Chapter 16
JavaFX UI Controls and Multimedia

Motivations
A graphical user interface (GUI) makes a system user-friendly and easy to use. Creating a GUI requires creativity and knowledge of how GUI components work. Since the GUI components in Java are very flexible and versatile, you can create a wide assortment of useful user interfaces.

Previous chapters briefly introduced several GUI components. This chapter introduces the frequently used GUI components in detail.
Objectives

- To create graphical user interfaces with various user-interface controls (§§16.2–16.11).
- To create a label with text and graphic using the **Label** class and explore properties in the abstract **Labeled** class (§16.2).
- To create a button with text and graphic using the **Button** class and set a handler using the **setOnAction** method in the abstract **ButtonBase** class (§16.3).
- To create a check box using the **CheckBox** class (§16.4).
- To create a radio button using the **RadioButton** class and group radio buttons using a **ToggleGroup** (§16.5).
- To enter data using the **TextField** class and password using the **PasswordField** class (§16.6).
- To enter data in multiple lines using the **TextArea** class (§16.7).
- To select a single item using **ComboBox** (§16.8).
- To select a single or multiple items using **ListView** (§16.9).
- To select a range of values using **ScrollBar** (§16.10).
- To select a range of values using **Slider** and explore differences between **ScrollBar** and **Slider** (§16.11).
- To develop a tic-tac-toe game (§16.12).
- To view and play video and audio using the **Media**, **MediaPlayer**, and **MediaView** (§16.13).
- To develop a case study for showing the national flag and play anthem (§16.14).

Frequently Used UI Controls

**Labeled**

- A *label* is a display area for a short text, a *Node*, or both. It is often used to label other controls (usually text fields).
- *Label* and *Button* share many common properties. These common properties are defined in the *Labeled* class.

```java
javafx.scene.control.Labeled
- alignment: ObjectProperty<Pos>
- contentDisplay: ObjectProperty<ContentDisplay>
- graphic: ObjectProperty<Node>
- graphicTextGap: DoubleProperty
- textField: ObjectProperty<Paint>
- text: StringProperty
- underline: BooleanProperty
-wrapText: BooleanProperty
```

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

- Specifies the alignment of the text and node in the labeled.
- Specifies the position of the node relative to the text using the constants TOP, BOTTOM, LEFT, and RIGHT defined in ContentDisplay.
- A graphic for the labeled.
- The gap between the graphic and the text.
- The paint used to fill the text.
- A text for the labeled.
- Whether text should be underlined.
- Whether text should be wrapped if the text exceeds the width.

A graphic or text or both can be placed in *Labeled*.

---

**Label**

```java
javafx.scene.control.Labeled
```

```java
javafx.scene.control.Label
+ Label()
+ Label(text: String)
+ Label(text: String, graphic: Node)
```

Creates an empty label.
Creates a label with the specified text.
Creates a label with the specified text and graphic.
**ButtonBase & Button**

A *button* triggers an event when clicked. JavaFX provides regular buttons, toggle buttons, check boxes & radio buttons. The common features of these buttons are defined in *ButtonBase & Labeled*.

```java
javafx.scene.control.ButtonBase

@onAction: ObjectProperty<EventHandler<ActionEvent>>

Defines a handler for handling a button's action.

javafx.scene.control.Button

+Button()
+Button(text: String)
+Button(text: String, graphic: Node)

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

Creates an empty button.
Creates a button with the specified text.
Creates a button with the specified text and graphic.
```

---

**Example: Button**

```
java
public class ButtonDemo {
    public static void main(String[] args) {
        Button button = new Button("Click Me!");
        button.setOnAction(event -> System.out.println("Button clicked!");
        primaryStage.setScene(new.Scene(root, 300, 200));
        primaryStage.show();
    }
}
```
**CheckBox**

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`, `textField`, `contentDisplay` from `ButtonBase` and `Labeled`.

```
javafx.scene.control.ButtonBase
  -onAction: ObjectProperty<EventHandler<ActionEvent>>
  +CheckBox()
  +CheckBox(text: String)
```

- `selected`: `BooleanProperty`  
  Indicates whether this check box is checked.
  Creates an empty check box.
  Creates a check box with the specified text.

```
javafx.scene.control.CheckBox
```

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

**Example: CheckBox**

```
CheckBoxDemo
```

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.
RadioButton

Radio buttons, also known as option buttons, enable us to choose a single item from a group of choices. In appearance radio buttons resemble check boxes, but check boxes display a square that is either checked or blank, whereas radio buttons display a circle that is either filled (if selected) or blank (if not selected).

Example: RadioButton
TextField

- A TextField can be used to enter or display a string
- TextField is a subclass of TextInputControl

 TextFieldDemo

Example: TextField

TextFieldDemo

Programming is fun
**TextArea**

A **TextArea** enables the user to enter multiple lines of text.

```java
import javafx.scene.control.TextArea;

TextArea()
TextArea(text: String)
```

### Attributes

- `text`: StringProperty
- `editable`: BooleanProperty
- `prefColumnCount`: IntegerProperty
- `prefRowCount`: IntegerProperty
- `wrapText`: BooleanProperty

### Methods

- Creates an empty text area.
- Creates a text area with the specified text.

---

**Example: TextArea**

```java
import javafx.scene.layout.BorderPane;
import javafx.application.Application;

BorderPane()
TextAreaDemo()
```

### Attributes

- `lblImageTitle`: Label
- `taDescription`: TextArea

### Methods

- `setImageView(im: ImageView)`
- `setDescription(text: String)`

---

The Canadian national flag ...

Canada

The Canadian national flag ...

Canada
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.

Example: ComboBox

This example lets users view an image and a description of a country's flag by selecting the country from a combo box.
ListView

A ListView is a component that performs basically the same function as a combo box, but it enables the user to choose a single value as well as multiple values.

```java
javafx.scene.control.ListView<T>
- items: ObjectProperty<ObservableList<T>>;
- orientation: BooleanProperty
- selectionModel: ObjectProperty<MultipleSelectionModel<T>>
+ ListView()
+ ListView(items: ObservableList<T>)
```

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

The items in the list view.
Indicates whether the items are displayed horizontally or vertically in the list view.
Specifies how items are selected. The SelectionModel is also used to obtain the selected items.
Creates an empty list view.
Creates a list view with the specified items.

Example: ListView

This program lets users select countries in a list and displays the flags of the selected countries in the labels.
ScrollBar

A ScrollBar is a control that enables the user to select from a range of values. The scrollbar appears in two styles: horizontal, vertical.

Properties: ScrollBar

Minimal value

Maximal value

Track

Thumb

Left button

Right button

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

The amount to adjust the scroll bar if the track of the bar is clicked (default: 10).
The maximum value represented by this scroll bar (default: 100).
The minimum value represented by this scroll bar (default: 0).
The amount to adjust the scroll bar when the increment() and decrement() methods are called (default: 1).
Current value of the scroll bar (default: 0).
The width of the scroll bar (default: 15).
Specifies the orientation of the scroll bar (default: HORIZONTAL).
Creates a default horizontal scroll bar.
Increments the value of the scroll bar by unitIncrement.
Decrements the value of the scroll bar by unitIncrement.
Example: ScrollBar

This example uses horizontal and vertical scrollbars to control a message displayed on a panel. The horizontal scrollbar is used to move the message to the left or the right, and the vertical scrollbar to move it up and down.

Slider

Slider is similar to ScrollBar, but Slider has more properties and can appear in many forms.
**Example: Slider**

Rewrite of the preceding `ScrollBar` example using the `Slider` to control a message displayed on a panel

![SliderDemo](image)

---

**Case Study: Bounce Ball**

Listing 15.17 gives a program that displays a bouncing ball. We can add a `Slider` to control the speed of the ball movement

![BounceBallSliderDemo](image)
Case Study: TicTacToe

Cell
- token: char
  + getToken(): char
  + setToken(token: char): void
  - handleMouseClick(): void

Token used in the cell (default: ' ').
Returns the token in the cell.
Sets a new token in the cell.
Handles a mouse click event.

Case Study: TicTacToe (cont.)

TicTacToe
- whoseTurn: char
  - cell: Cell[][]
  - lblStatus: Label
  + TicTacToe()
  + isFull(): boolean
  + isWon(token: char): boolean

Indicates which player has the turn, initially X.
A 3 x 3, two-dimensional array for cells.
A label to display game status.
Constructs the TicTacToe user interface.
Returns true if all cells are filled.
Returns true if a player with the specified token has won.
Media

We can use the: 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.

JavaFX supports:
MP3, AIFF, WAV, and MPEG-4 audio formats
FLV and MPEG-4 video formats

MediaPlayer

The MediaPlayer class plays and controls the media with properties such as autoPlay, currentCount, cycleCount, mute, volume, and totalDuration.

<table>
<thead>
<tr>
<th>java.awt.image.Media</th>
<th>java.awt.image.Media</th>
<th>java.awt.image.Media</th>
</tr>
</thead>
<tbody>
<tr>
<td>-duration: ReadOnlyObjectProperty</td>
<td>-width: ReadOnlyIntegerProperty</td>
<td>-height: ReadOnlyIntegerProperty</td>
</tr>
<tr>
<td>+Media(source: String)</td>
<td>+MediaPlayer(media: Media)</td>
<td>+play(): void</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+pause(): void</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+seek(): void</td>
</tr>
</tbody>
</table>

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

The durations in seconds of the source media.
The width in pixels of the source video.
The height in pixels of the source video.
Creates a Media from a URL source.

Specifies whether the playing should start automatically.
The number of completed playback cycles.
Specifies the number of time the media will be played.
Specifies whether the audio is muted.
The volume for the audio.
The amount of time to play the media from start to finish.

Creates a player for a specified media.
Plays the media.
Pauses the media.
Seeks the player to a new playback time.
MediaView

The MediaView class is a subclass of Node that provides a view of the Media being played by a MediaPlayer. The MediaView class provides the properties for viewing the media.

```java
javafx.scene.media.MediaView
-x: DoubleProperty
-y: DoubleProperty
-mediaPlayer: 
    ObjectProperty<MediaPlayer>
-fitWidth: DoubleProperty
-fitHeight: DoubleProperty
+MediaView()
+MediaView(mediaPlayer: MediaPlayer)
```

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

- Specifies the current x-coordinate of the media view.
- Specifies the current y-coordinate of the media view.
- Specifies a media player for the media view.
- Specifies the width of the view for the media to fit.
- Specifies the height of the view for the media to fit.
- Creates an empty media view.
- Creates a media view with the specified media player.

Example: Media, MediaPlayer & MediaView

This example displays a video in a window. We can use the:

- play/pause button to play or pause the video
- rewind button to restart the video
- slider to control the volume of the audio

http://www.eclipsecat.com/sites/default/files/small.mp4
Case Study: Flags & Anthems

This program displays a nation’s flag and plays its anthem.