3 Checkbox

Checkbox is used to provide more than one choices at a time. For using checkbox in our application program, we must insert following line in the program at the beginning —

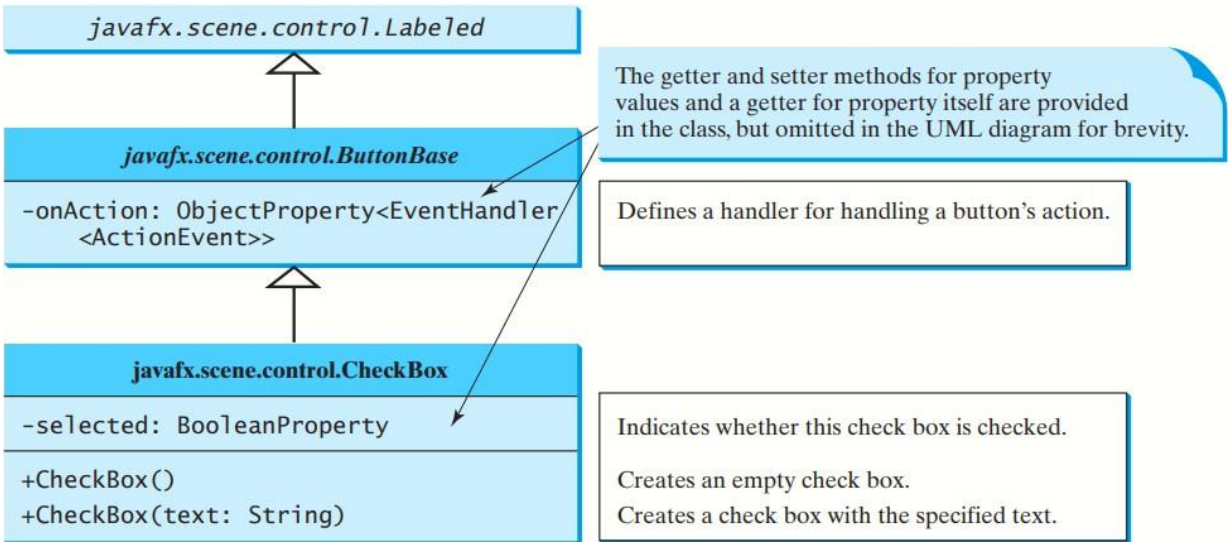
```
import javafx.scene.control.Checkbox
```

The checkbox can be created using following statement

```
CheckBox ch:new CheckBox("Label Name");
```

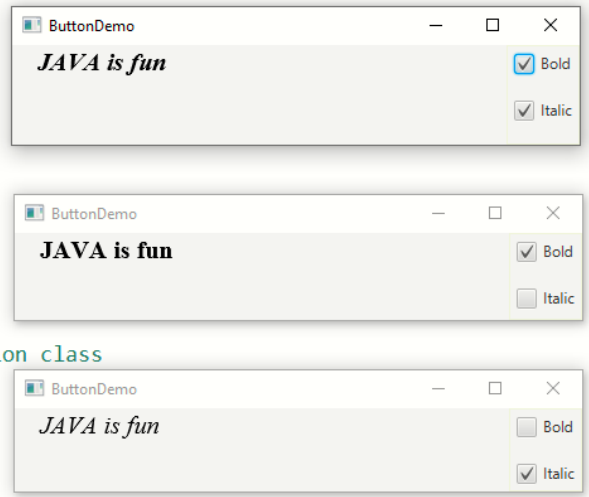
The checkbox is selected true by using the method, setSelected("true")

A **CheckBox** is used for the user to make a selection. Like Button, CheckBox inherits all the properties such as **onAction, text, graphic, alignment, graphicTextGap, textFill, contentDisplay** from **ButtonBase** and **Labeled**



Example: Demonstrating checkbox.

```java
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class CheckBoxDemo extends ButtonDemo {

    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        BorderPane pane = new BorderPane();
        Text text = new Text(20, 20, "JAVA is fun");
        pane.getChildren().add(text);

        Font fontBold = Font.font("Times New Roman", FontWeight.BOLD, FontPosture.REGULAR, 20);
        Font fontItalic = Font.font("Times New Roman",FontWeight.NORMAL, FontPosture.ITALIC, 20);
        Font fontBoldItalic = Font.font("Times New Roman",FontWeight.BOLD, FontPosture.ITALIC, 20);

        VBox paneForCheckBoxes = new VBox(20);
        paneForCheckBoxes.setPadding(new Insets(5, 5, 5, 5));
        paneForCheckBoxes.setStyle("-fx-border-color: beige");
        CheckBox chkBold = new CheckBox("Bold");
        CheckBox chkItalic = new CheckBox("Italic");
        paneForCheckBoxes.getChildren().addAll(chkBold, chkItalic);
        pane.setRight(paneForCheckBoxes);
         EventHandler<ActionEvent> handler = e -> {
           if (chkBold.isSelected() && chkItalic.isSelected())
             text.setFont(fontBoldItalic); // The Bold check box checked
           else if (chkItalic.isSelected())
             text.setFont(fontItalic); // The Italic check box checked
           else
               text.setFont(fontBold);
         };

         chkBold.setOnAction(handler);
         chkItalic.setOnAction(handler);
       // Create a scene and place it in the stage
       Scene scene = new Scene(pane, 450, 200);
       primaryStage.setTitle("ButtonDemo"); // Set the stage title
       primaryStage.setScene(scene); // Place the scene in the stage
       primaryStage.show(); // Display the stage
    }
    public static void main(String[] args) {
    Launch(args);  }
}
```



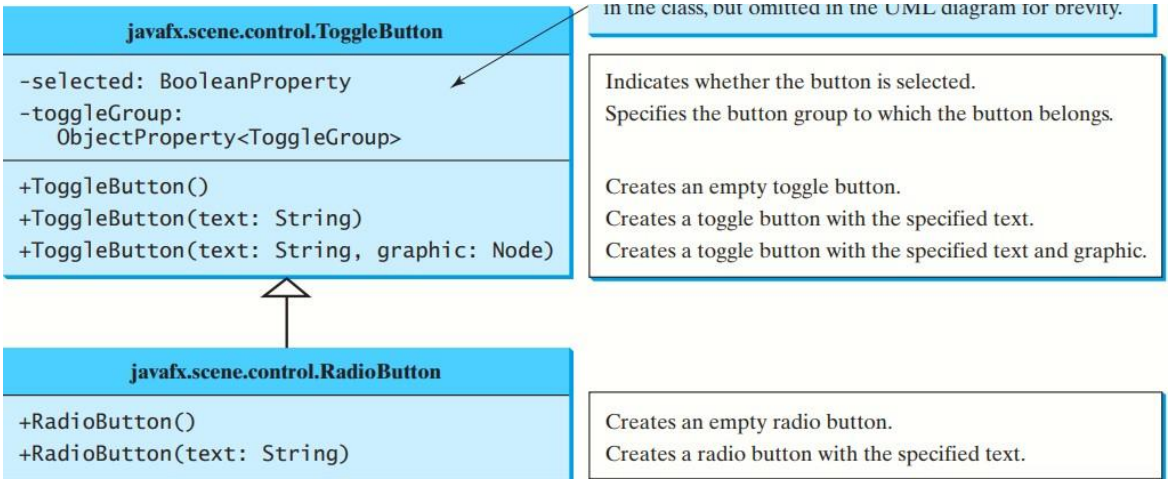
8.4 RadioButton

- The **RadioButton** control is used to make a choice.
- Radio buttons, also known as option buttons, enable the user to choose a single item from a group of choices.

We can create a radiobutton as follows:

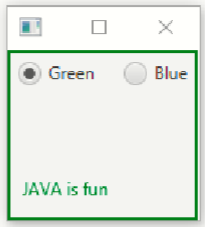
```
RadioButton rb = new RadioButton("Label");
```

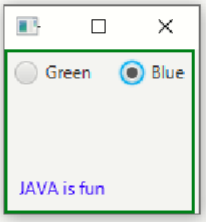
We can group JavaFX RadioButton instances into a **ToggleGroup**. A ToggleGroup allows atmost one RadioButton to be selected at any time.



Following example shows how to create and use the radio button control.

```
1  package application;
2  import javafx.stage.Stage;
3  import javafx.scene.Scene;
4  import javafx.geometry.Insets;
5  import javafx.scene.control.RadioButton;
6  import javafx.scene.control.ToggleGroup;
7  import javafx.scene.layout.BorderPane;
8  import javafx.scene.layout.HBox;
9  import javafx.scene.paint.Color;
10 import javafx.scene.text.Text;
11
12 public class RadioButtonDemo extends ButtonDemo {
13
14     @Override // Override the start method
15     public void start(Stage primaryStage) {
16         BorderPane pane = new BorderPane();
17         Text text = new Text(10, 100, "JAVA is fun");
18         pane.getChildren().add(text);
19
20         HBox paneForRadioButtons = new HBox(20);
21         paneForRadioButtons.setPadding(new Insets(5, 5, 5, 5));
22         paneForRadioButtons.setStyle("-fx-border-width: 2px; -fx-border-color: green");
23
24         RadioButton rbGreen = new RadioButton("Green");
25         RadioButton rbBlue = new RadioButton("Blue");
26         paneForRadioButtons.getChildren().addAll(rbGreen, rbBlue);
27         pane.setLeft(paneForRadioButtons);
28
29         ToggleGroup group = new ToggleGroup();
30         rbGreen.setToggleGroup(group);
31         rbBlue.setToggleGroup(group);
```





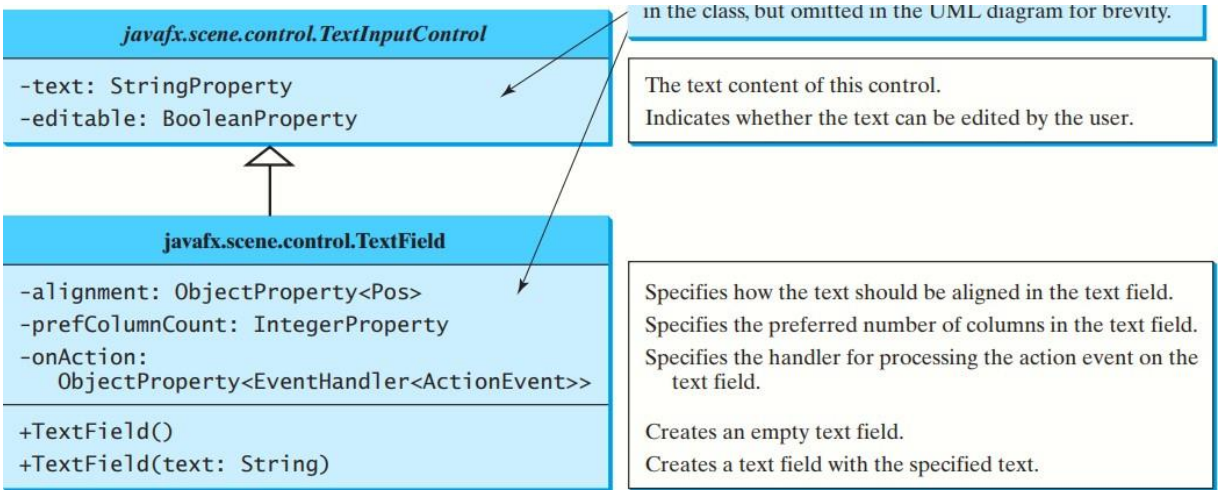
```
33          rbGreen.setOnAction(e -> {
34            if (rbGreen.isSelected())
35              text.setFill(Color.GREEN);
36          });
37
38          rbBlue.setOnAction(e -> {
39            if (rbBlue.isSelected())
40              text.setFill(Color.BLUE);
41          });
42        Scene scene = new Scene(pane, 450, 200);
43        primaryStage.setTitle("RadioButtonDemo"); // Set the stage title
44        primaryStage.setScene(scene); // Place the scene in the stage
45        primaryStage.show(); // Display the stage
46      }
47      public static void main(String[] args) {
48      launch(args);   }
49  }
```

8.5 Textfield

TextField control allows the user to enter the text that can be read by the application.

The package **javafx. scene.control.TextField** need to be imported for using the TextField control.

A text field can be used to enter or display a string. **TextField** is a subclass of **TextInputControl.**

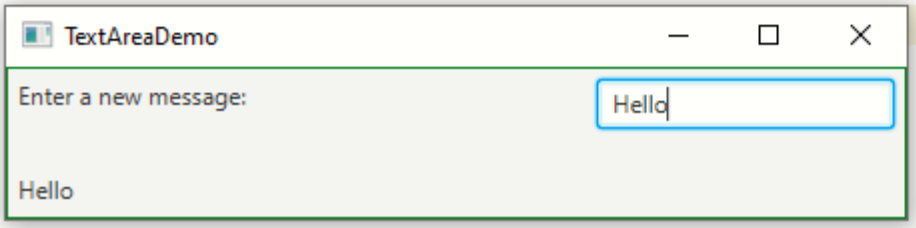


NOTE:

If a text field is used for entering a password, use PasswordField to replace TextField. PasswordField extends TextField and hides the input text with echo characters ******.

Following example shows how to create and use the **Text Field** control.

```
package chapter16;
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class TextFieldDemo extends Application {
  @Override
 public void start(Stage primaryStage) {
   Label lbl = new Label();
    BorderPane paneForTextField = new BorderPane();
    paneForTextField.setPadding(new Insets(5, 5, 5, 5));
    paneForTextField.setStyle("-fx-border-color: green");
    paneForTextField.setLeft(new Label("Enter a new message: "));
    TextField tf = new TextField();
    tf.setAlignment(Pos.BOTTOM_LEFT);
    paneForTextField.setRight(tf);
    tf.setOnAction(e -> lbl.setText(tf.getText()));
    paneForTextField.setBottom(lbl);
    Scene scene = new Scene(paneForTextField, 450, 200);
    primaryStage.setTitle("TextAreaDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
  public static void main(String[] args) {
    launch(args);
  }
}
```



8.6 TextArea

The TextArea control allows to enter multiline text.

This control is represented by class **javafx.scene. control.TextArea**.

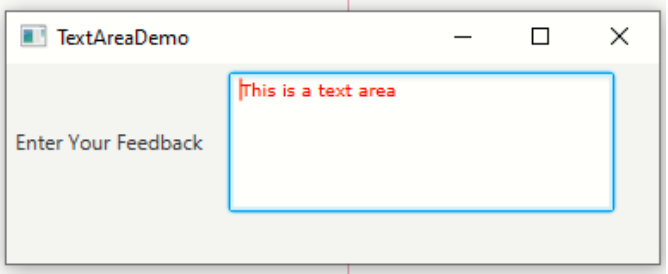
The textarea control can be created using:

```
TextArea ta = new TextArea();
```

We can set the size of the TextArea using **setPrefHeight()** and **setPrefWidth0** functions.

You can place any node in a ScrollPane. ScrollPane provides vertical and horizontal scrolling automatically if the control is too large to fit in the viewing area.

```java
package chapter16;
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextArea;
import javafx.scene.layout.GridPane;
import javafx.scene.text.Font;
public class TextAreaDemo extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    GridPane root = new GridPane();
    root.setPadding(new Insets(5));
    root.setHgap(5);
    root.setVgap(5);
    TextArea taNote = new TextArea("This is a text area");
    taNote.setPrefColumnCount(20);
    taNote.setPrefRowCount(5);
    taNote.setWrapText(true);
    taNote.setStyle("-fx-text-fill: red");
    taNote.setFont(Font.font("Verdana", 10));
    root.add(taNote,1,0);
    root.add(new Label("Enter Your Feedback   "),0,0);
    // Create a scene and place it in the stage
    Scene scene = new Scene(root, 450, 200);
    primaryStage.setTitle("TextAreaDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
  public static void main(String[] args) {
    launch(args);
  }
}
```



8.7 ComboBox

- A **combo box**, also known as a choice list or **drop-down list**, contains a list of items from which the user can choose.
- We can have predefined list of choices using combo box.
- This control is represented by **javafx.scene. control.ComboBox** class.

We can create the comboBox using following statement

```
ComboBox cb = new ComboBox();
```

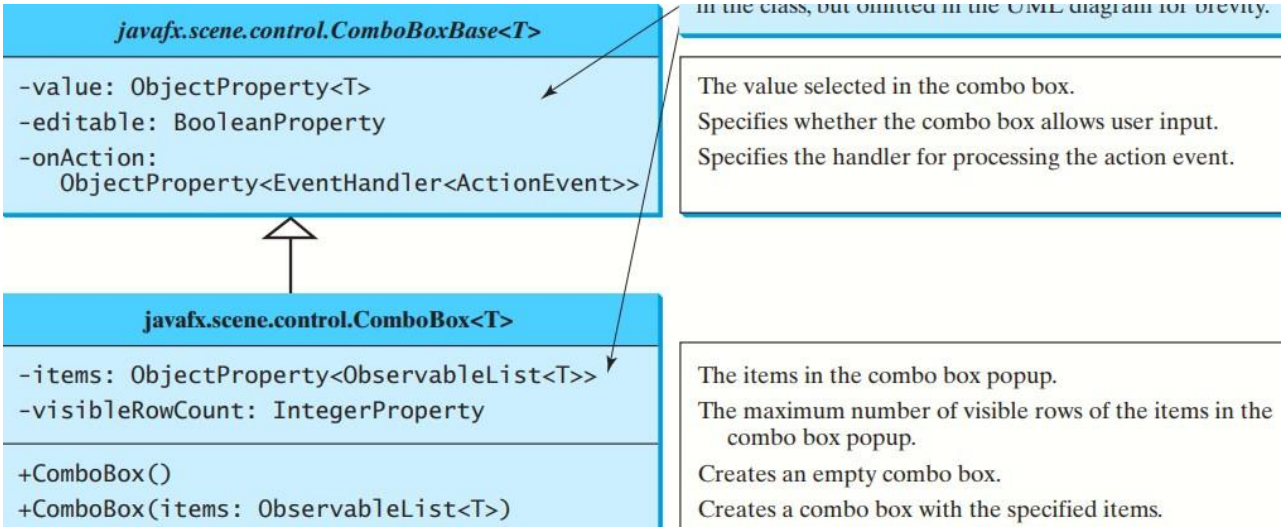
Then we need to add the list of choices to the comboBox. This can be done using

```
cb.getItems().add("Option1");
cb.getItems().add("Option1");
```

….
and so on.

The following statements create a combo box with four items, red color, and value set to the first item.

```
ComboBox<String> cbo = new ComboBox<>();
cbo.getItems().addAll("Item 1", "Item 2",
  "Item 3", "Item 4");
cbo.setStyle("-fx-color: red");
cbo.setValue("Item 1");
```



Following program illustrates the use of **ComboBox** control in the JavaFX application program.

```
1  package application;
2  import javafx.application.*;
8  public class ComboBoxDemo extends Application {
9      public void start(Stage primaryStage)throws Exception{
10
11         Label lbl1 = new Label("Select your favorite country: ");
12         Label lbl2 = new Label();
13         ComboBox<String> cb = new ComboBox<String>();
14         cb.getItems().addAll("India","UK","US","Japan");
15
16         GridPane root = new GridPane();
17         root.addRow(0,lbl1,cb);
18
19         cb.setOnAction(e -> lbl2.setText(cb.getValue()));
20         root.add(lbl2,0,50);
21
22         Scene scene = new Scene(root, 350, 200);
23         primaryStage.setTitle("ComboBox Demo");
24         primaryStage.setScene(scene);
25         primaryStage.show();
26     }
27     public static void main(String[] args) {
28         Launch(args); }
29 }
```

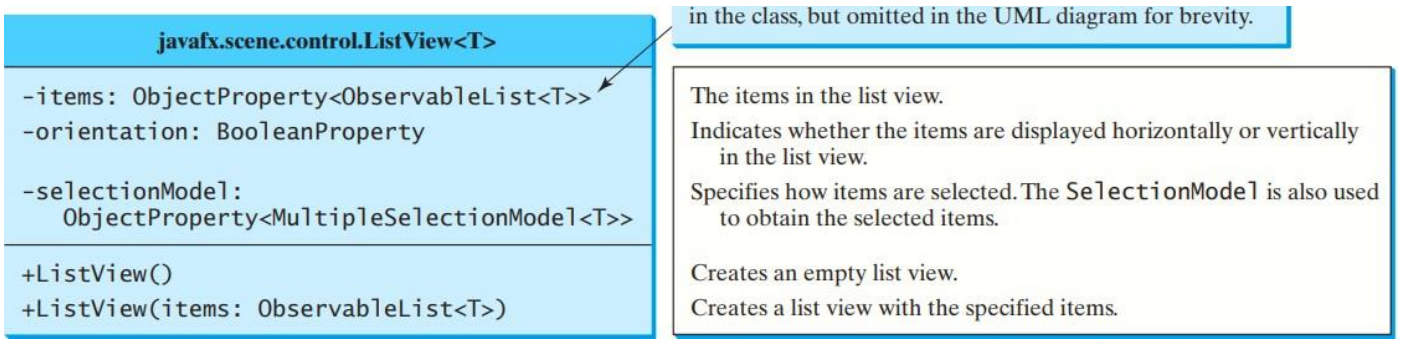
8.8 ListView

- A ListView is a control that basically performs the same function as a combo box, but it enables the user to choose a single value or multiple values from a predefined list of choices.
- The JavaFX ListView control is represented by the class **javafx.scene.control.ListView** .

The ListView can be created as follows –

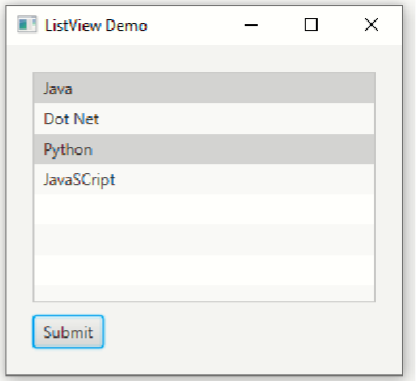
```
ListView lv = new ListView()
```

The selection mode is defined in one of the two constants **SelectionMode.MULTIPLE** and **SelectionMode.SINGLE**, which indicates whether a single item or multiple items can be selected.
The **default** value is SelectionMode.SINGLE.



Following example program shows how to use ListView Control

```
1  package application;
2  import javafx.application.Application;
3  import javafx.collections.ObservableList;
4  import javafx.geometry.Insets;
5  import javafx.scene.Scene;
6  import javafx.scene.control.Button;
7  import javafx.scene.control.SelectionMode;
8  import javafx.scene.layout.VBox;
9  import javafx.stage.Stage;
10 import javafx.scene.control.ListView;
11
12 public class ListViewDemo extends Application {
13     ListView<String> listView = new ListView<String>();
14     @Override
15     public void start(Stage primaryStage) throws Exception {
16         primaryStage.setTitle("ListView Demo");
17         Button button = new Button("Submit");
18
19         listView.getItems().addAll("Java", "Dot Net", "Python", "JavaSCript");
20         listView.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
21
22         button.setOnAction(e -> buttonClicked());
23
24         VBox layout = new VBox(10);
25         layout.setPadding(new Insets(20, 20, 20, 20));
26         layout.getChildren().addAll(listView, button);
27
28         Scene scene = new Scene(layout, 300, 250);
29         primaryStage.setScene(scene);
30         primaryStage.show();
31     }
33     private void buttonClicked(){
34         String message = "";
35         ObservableList<String> languages;
36         languages = listView.getSelectionModel().getSelectedItems();
37
38         for(String m: languages)
39             message += m + "\n";
40
41         System.out.println(message);
42     }
43     public static void main(String[] args) {
44         launch(args);
45     }
46 }
```



Problems Javadoc Declaration Console

ListViewDemo [Java Application] C:\Program Files\JAVA\jdk-13.0.2\bin\javaw.exe (Mar 30, 2020, 10:01:38 AM)

```
Java
Python
```

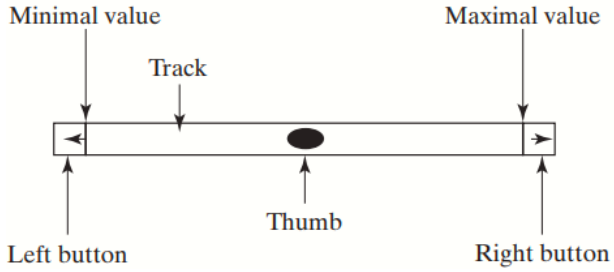
8.9 Scrollbar

ScrollBar is a control that enables the user to select from a range of values. For creating this control we use **javafx.scene.control.ScrollBar** class.

We can create the scrollbar in horizontal direction or in vertical direction. If we want the scrollbar to be displayed vertically then we call setOrientation() method

```
sb. setOrientation(Orientation.VERTICAL);
```

A scroll bar represents a range of values graphically

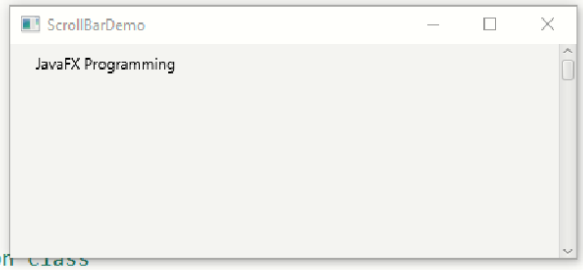


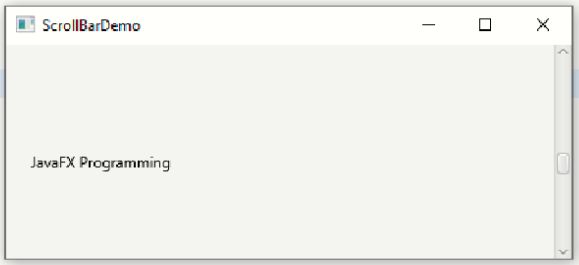
Following example program shows how to use **ScrollBar** Control

javafx.scene.control.ScrollBar	
-blockIncrement: DoubleProperty	The amount to adjust the scroll bar if the track of the bar is clicked (default: 10).
-max: DoubleProperty	The maximum value represented by this scroll bar (default: 100).
-min: DoubleProperty	The minimum value represented by this scroll bar (default: 0).
-unitIncrement: DoubleProperty	The amount to adjust the scroll bar when the `increment()` and `decrement()` methods are called (default: 1).
-value: DoubleProperty	Current value of the scroll bar (default: 0).
-visibleAmount: DoubleProperty	The width of the scroll bar (default: 15).
-orientation: ObjectProperty<Orientation>	Specifies the orientation of the scroll bar (default: HORIZONTAL).
+ScrollBar()	Creates a default horizontal scroll bar.
+increment()	Increments the value of the scroll bar by `unitIncrement`.
+decrement()	Decrements the value of the scroll bar by `unitIncrement`.

Following example program shows how to use **ScrollBar** Control

```
2  import javafx.stage.Stage;
3  import javafx.geometry.Orientation;
4  import javafx.scene.Scene;
5  import javafx.scene.control.ScrollBar;
6  import javafx.scene.layout.BorderPane;
7  import javafx.scene.layout.Pane;
8  import javafx.scene.text.Text;
9
10 public class ScrollBarDemo extends Application {
11   @Override // Override the start method in the Application class
12   public void start(Stage primaryStage) {
13     Text text = new Text(20, 20, "JavaFX Programming");
14
15     ScrollBar sbVertical = new ScrollBar();
16     sbVertical.setOrientation(Orientation.VERTICAL);
17
18     // Create a text in a pane
19     Pane paneForText = new Pane();
20     paneForText.getChildren().add(text);
21
22     // Create a border pane to hold text and scroll bars
23     BorderPane pane = new BorderPane();
24     pane.setCenter(paneForText);
25     pane.setRight(sbVertical);
26
27     // Listener for vertical scroll bar value change
28     sbVertical.valueProperty().addListener(ov ->
29       text.setY(sbVertical.getValue() * paneForText.getHeight() /
30         sbVertical.getMax()));
```



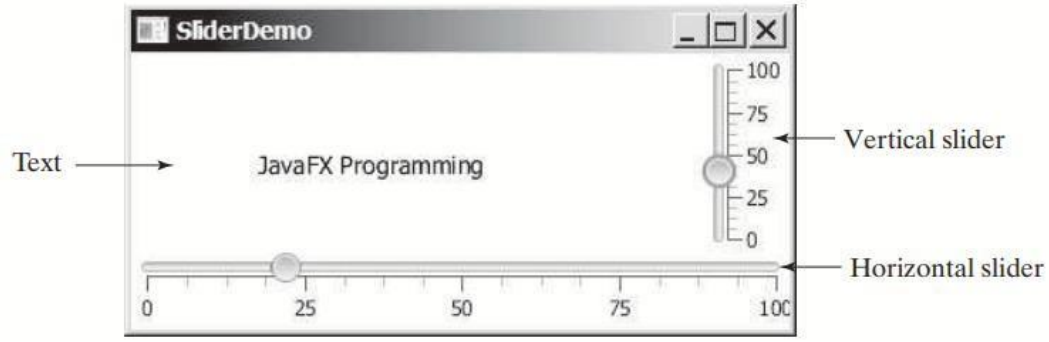


```
32      // Create a scene and place it in the stage
33      Scene scene = new Scene(pane, 450, 170);
34      primaryStage.setTitle("ScrollBarDemo"); // Set the stage title
35      primaryStage.setScene(scene); // Place the scene in the stage
36      primaryStage.show(); // Display the stage
37    }
38    public static void main(String[] args) {
39      Launch(args);
40    }
41  }
```

8.10 Slider

Slider is similar to **ScrollBar**, but Slider has more properties and can appear in many forms. JavaFX slider is represented by the JavaFX class **javafx.scene.controI.Slider**
The values of a vertical scroll bar increase from top to bottom, but the values of a vertical slider decrease from top to bottom.

The sliders move the message on a pane horizontally and vertically.



Note
The values of a vertical scroll bar increase from top to bottom, but the values of a vertical slider decrease from top to bottom.

javafx.scene.control.Slider	
-blockIncrement: DoubleProperty	The amount to adjust the slider if the track of the bar is clicked (default: 10)
-max: DoubleProperty	The maximum value represented by this slider (default: 100).
-min: DoubleProperty	The minimum value represented by this slider (default: 0).
-value: DoubleProperty	Current value of the slider (default: 0).
-orientation: ObjectProperty<Orientation>	Specifies the orientation of the slider (default: HORIZONTAL).
-majorTickUnit: DoubleProperty	The unit distance between major tick marks.
-minorTickCount: IntegerProperty	The number of minor ticks to place between two major ticks.
-showTickLabels: BooleanProperty	Specifies whether the labels for tick marks are shown.
-showTickMarks: BooleanProperty	Specifies whether the tick marks are shown.
+Slider()	Creates a default horizontal slider.
+Slider(min: double, max: double, value: double)	Creates a slider with the specified min, max, and value.

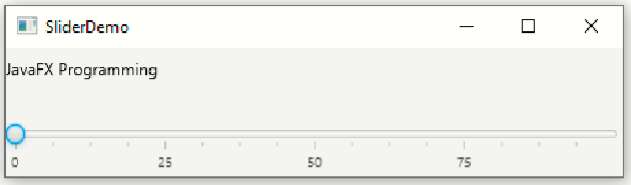
in the class, but omitted in the UML diagram for brevity.

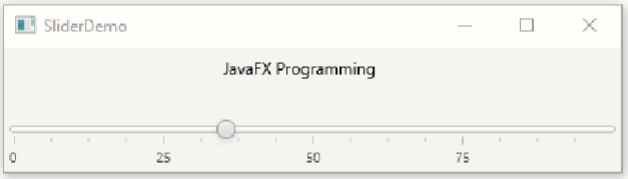
Following, example shows how to use slider control.

(In following program setting the scene and main() class definition is omitted as it is same as above programs.)

- A listener is registered to listen for the **slHorizontal** value property change (lines 28–30)
- When the value of the slider changes, the listener is notified by invoking the handler to set a new position for the text (lines 29-30).

```java
package application;
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Slider;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;
import javafx.scene.text.Text;

public class SliderDemo extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    Text text = new Text(20, 20, "JavaFX Programming");

    Slider slHorizontal = new Slider();
    slHorizontal.setShowTickLabels(true);
    slHorizontal.setShowTickMarks(true);

    // Create a text in a pane
    Pane paneForText = new Pane();
    paneForText.getChildren().add(text);

    // Create a border pane to hold text and scroll bars
    BorderPane pane = new BorderPane();
    pane.setCenter(paneForText);
    pane.setBottom(slHorizontal);

    slHorizontal.valueProperty().addListener(ov ->
      text.setX(slHorizontal.getValue() * paneForText.getWidth() /
        slHorizontal.getMax()));
```



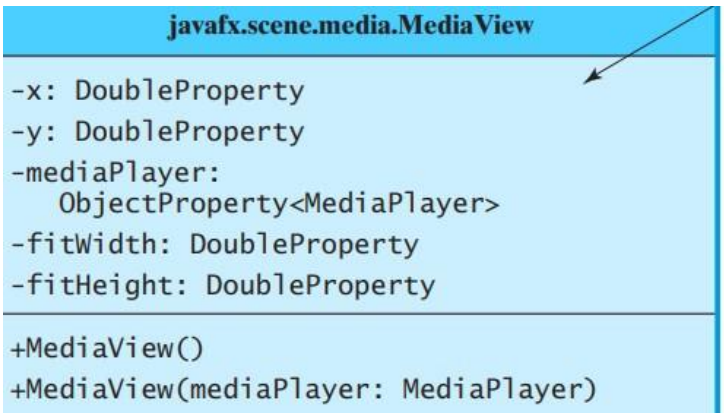


8.11 Video and Audio

JavaFX provides the media API using which it is possible to play the audio or video.
You can use the **javafx.scene.media.Media** class to obtain the source of the media, the **MediaPlayer** class to play and control the media, and the **MediaView** class to display the video and provides the properties for viewing the media.

javafx.scene.media.Media	
-duration: ReadOnlyObjectProperty<Duration>	The durations in seconds of the source media.
-width: ReadOnlyIntegerProperty	The width in pixels of the source video.
-height: ReadOnlyIntegerProperty	The height in pixels of the source video.
+Media(source: String)	Creates a `Media` from a URL source.

javafx.scene.media.MediaPlayer	
-autoPlay: BooleanProperty	Specifies whether the playing should start automatically.
-currentCount: ReadOnlyIntegerProperty	The number of completed playback cycles.
-cycleCount: IntegerProperty	Specifies the number of time the media will be played.
-mute: BooleanProperty	Specifies whether the audio is muted.
-volume: DoubleProperty	The volume for the audio.
-totalDuration: ReadOnlyObjectProperty<Duration>	The amount of time to play the media from start to finish.
+MediaPlayer(media: Media)	Creates a player for a specified media.
+play(): void	Plays the media.
+pause(): void	Pauses the media.
+seek(): void	Seeks the player to a new playback time.



Playing Audio
Following steps are followed to play an audio using JavaFX application

Step 1 : Create an instance of Media class. The file name along with its complete path location is passed as a parameter to this class. For example
`Media media = new Media();`

Step 2 : Now pass the object of this Media class to the MediaPlayer class.
`MediaPlayer = new MediaPlayer(media);`

Step 3 : Now for playing the audio we use **play()** method when **onReady** event is triggered.

A Media object supports live streaming. You can now download a large media file and the same time. A Media object can be shared by multiple media players and different views can use the same MediaP1ayer object.

```
public class MyJavaFXApplication extends Application {

  @Override
  public void start(Stage primaryStage) {
    Media media=new Media("file:///D:/bell_ringing.mp3");
    MediaPlayer media_player=new MediaPlayer(media);
    media_player.setAutoPlay(true);
    Label L=new Label("Bell is Ringing....");
    HBox root=new HBox();
    root.getChildren().add(L);
    Scene scene=new Scene(root,100,100);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Audio Playing Demo Program");
    primaryStage.show();
  }
```

Playing Video
In the same manner it is possible to play the video using JavaFX application program.

Note that for displaying the video at the output we have to add instance of **MediaView** along with **Media** and **MediaPlayer** instances. These code for this is as follows.

`MediaView vp = new MediaView();`

Thus the instance is created in the variable **vp**. This control is then added to the **HBox** pane using **getChildren().add()** method.

Here is the demonstration.

```
1  package application;
2  import java.io.File;
3  import javafx.application.Application;
4  import javafx.scene.media.Media;
5  import javafx.scene.media.MediaPlayer;
6  import javafx.stage.Stage;
7  public class MediaDemo  extends Application
8  {
9      @Override
10     public void start (Stage primaryStage)  {
11
12         String path = "src/media/Piper.mp4";
13
14         //Instantiating Media class
15         Media media = new Media(new File(path).toURI().toString());
16
17         //Instantiating MediaPlayer class
18         MediaPlayer mediaPlayer = new MediaPlayer(media);
19
20         //by setting this property to true, the audio will be played
21         mediaPlayer.setAutoPlay(true);
22         primaryStage.show();
23     }
24     public static void main(String[] args) {
25         launch(args);
26     }
27 }
```