Chapter 16
JavaFX UI Controls and Multimedia

1

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.
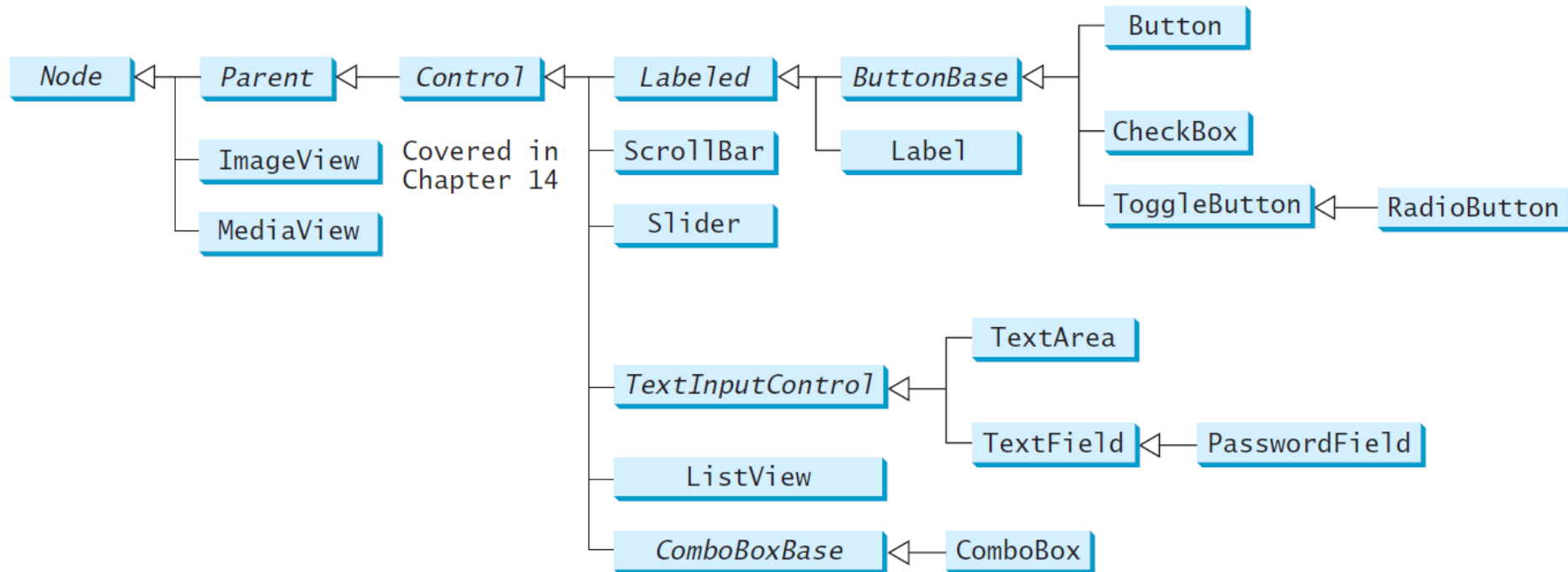
Objectives

F To create graphical user interfaces with various user-interface controls (§§16.2–16.11).

F To create a label with text and graphic using the **Label** class and explore properties in the abstract **Labeled** class (§16.2).

F 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).

F To create a check box using the **CheckBox** class (§16.4).

F To create a radio button using the **RadioButton** class and group radio buttons using a **ToggleGroup** (§16.5).

F To enter data using the **TextField** class and password using the **PasswordField** class (§16.6).

F To enter data in multiple lines using the **TextArea** class (§16.7).

F To select a single item using **ComboBox** (§16.8).

F To select a single or multiple items using **ListView** (§16.9).

F To select a range of values using **ScrollBar** (§16.10).

F To select a range of values using **Slider** and explore differences between **ScrollBar** and **Slider** (§16.11).

F To develop a tic-tac-toe game (§16.12).

F To view and play video and audio using the **Media**, **MediaPlayer**, and **MediaView** (§16.13).

F To develop a case study for showing the national flag and play anthem (§16.14).

Frequently Used UI Controls

```
Node ◁— Parent ◁— Control ◁— Labeled ◁— ButtonBase ◁— Button
         │                    │          │              CheckBox
         ImageView   Covered in ScrollBar  Label
         │           Chapter 14                          ToggleButton ◁— RadioButton
         MediaView            Slider

                             TextArea
                    TextInputControl ◁
                                      TextField ◁— PasswordField
                             ListView
                             ComboBoxBase ◁— ComboBox
```

Throughout this book, the prefixes **lbl**, **bt**, **chk**, **rb**, **tf**, **pf**, **ta**, **cbo**, **lv**, **scb**, **sld**, and **mp** are used to name reference variables for **Label**, **Button**, **CheckBox**, **RadioButton**, **TextField**, **PasswordField**, **TextArea**, **ComboBox**, **ListView**, **ScrollBar**, **Slider**, and **MediaPlayer**.

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

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.

javafx.scene.control.Labeled

-alignment: ObjectProperty<Pos>	Specifies the alignment of the text and node in the labeled.
-contentDisplay: ObjectProperty<ContentDisplay>	Specifies the position of the node relative to the text using the constants `TOP`, `BOTTOM`, `LEFT`, and `RIGHT` defined in `ContentDisplay`.
-graphic: ObjectProperty<Node>	A graphic for the labeled.
-graphicTextGap: DoubleProperty	The gap between the graphic and the text.
-textFill: ObjectProperty<Paint>	The paint used to fill the text.
-text: StringProperty	A text for the labeled.
-underline: BooleanProperty	Whether text should be underlined.
-wrapText: BooleanProperty	Whether text should be wrapped if the text exceeds the width.

Label

The Label class defines labels.
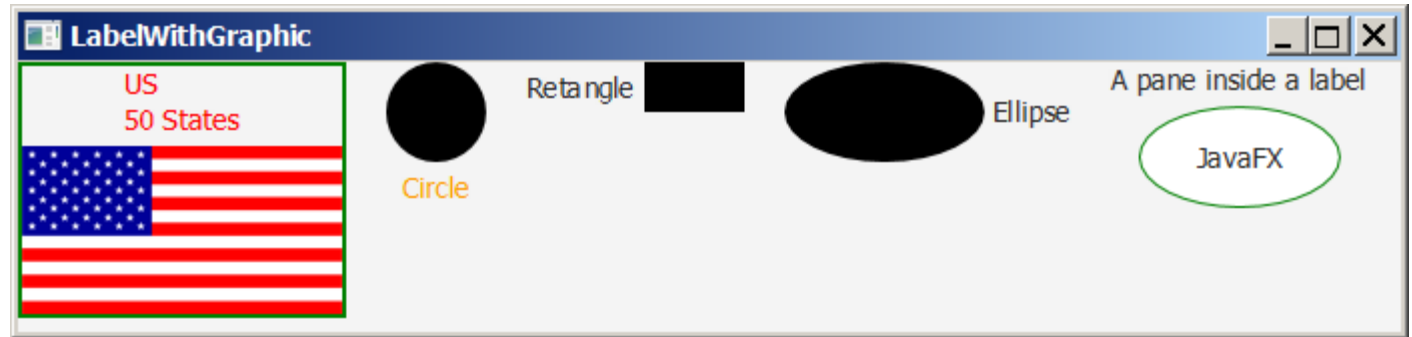

```
javafx.scene.control.Labeled
              △
              │
     javafx.scene.control.Label

+Label()                              Creates an empty label.
+Label(text: String)                  Creates a label with the specified text.
+Label(text: String, graphic: Node)   Creates a label with the specified text and graphic.
```
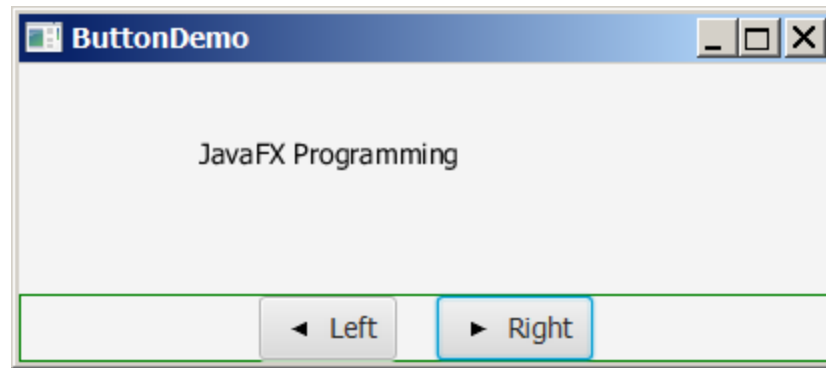
LabelWithGraphic   Run

ButtonBase and Button

A *button* is a control that triggers an action event when clicked. 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.

```
javafx.scene.control.Labeled
```
⬆
```
javafx.scene.control.ButtonBase
```
-onAction: ObjectProperty<EventHandler   <ActionEvent>>

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.

Defines a handler for handling a button's action.

⬆
```
javafx.scene.control.Button
```
+Button()
+Button(text: String)
+Button(text: String, graphic: Node)

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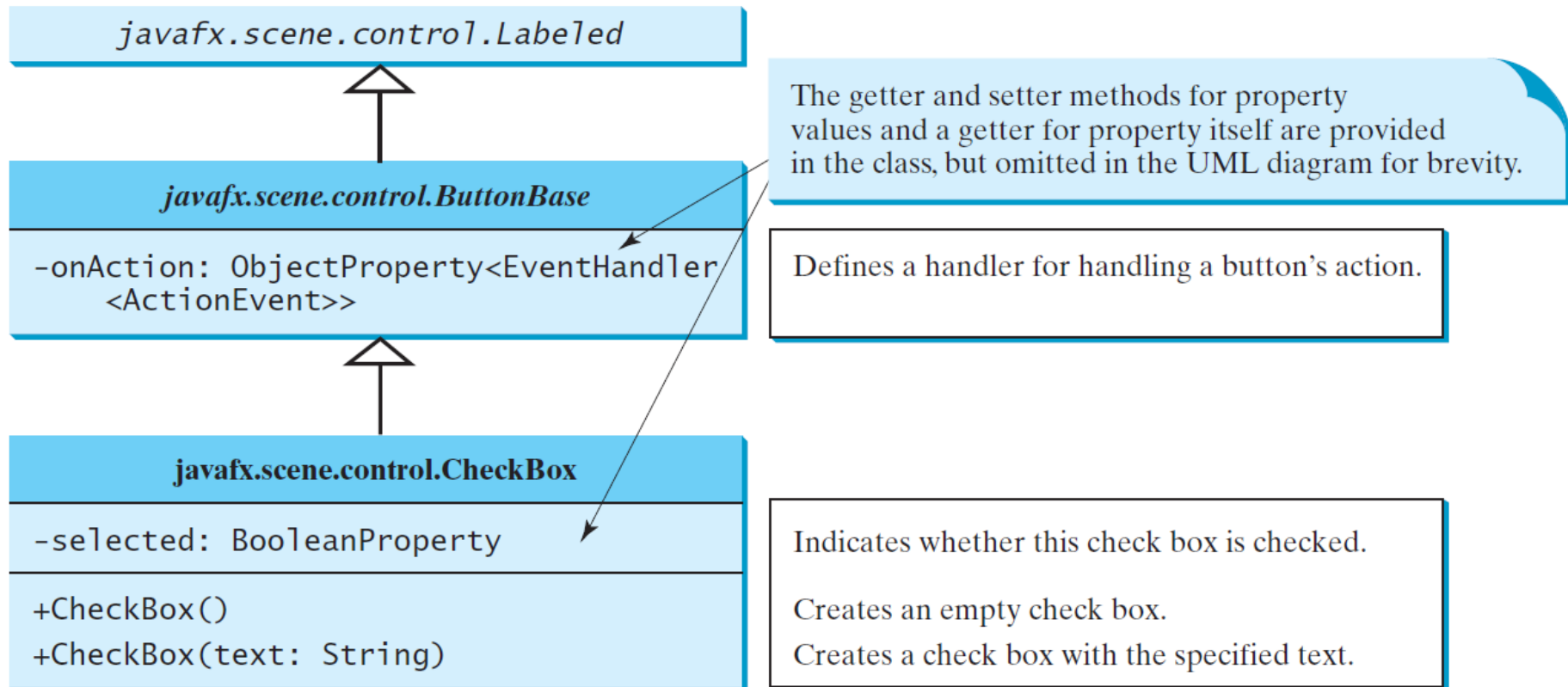Creates a button with the specified text and graphic.
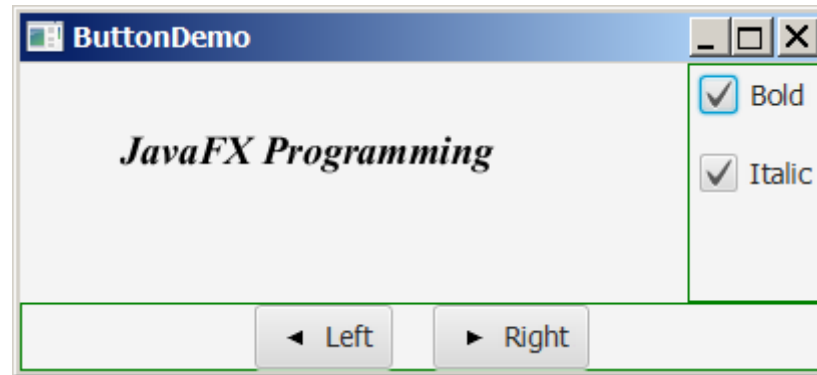
Button Example

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**, **textFill**, **contentDisplay** from **ButtonBase** and **Labeled**.

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

```
javafx.scene.control.ButtonBase
```
```
-onAction: ObjectProperty<EventHandler
    <ActionEvent>>
```

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.

Defines a handler for handling a button's action.

```
javafx.scene.control.CheckBox
```
```
-selected: BooleanProperty

+CheckBox()
+CheckBox(text: String)
```

Indicates whether this check box is checked.

Creates an empty check box.
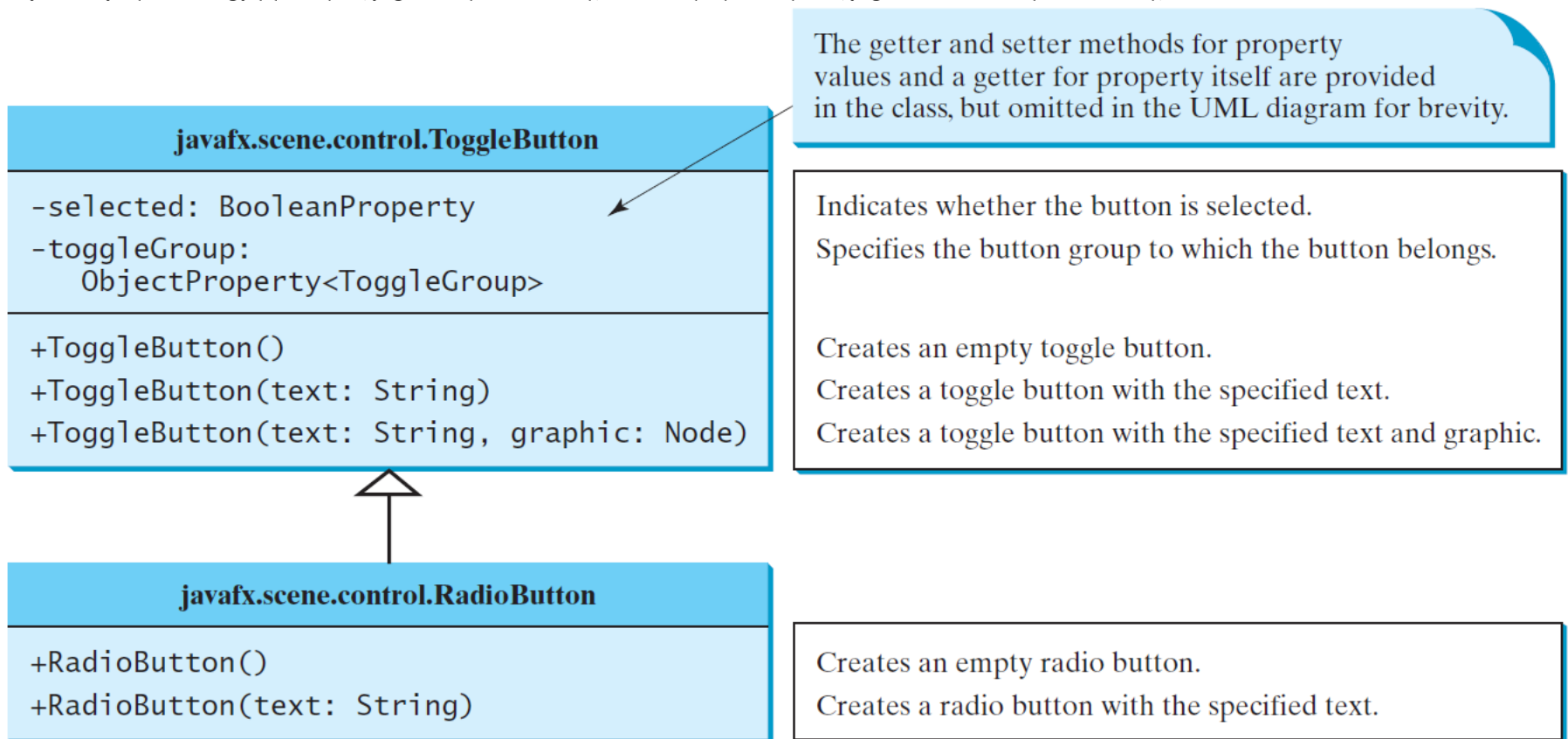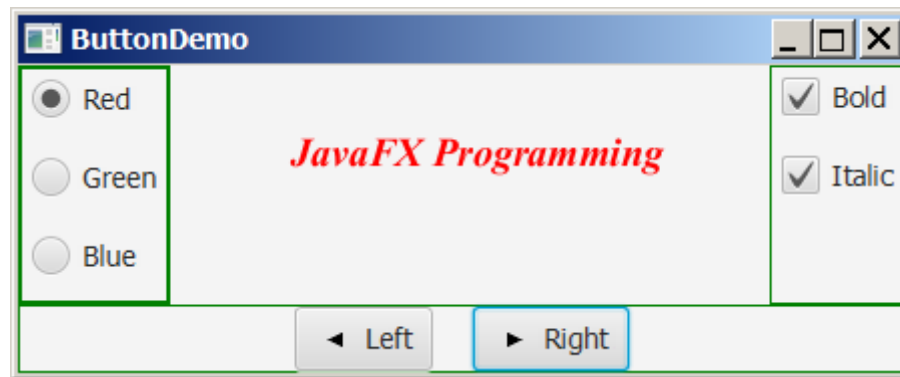Creates a check box with the specified text.

CheckBox Example

RadioButton

Radio buttons, also known as *option buttons*, enable you 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

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.

javafx.scene.control.ToggleButton

-selected: BooleanProperty
-toggleGroup: ObjectProperty\<ToggleGroup\>
+ToggleButton()
+ToggleButton(text: String)
+ToggleButton(text: String, graphic: Node)

Indicates whether the button is selected.
Specifies the button group to which the button belongs.

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

javafx.scene.control.RadioButton

+RadioButton()
+RadioButton(text: String)

Creates an empty radio button.
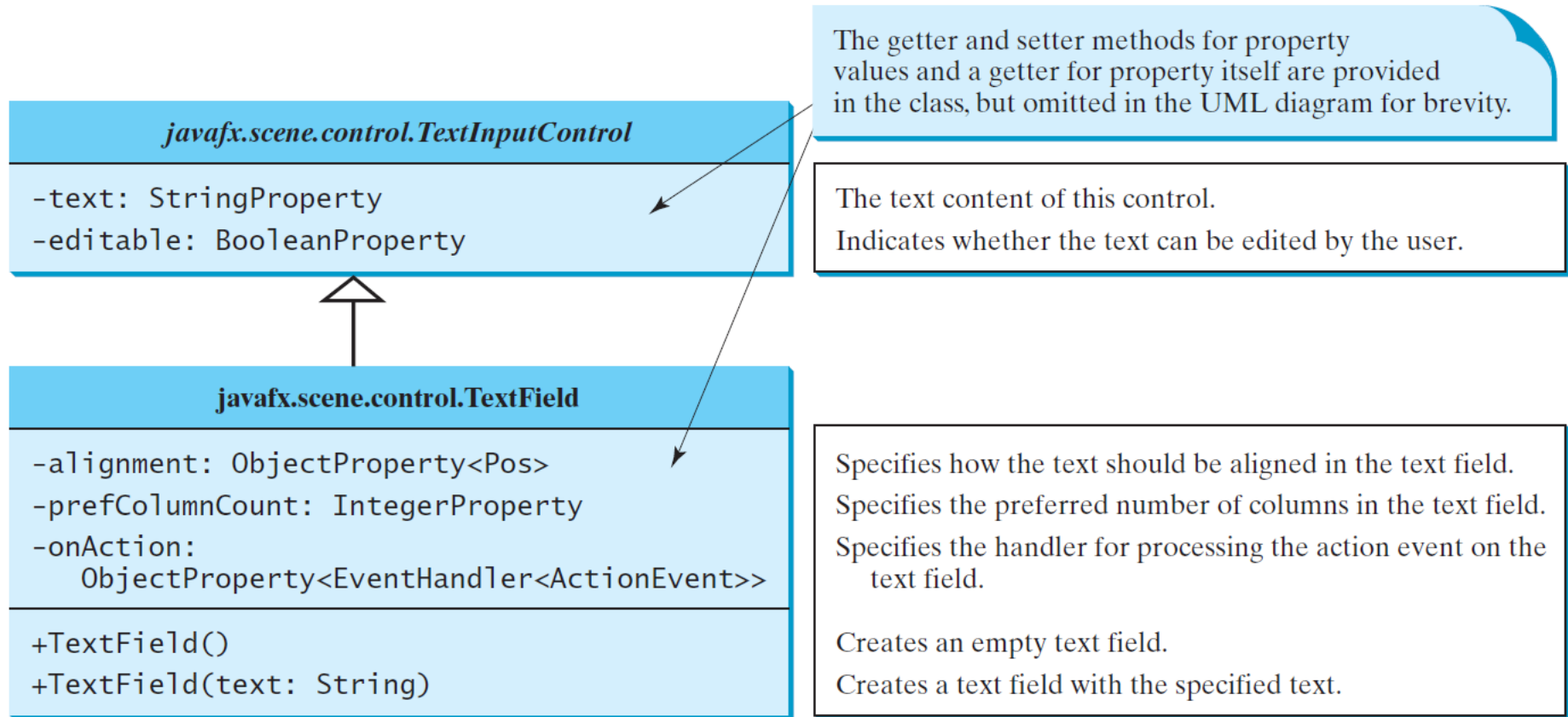Creates a radio button with the specified text.

RadioButton Example

RadioButtonDemo   Run
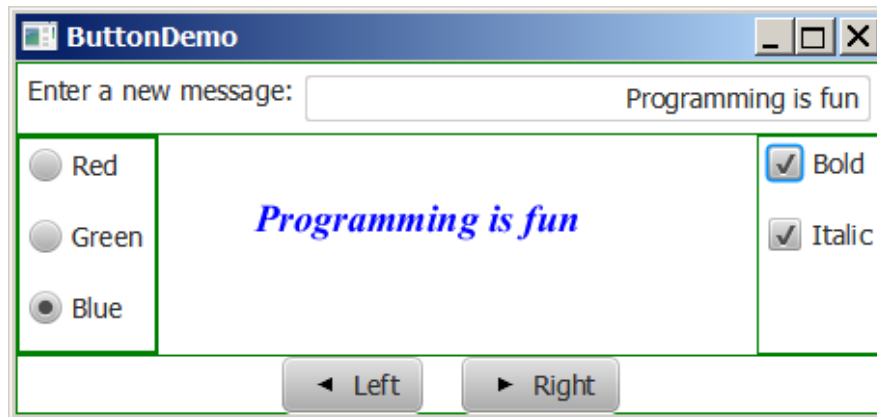
TextField

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

```
javafx.scene.control.TextInputControl

-text: StringProperty
-editable: BooleanProperty
```
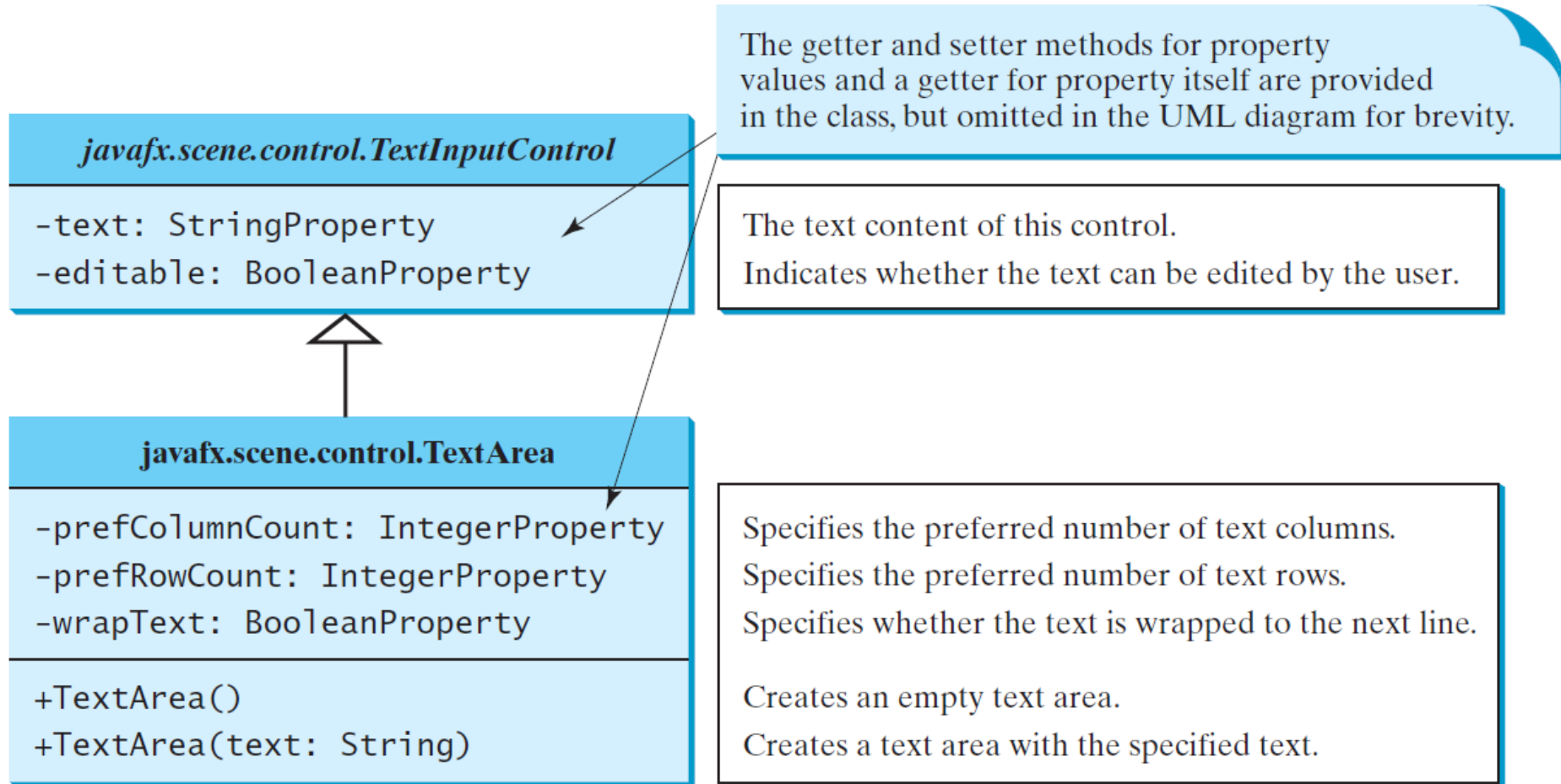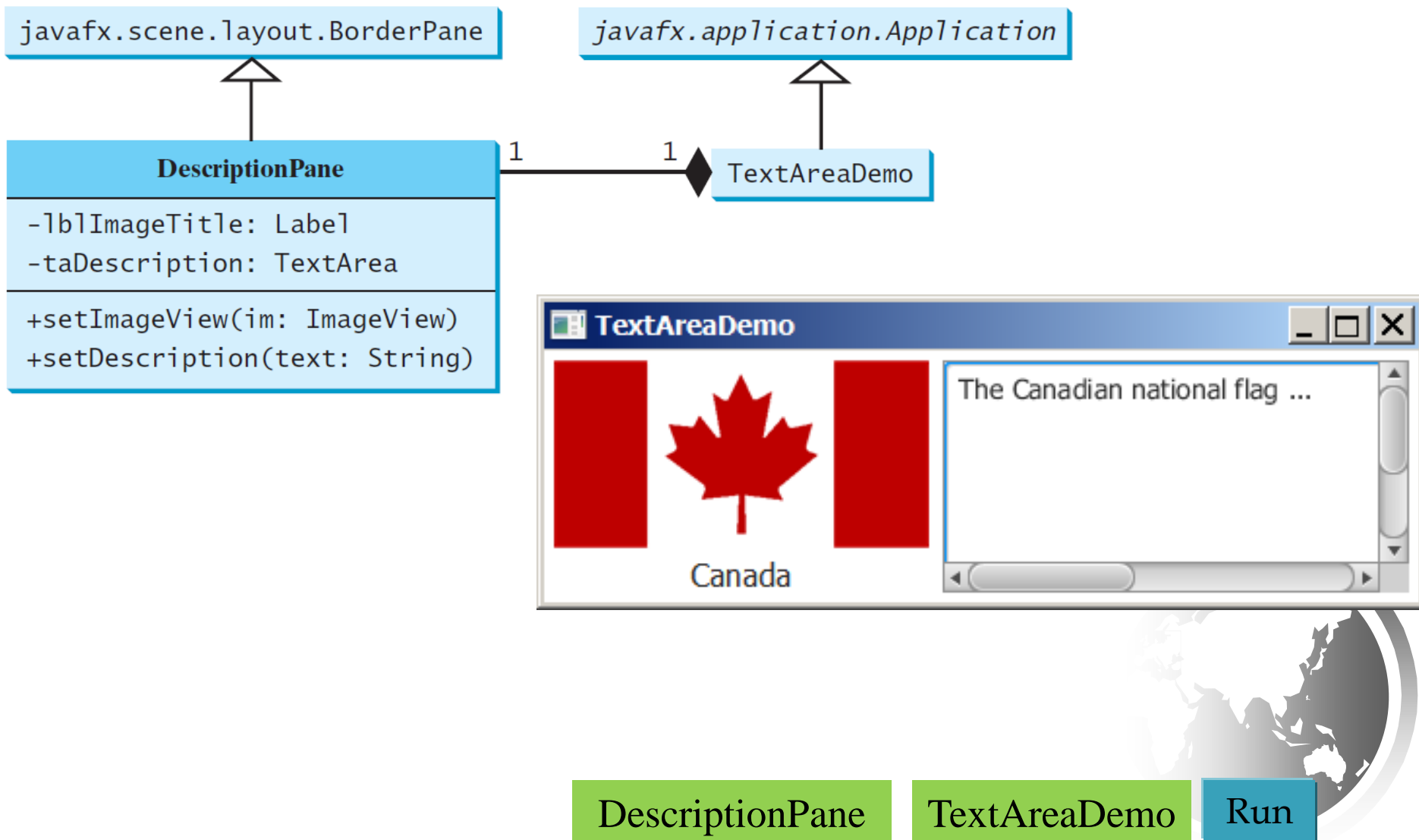
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

The text content of this control.

Indicates whether the text can be edited by the user.

```
javafx.scene.control.TextField

-alignment: ObjectProperty<Pos>
-prefColumnCount: IntegerProperty
-onAction:
   ObjectProperty<EventHandler<ActionEvent>>

+TextField()
+TextField(text: String)
```

Specifies how the text should be aligned in the text field.
Specifies the preferred number of columns in the text field.
Specifies the handler for processing the action event on the text field.

Creates an empty text field.
Creates a text field with the specified text.

TextField Example

TextArea

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

javafx.scene.control.TextInputControl	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.

javafx.scene.control.TextInputControl	
-text: StringProperty	The text content of this control.
-editable: BooleanProperty	Indicates whether the text can be edited by the user.

javafx.scene.control.TextArea	
-prefColumnCount: IntegerProperty	Specifies the preferred number of text columns.
-prefRowCount: IntegerProperty	Specifies the preferred number of text rows.
-wrapText: BooleanProperty	Specifies whether the text is wrapped to the next line.
+TextArea()	Creates an empty text area.
+TextArea(text: String)	Creates a text area with the specified text.

TextArea Example

javafx.scene.layout.BorderPane

javafx.application.Application

DescriptionPane

-lblImageTitle: Label
-taDescription: TextArea

+setImageView(im: ImageView)
+setDescription(text: String)

1   1   TextAreaDemo

TextAreaDemo

The Canadian national flag ...

Canada

DescriptionPane   TextAreaDemo   Run

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.

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.

javafx.scene.control.ComboBoxBase<T>

```
-value: ObjectProperty<T>
-editable: BooleanProperty
-onAction:
   ObjectProperty<EventHandler<ActionEvent>>
```

The value selected in the combo box.

Specifies whether the combo box allows user input.

Specifies the handler for processing the action event.

javafx.scene.control.ComboBox<T>

```
-items: ObjectProperty<ObservableList<T>>
-visibleRowCount: IntegerProperty

+ComboBox()
+ComboBox(items: ObservableList<T>)
```

The items in the combo box popup.

The maximum number of visible rows of the items in the combo box popup.

Creates an empty combo box.

Creates a combo box with the specified items.

ComboBox Example

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

CentralBoxDemo

Run

ListView

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

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: Using ListView

This example gives a program that lets users select countries in a list and display the flags of the selected countries in the labels.

ListViewDemo   Run

ScrollBar

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

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.

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.

Scroll Bar Properties

Minimal value

Track

Maximal value

Thumb

Left button

Right button
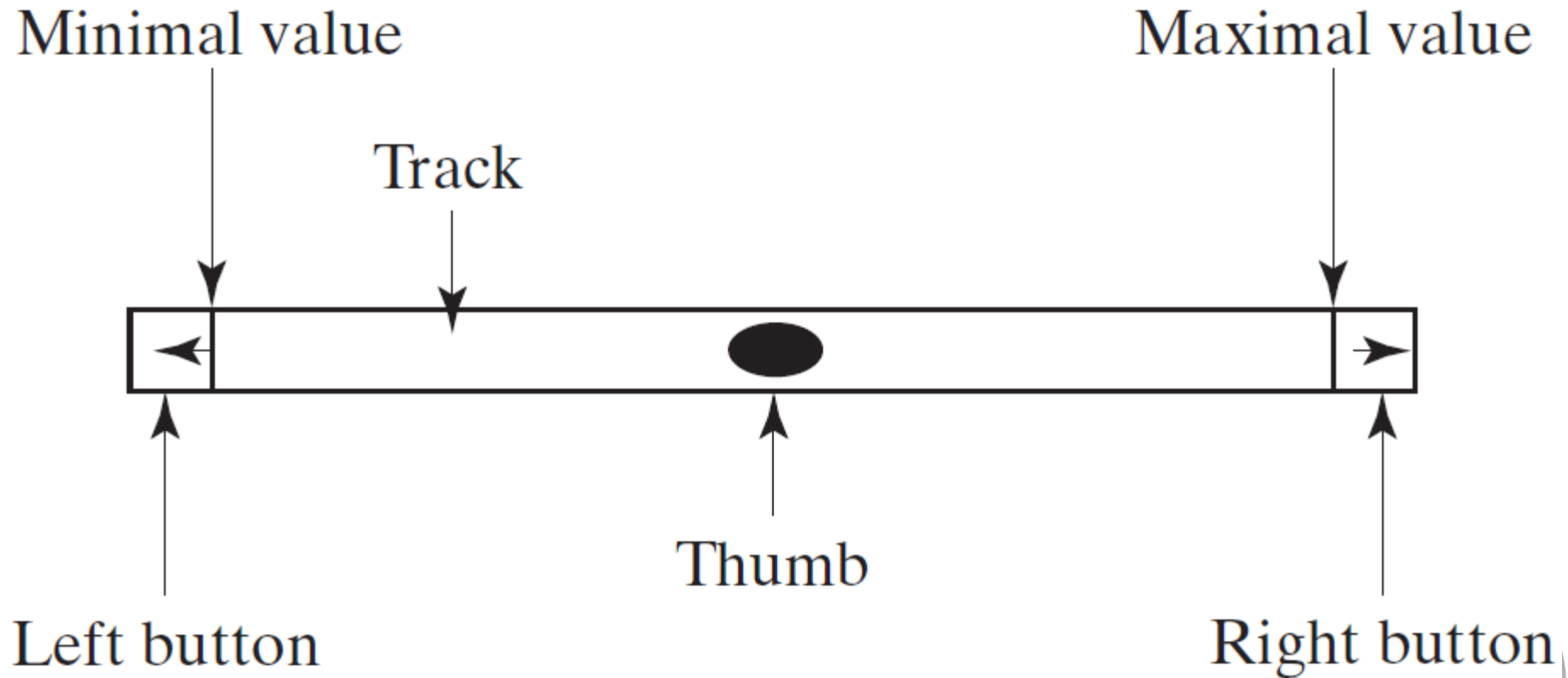
Example: Using Scrollbars

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.

ScrollBarDemo

JavaFX Programming
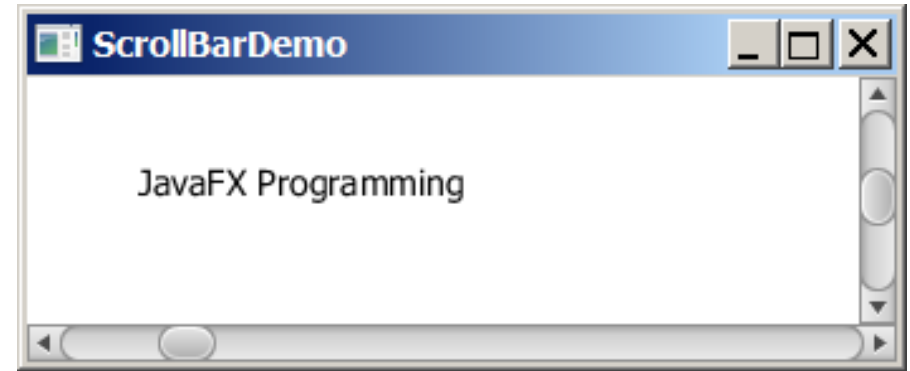
ScrollBarDemo   Run

Slider

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

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.

| 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.

Example: Using Sliders

Rewrite the preceding program using the sliders to control a message displayed on a panel instead of using scroll bars.
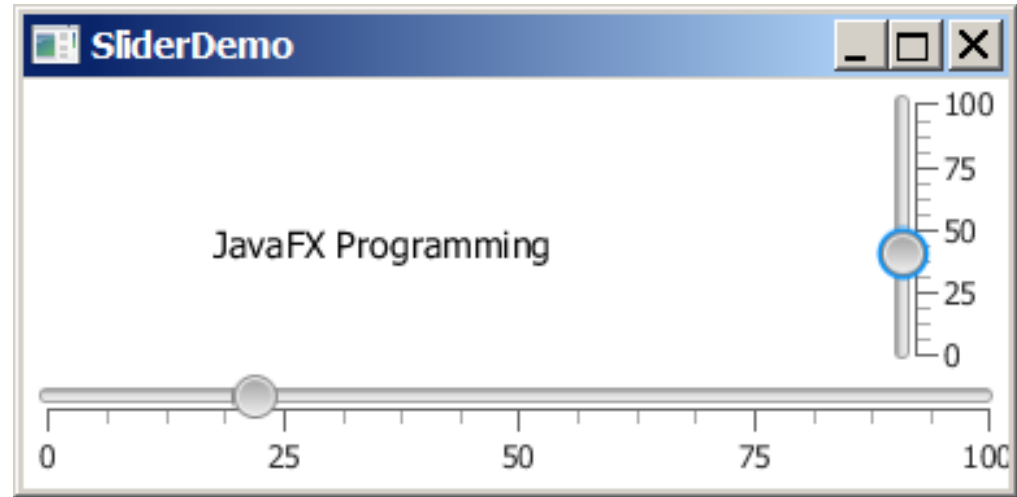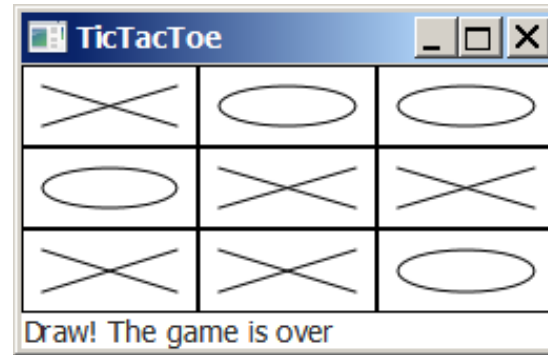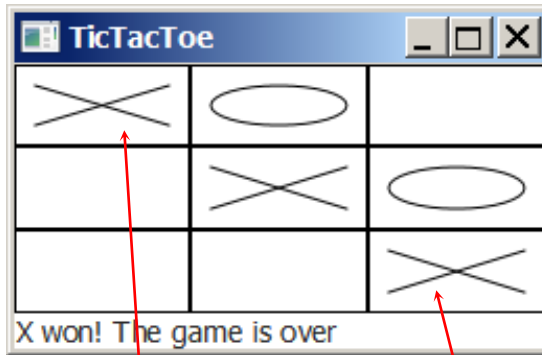
Case Study: Bounce Ball

Listing 15.17 gives a program that displays a bouncing ball. You can add a slider to control the speed of the ball movement.
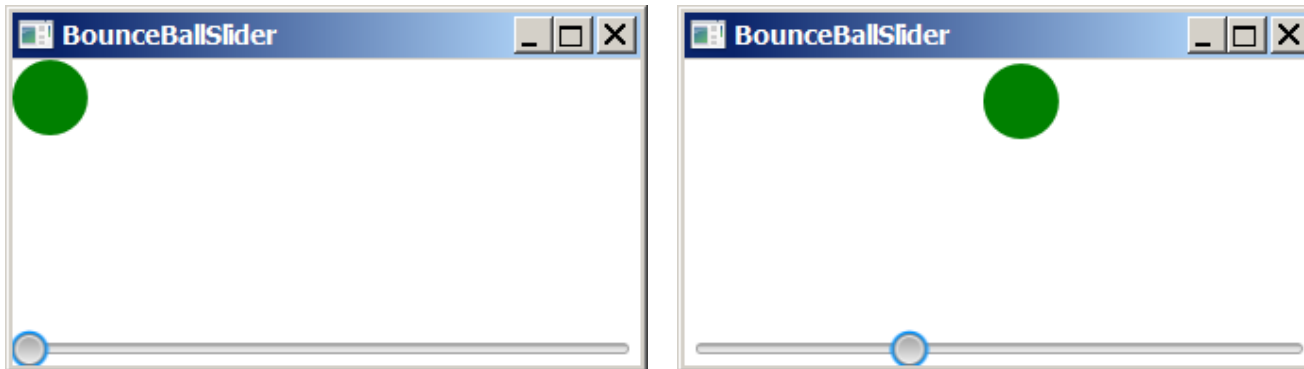
SliderDemo   Run

Case Study: TicTacToe


```
javafx.scene.layout.Pane
```

Cell	
-token: char	Token used in the cell (default: ' ').
+getToken(): char	Returns the token in the cell.
+setToken(token: char): void	Sets a new token in the cell.
-handleMouseClick(): void	Handles a mouse click event.

Case Study: TicTacToe, cont.

| Cell | *javafx.application.Application* |

9

1

TicTacToe

-whoseTurn: char
-cell: Cell[][]
-lblStatus: Label

+TicTacToe()
+isFull(): boolean
+isWon(token: char): boolean

Indicates which player has the turn, initially X.
A 3 × 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.

TicTacToe   Run

Media

You 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.

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.

javafx.scene.media.Media

```
-duration: ReadOnlyObjectProperty
    <Duration>
-width: ReadOnlyIntegerProperty
-height: ReadOnlyIntegerProperty

+Media(source: String)
```

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.

MediaPlayer

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

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.

javafx.scene.media.MediaPlayer

```
-autoPlay: BooleanProperty
-currentCount: ReadOnlyIntegerProperty
-cycleCount: IntegerProperty
-mute: BooleanProperty
-volume: DoubleProperty
-totalDuration:
    ReadOnlyObjectProperty<Duration>

+MediaPlayer(media: Media)
+play(): void
+pause(): void
+seek(): void
```

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.

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.

javafx.scene.media.MediaView	
-x: DoubleProperty	Specifies the current x-coordinate of the media view.
-y: DoubleProperty	Specifies the current y-coordinate of the media view.
-mediaPlayer: ObjectProperty<MediaPlayer>	Specifies a media player for the media view.
-fitWidth: DoubleProperty	Specifies the width of the view for the media to fit.
-fitHeight: DoubleProperty	Specifies the height of the view for the media to fit.
+MediaView()	Creates an empty media view.
+MediaView(mediaPlayer: MediaPlayer)	Creates a media view with the specified media player.

Example: Using Media

This example displays a video in a view. You can use the play/pause button to play or pause the video and use the rewind button to restart the video, and use the slider to control the volume of the audio.

media: Media ◇— mediaPlayer: MediaPlayer ◇— mediaView: MediaView

MediaDemo   Run
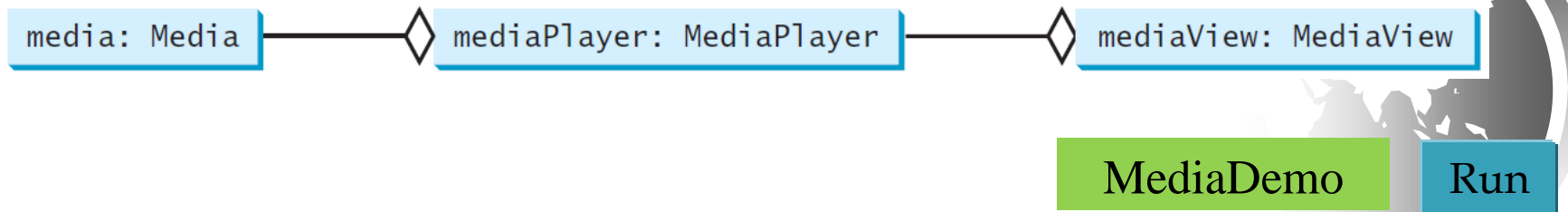
Case Study: National Flags and Anthems

This case study presents a program that displays a nation's flag and plays its anthem.

FlagAnthem

Run