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).

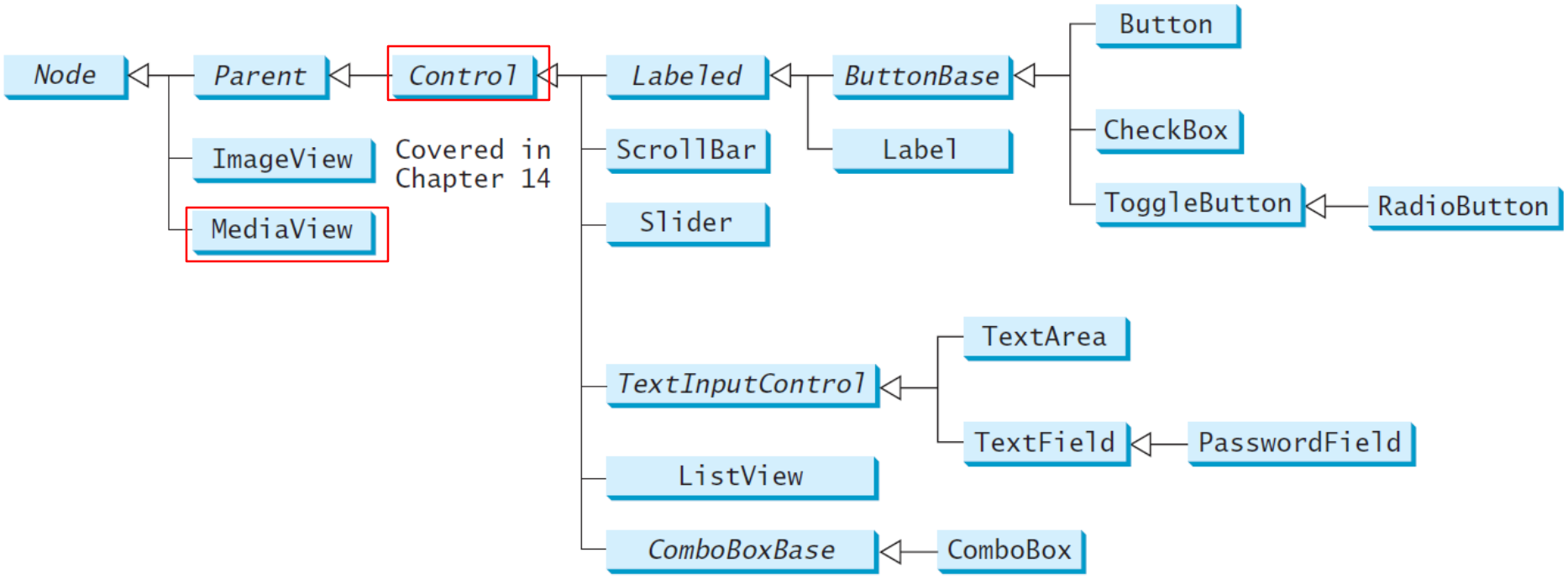
To select a range of values using **Slider** and explore differences between **ScrollBar** and **Slider** (§16.11).

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).

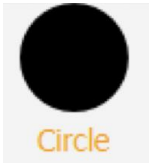
Frequently Used UI Controls



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.



javafx.scene.control.Labeled

```
-alignment: ObjectProperty<Pos>
-contentDisplay:
   ObjectProperty<ContentDisplay>
-graphic: ObjectProperty<Node>
-graphicTextGap: DoubleProperty
-textFill: 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.

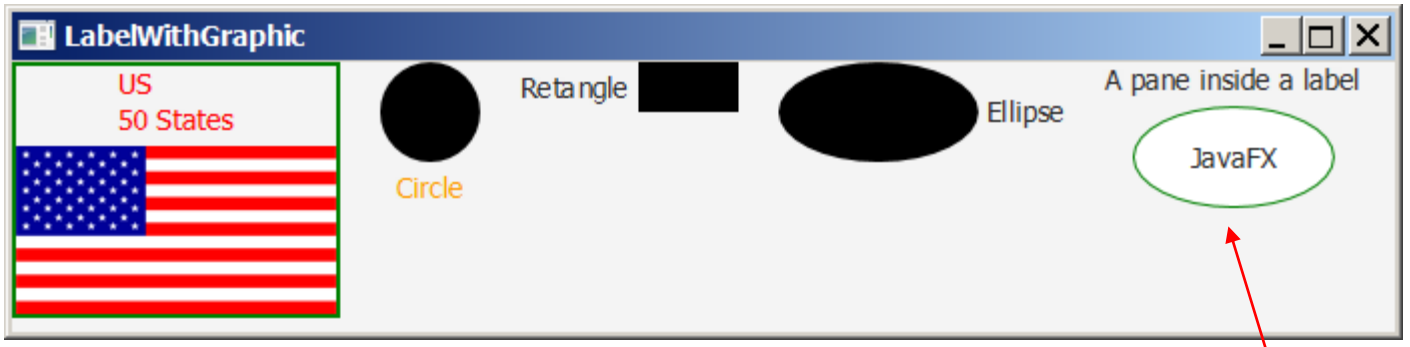
A text for the labeled.

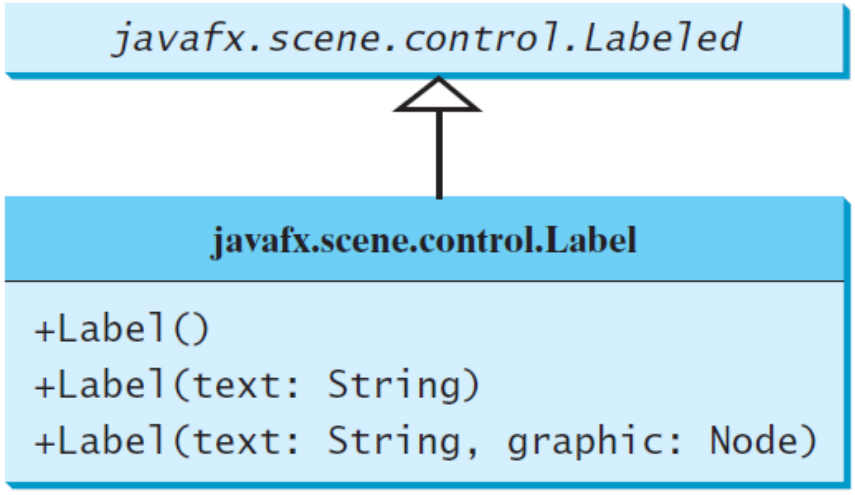
Whether text should be underlined.

Whether text should be wrapped if the text exceeds the width.

Label

The Label class defines labels.






```
Label lb5 = new Label("A pane inside a label", stackPane);
```

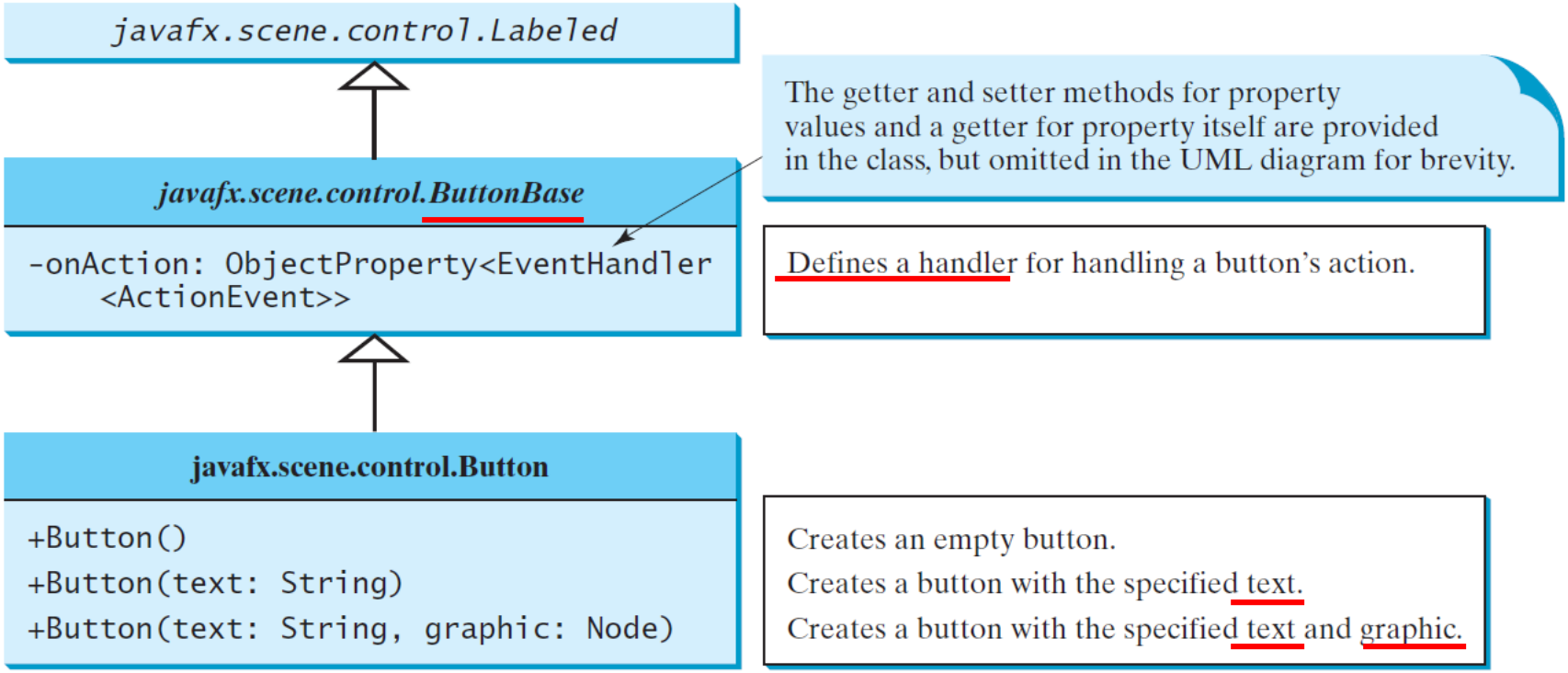
Creates an empty label.	
Creates a label with the specified text.	
Creates a label with the specified text and graphic.	

LabelWithGraphic

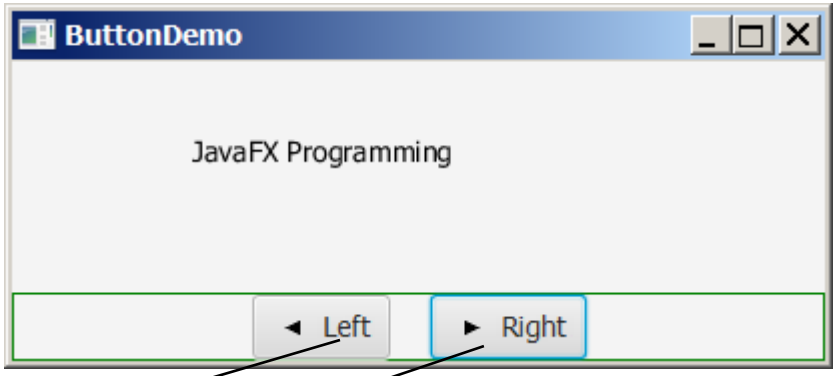
Run

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.



Button Example



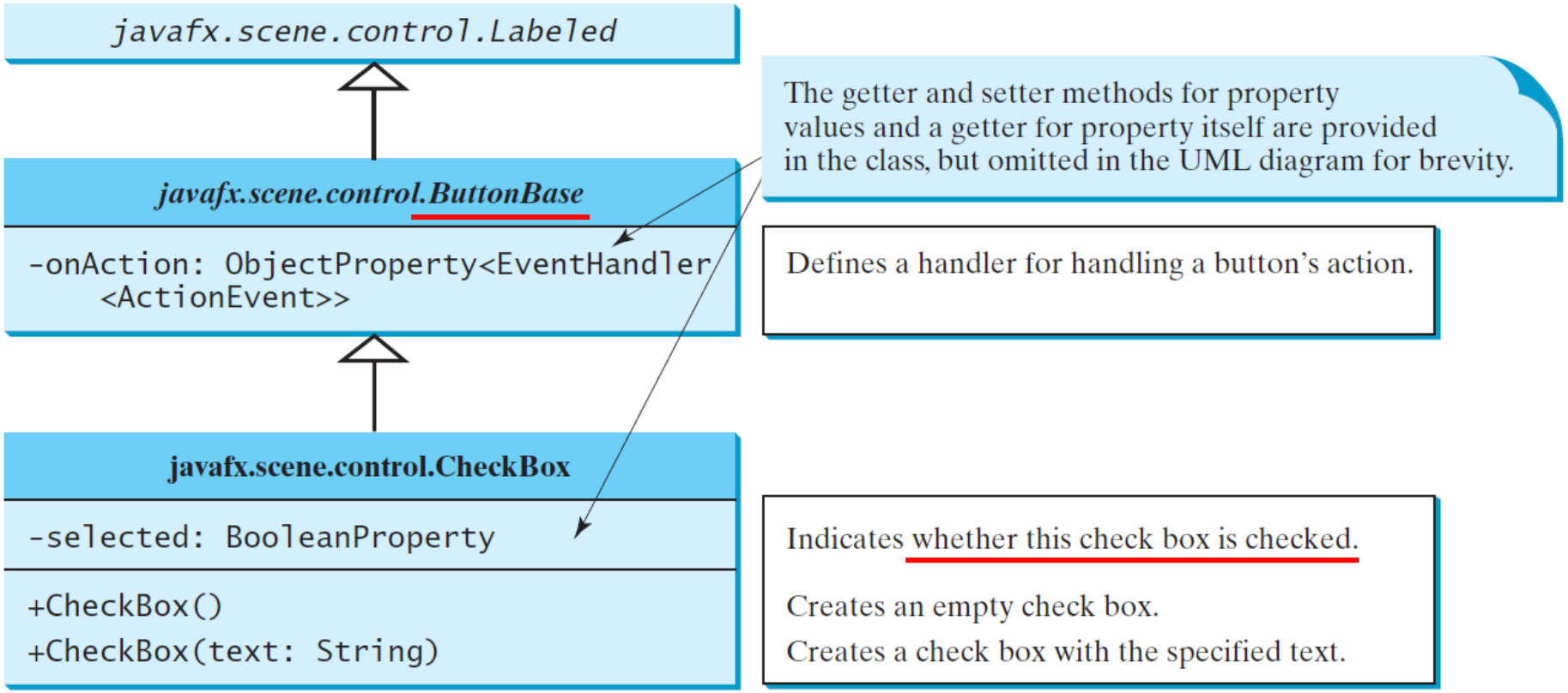

```
btLeft.setOnAction(e -> text.setX(text.getX() - 10));
btRight.setOnAction(e -> text.setX(text.getX() + 10));
```

ButtonDemo

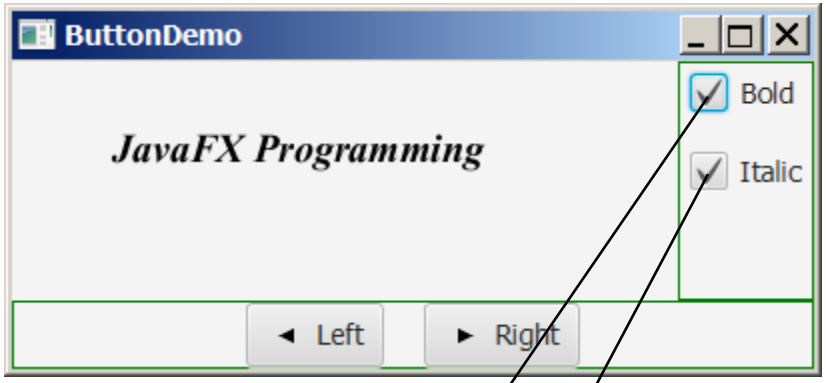
Run

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**.



CheckBox Example



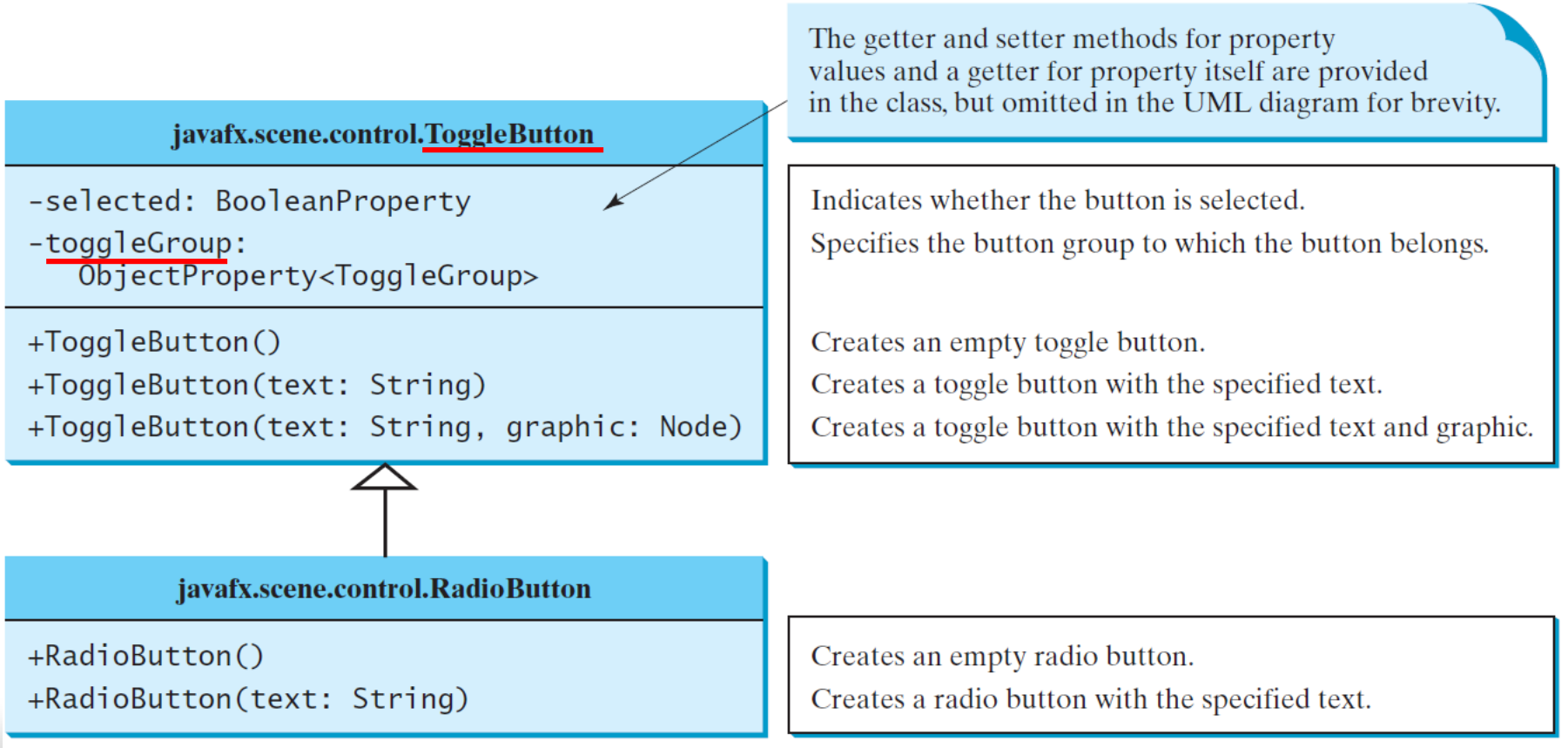

```
CheckBox chkBold = new CheckBox("Bold");
CheckBox chkItalic = new CheckBox("Italic");

EventHandler<ActionEvent> handler = e -> { *** }
chkBold.setOnAction(handler);
chkItalic.setOnAction(handler);
```

CheckBoxDemo

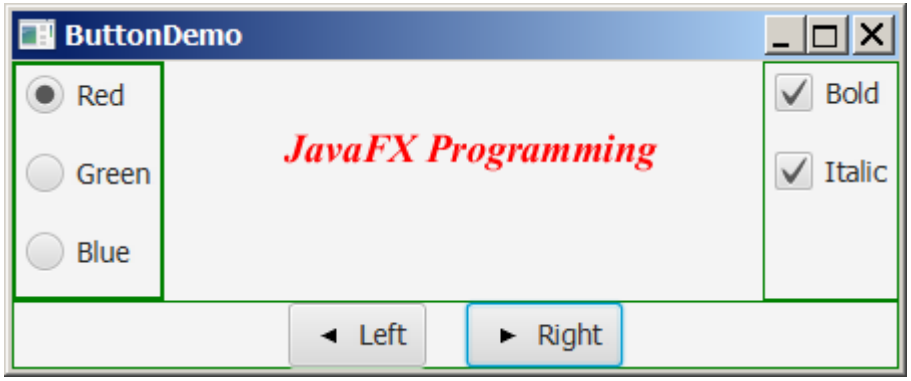
Run

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 is either filled (if selected) or blank (if not selected).



RadioButton Example



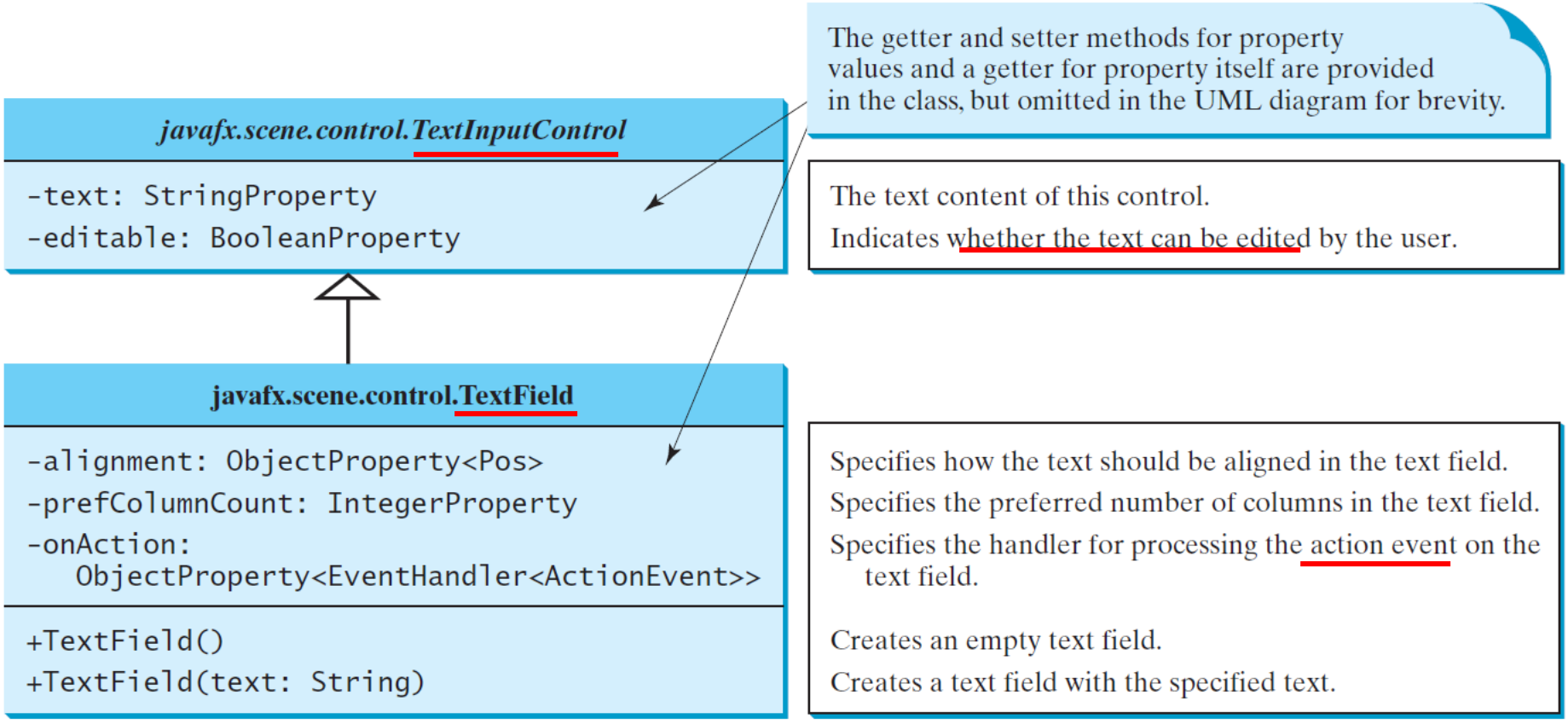

```
RadioButton rbRed = new RadioButton("Red");
RadioButton rbGreen = new RadioButton("Green");
RadioButton rbBlue = new RadioButton("Blue");

ToggleGroup group = new ToggleGroup();
  rbRed.setToggleGroup(group);
  rbGreen.setToggleGroup(group);
  rbBlue.setToggleGroup(group);
```

RadioButtonDemo

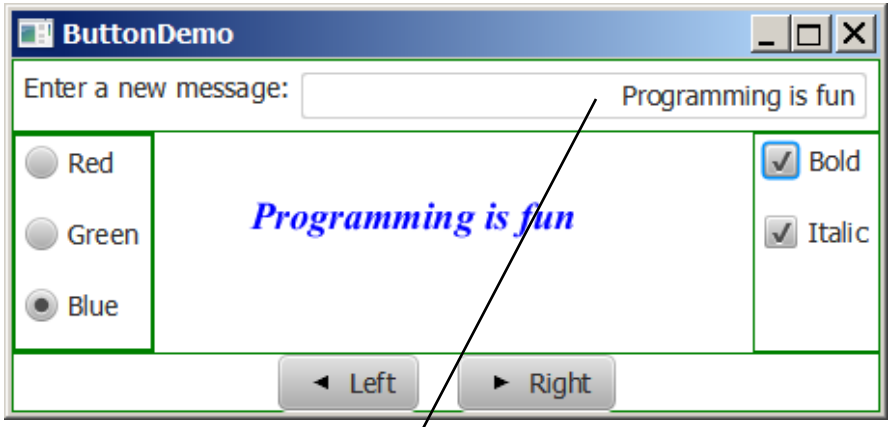
Run

TextField

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



TextField Example



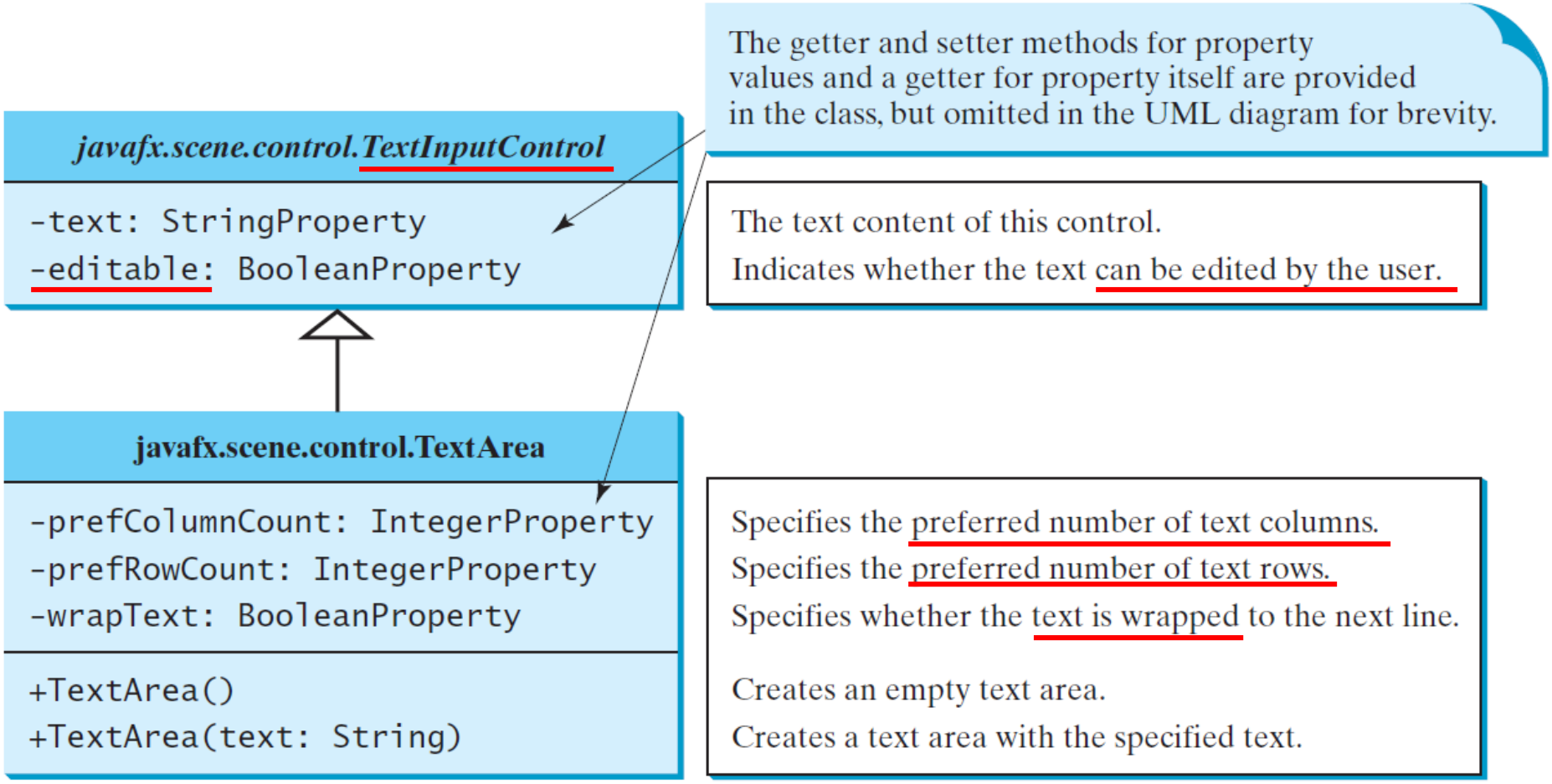

```
TextField tf = new TextField();
tf.setAlignment(Pos.BOTTOM_RIGHT);
  ***
tf.setOnAction(e -> text.setText(tf.getText()));
```

TextFieldDemo

Run

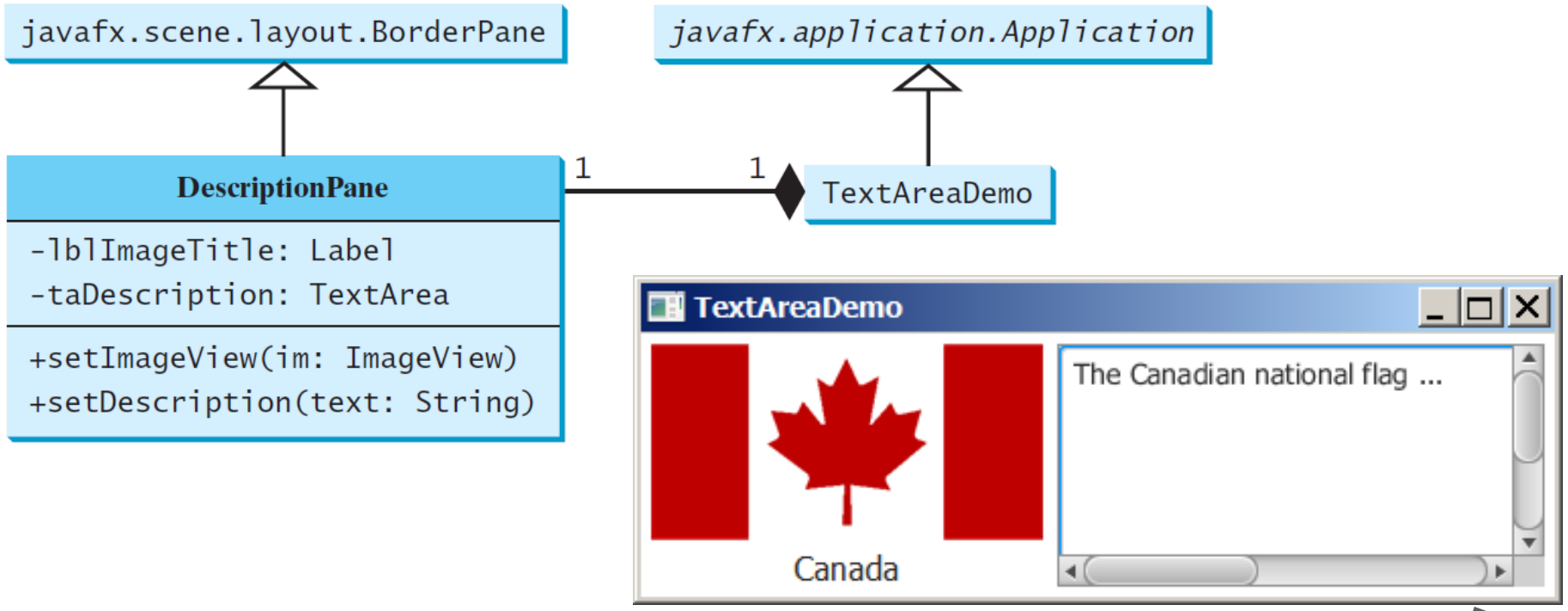
TextArea

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



```
private TextArea taDescription = new TextArea();

taDescription.setWrapText(true);
taDescription.setEditable(false);

// Create a scroll pane to hold the text area
ScrollPane scrollPane = new ScrollPane(taDescription);
```

TextArea Example



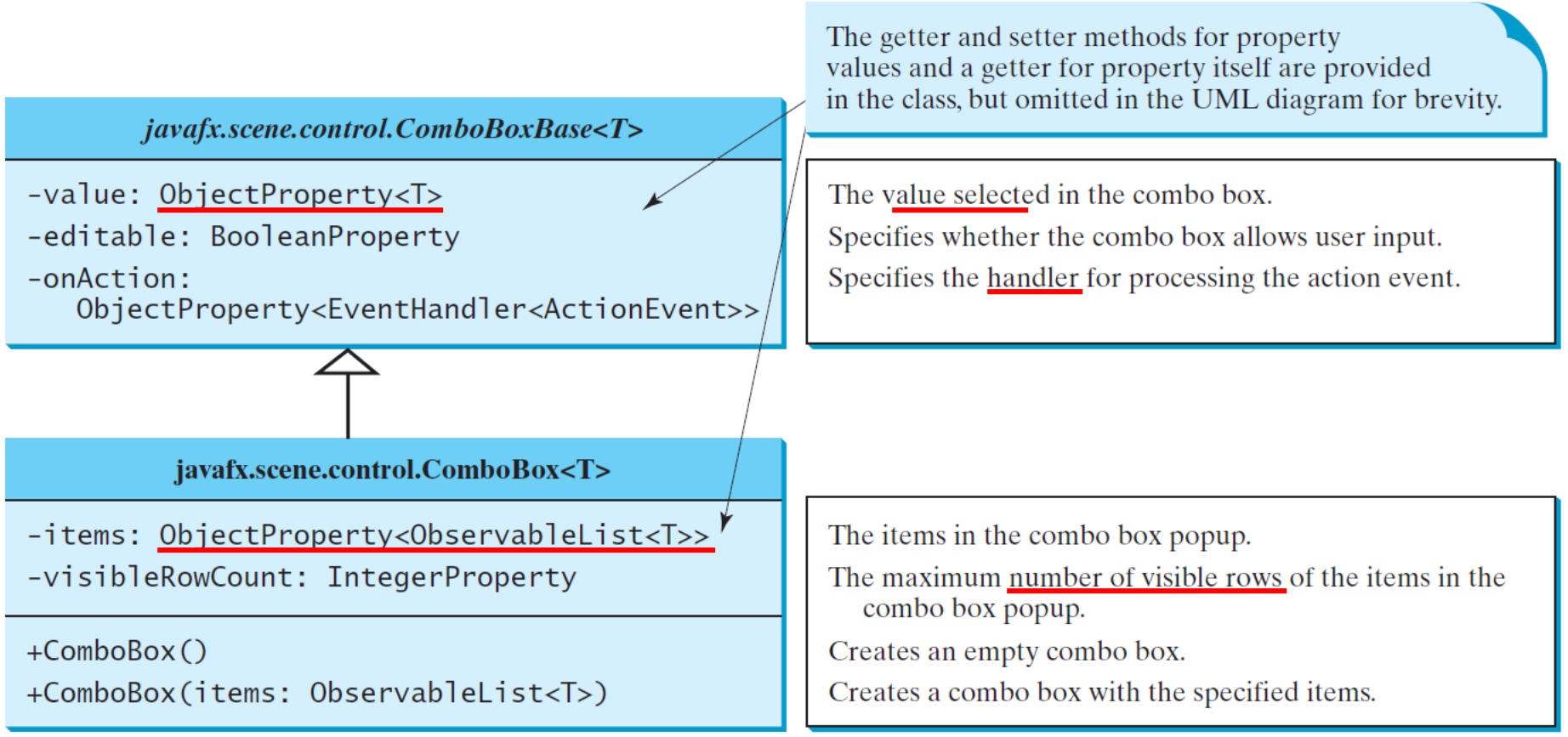
DescriptionPane

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.



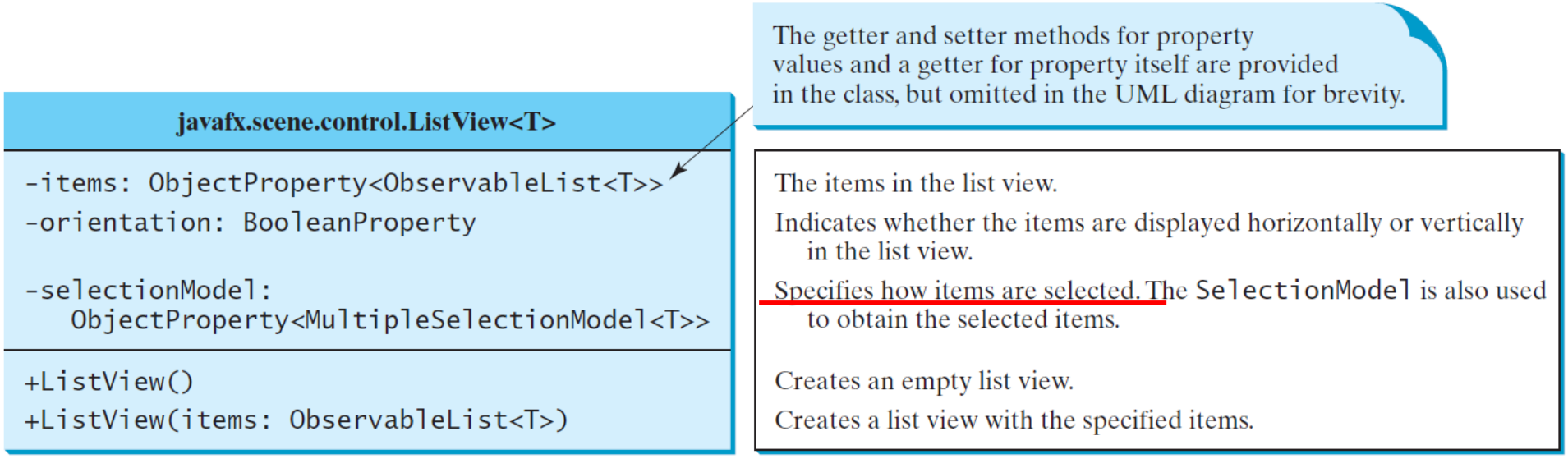
ComboBox Example

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

```
private String[] flagTitles = {"Canada", "China", "Denmark",
   "France", "Germany", "India", "Norway", "United Kingdom",
   "United States of America"};
***
private ComboBox<String> cbo = new ComboBox<>();
***
ObservableList<String> items =
     FXCollections.observableArrayList(flagTitles);
cbo.getItems().addAll(items);
***
// Display the selected country
cbo.setOnAction(e -> setDisplay(items.indexOf(cbo.getValue())));
```

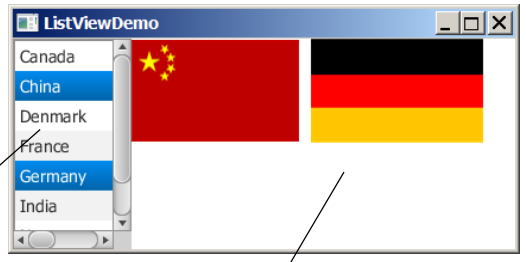
ListView

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.**



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.




```
ListView<String> lv = new ListView<>
    (FXCollections.observableArrayList(flagTitles));
lv.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
lv.getSelectionModel().selectedItemProperty().addListener(
    ov -> {
      imagePane.getChildren().clear();
      for (Integer i: lv.getSelectionModel().getSelectedIndices()) {
        imagePane.getChildren().add(ImageViews[i]);
          } // display only the selected flags
      });
```

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
-max: DoubleProperty
-min: DoubleProperty
-unitIncrement: DoubleProperty

-value: DoubleProperty
-visibleAmount: DoubleProperty
-orientation: ObjectProperty<Orientation>

+ScrollBar()
+increment()
+decrement()
```

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).

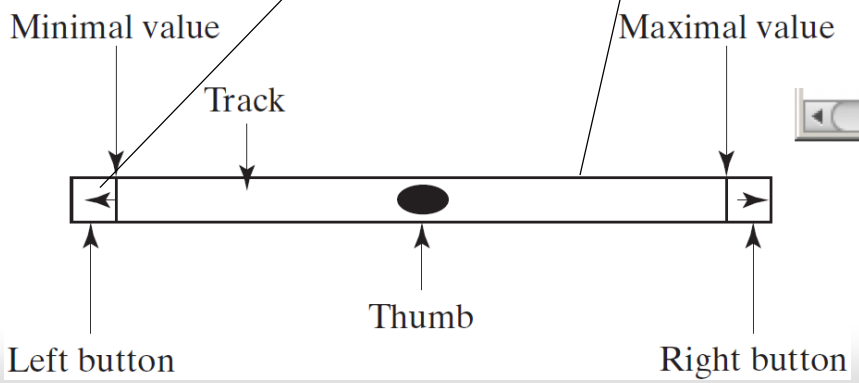
The width of the scroll bar (default: 15).

Specifies the orientation of the scroll bar (default: `HORIZONTAL`).

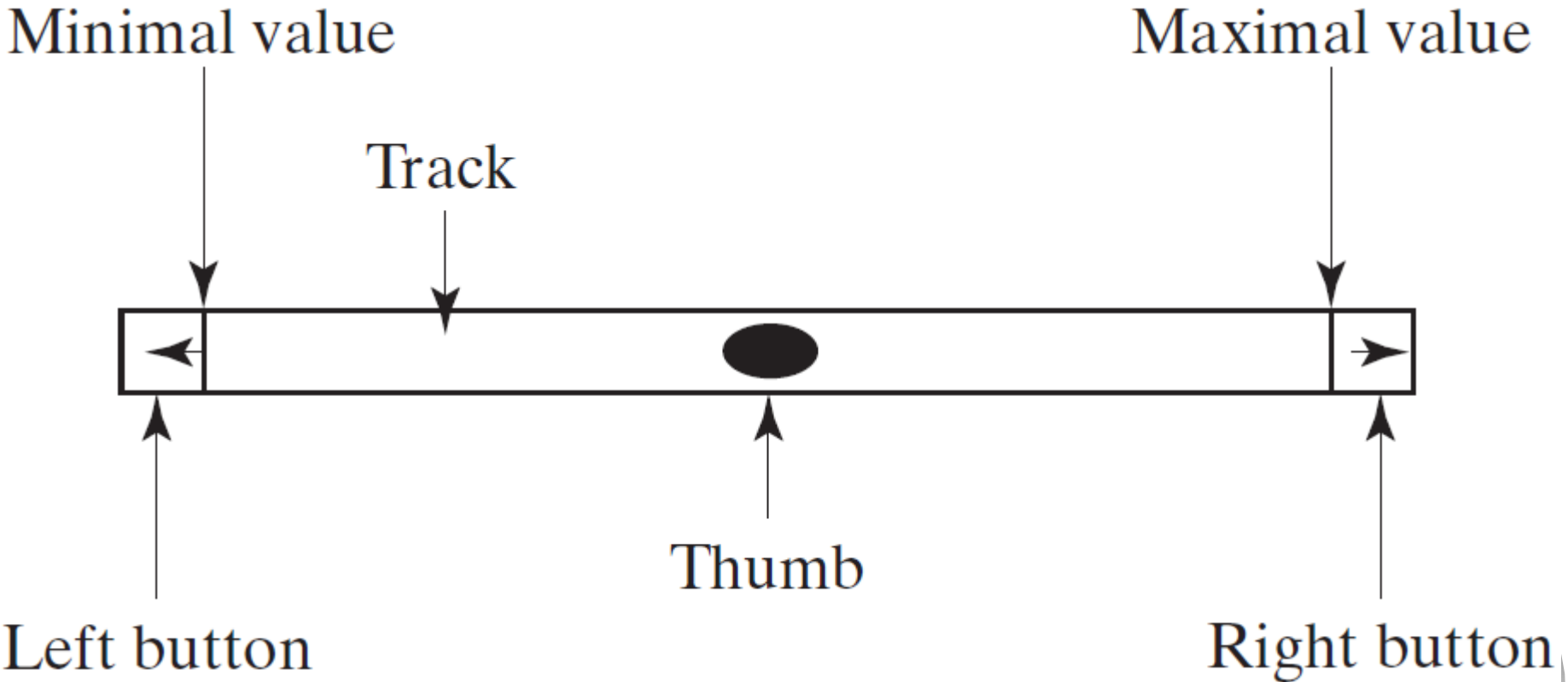
Creates a default horizontal scroll bar.

Increments the value of the scroll bar by `unitIncrement`.

Decrements the value of the scroll bar by `unitIncrement`.

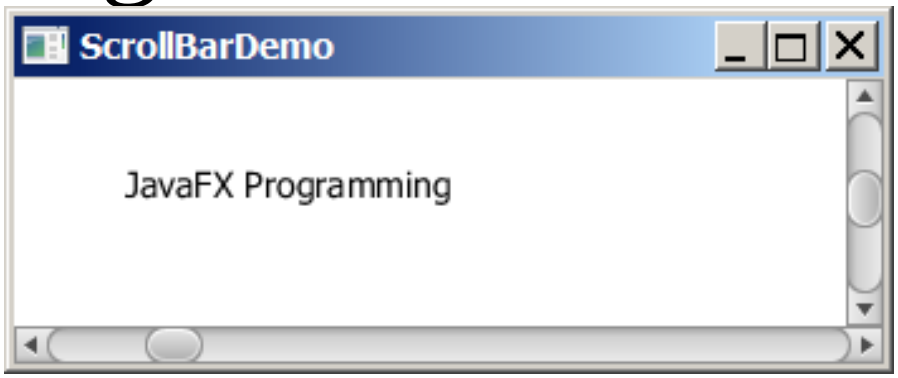


Scroll Bar Properties



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.

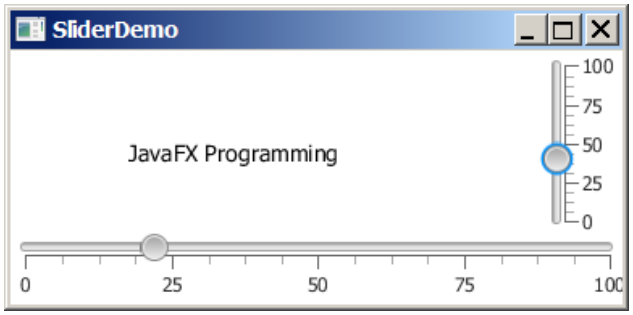



```
BorderPane pane = new BorderPane();
  pane.setCenter(paneForText);
  pane.setBottom(sbHorizontal);
  pane.setRight(sbVertical);

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

sbVertical.valueProperty().addListener(ov -> ***
```

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
-max: DoubleProperty
-min: DoubleProperty
-value: DoubleProperty
-orientation: ObjectProperty<Orientation>
-majorTickUnit: DoubleProperty
-minorTickCount: IntegerProperty
-showTickLabels: BooleanProperty
-showTickMarks: BooleanProperty
```

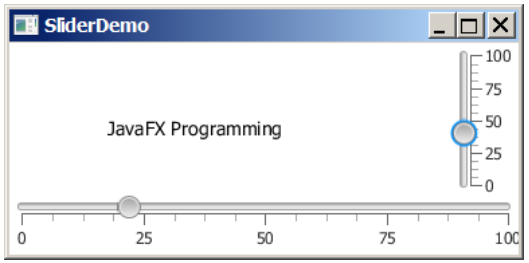
```
+Slider()
+Slider(min: double, max: double,
   value: double)
```

The amount to adjust the slider if the track of the bar is clicked (default: 10).
The maximum value represented by this slider (default: 100).
The minimum value represented by this slider (default: 0).
Current value of the slider (default: 0).
Specifies the orientation of the slider (default: HORIZONTAL).
The unit distance between major tick marks.
The number of minor ticks to place between two major ticks.
Specifies whether the labels for tick marks are shown.
Specifies whether the tick marks are shown.

Creates a default horizontal slider.
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.



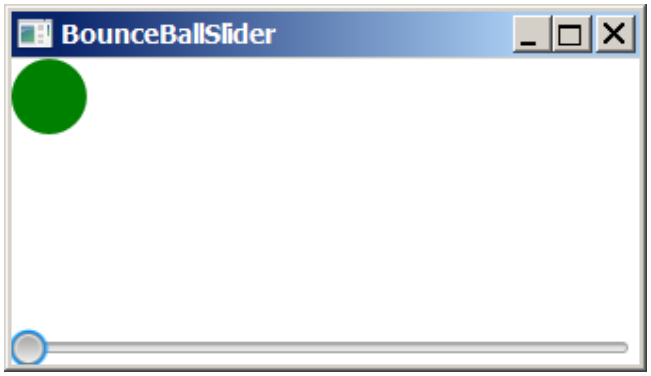

```
Slider slHorizontal = new Slider();
  slHorizontal.setShowTickLabels(true);
  slHorizontal.setShowTickMarks(true);

Slider slVertical = new Slider();
slVertical.setOrientation(Orientation.VERTICAL);

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

slVertical.valueProperty().addListener(ov -> ***
```

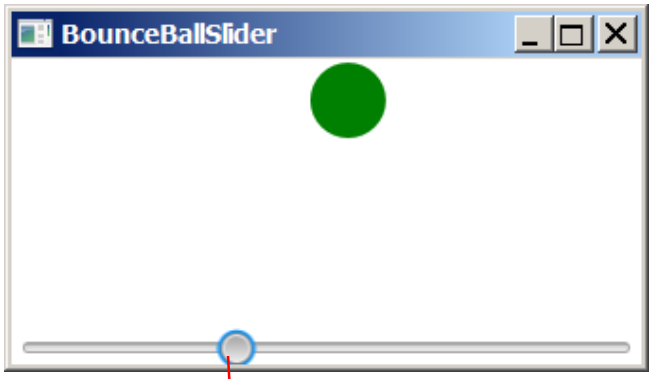
SliderDemo

Run

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.



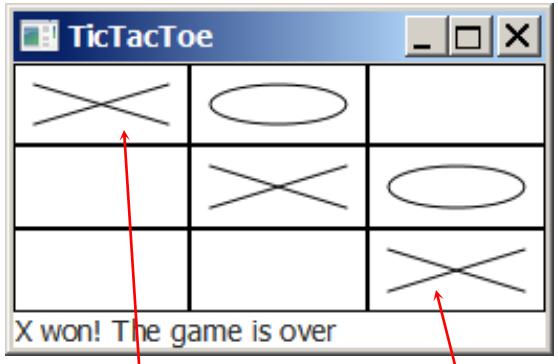


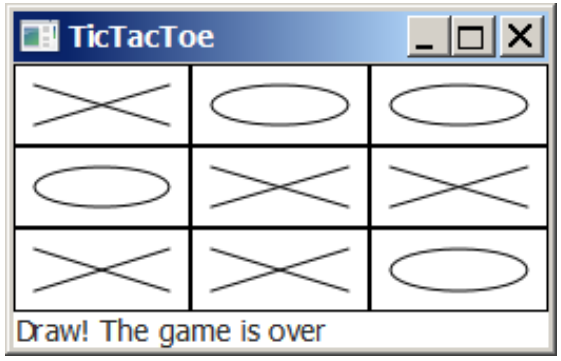

```
BallPane ballPane = new BallPane();
  Slider slSpeed = new Slider();
  slSpeed.setMax(20);
  ballPane.rateProperty().bind(slSpeed.valueProperty());
```

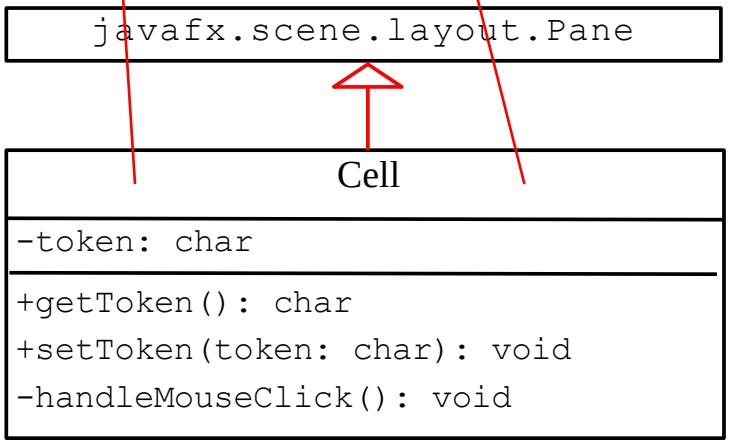
BounceBallSlider

Run

Case Study: TicTacToe







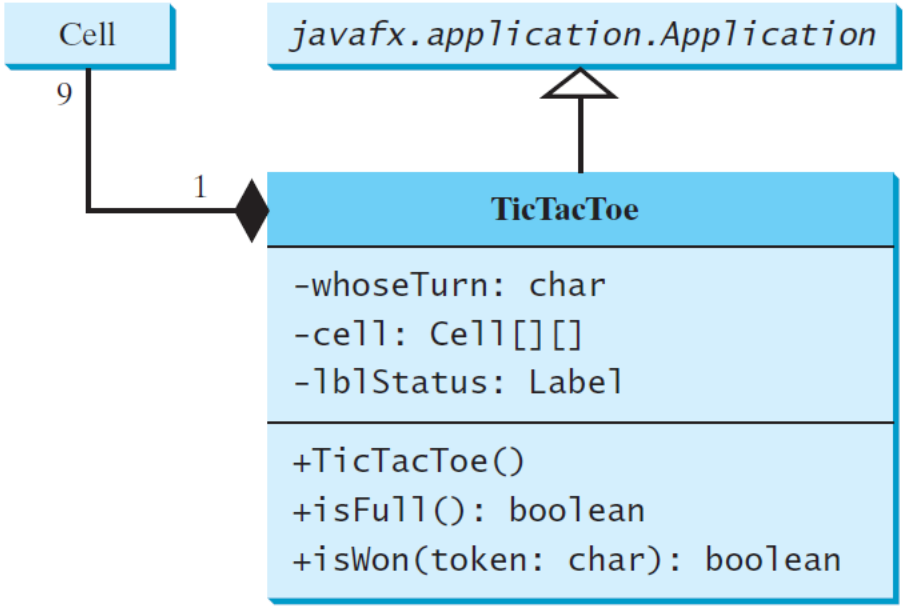
Token used in the cell (default: ' ').

Returns the token in the cell.

Sets a new token in the cell.

Handles a mouse click event.

Case Study: TicTacToe, cont.



Indicates which player has the turn, initially X.
A 3 × 3, two-dimensional array for cells.
A label to display game status.

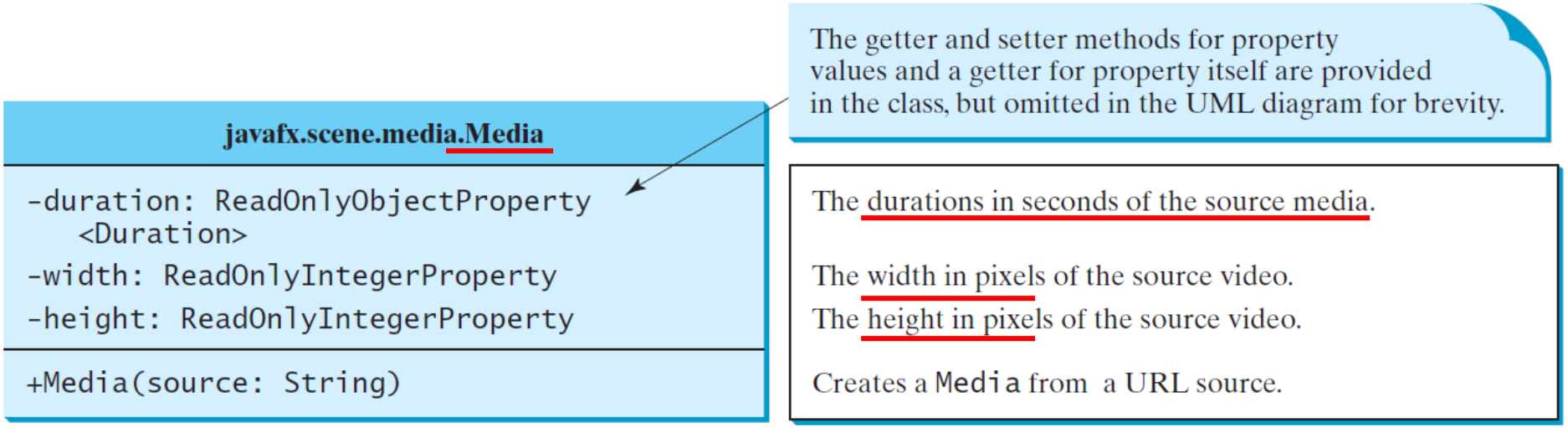
Constructs the TicTacToe user interface.
Returns true if all cells are filled.
Returns true if a player with the specified token has won.

TicTacToe

Run

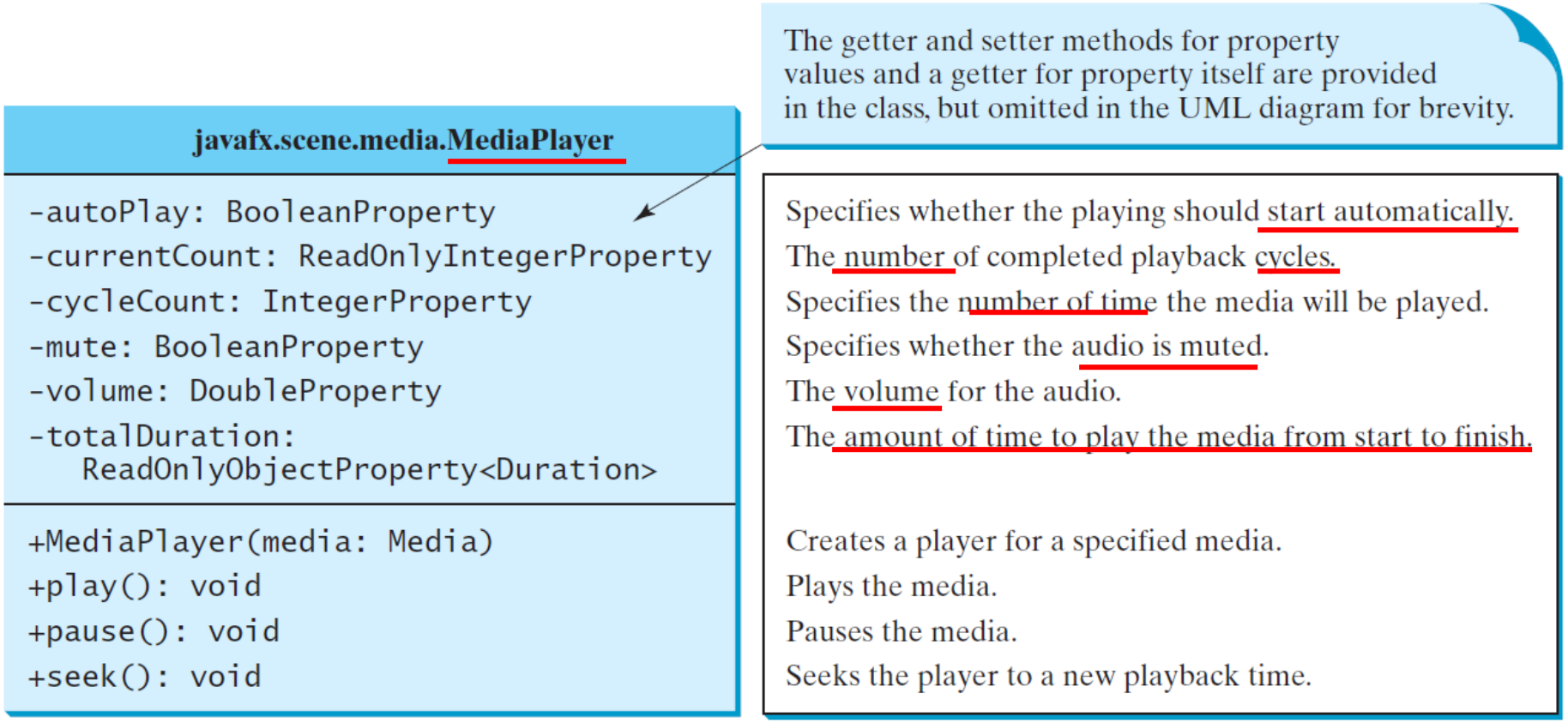
Media

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.



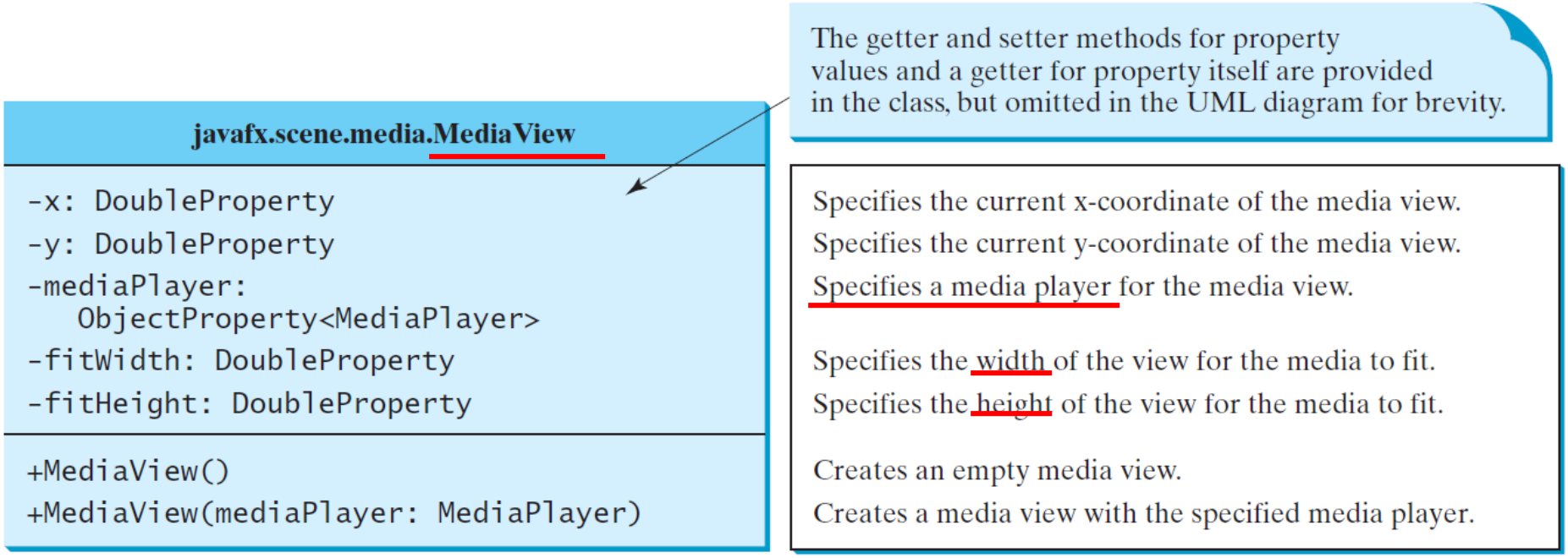
MediaPlayer

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



MediaView

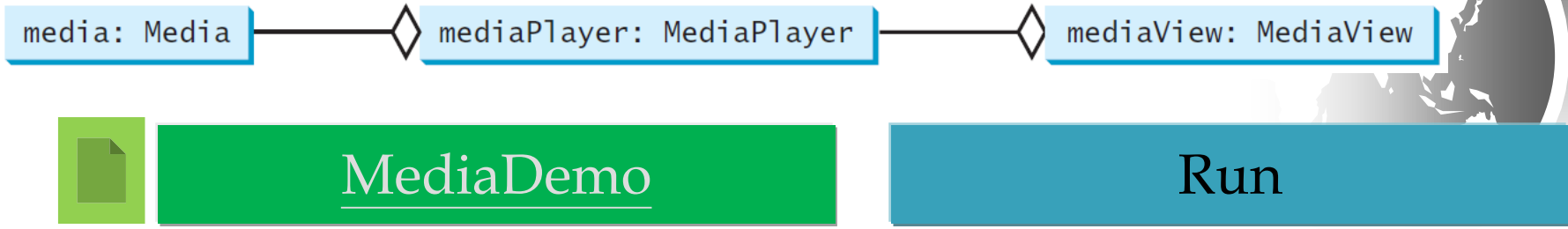
The **MediaView** class is a subclass of **Node** that provides a view of the **Media** being played by a **MediaPlayer**. The **MediaView** class provides the properties for viewing the media.



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.


```
private static final String MEDIA_URL =
"https://liveexample.pearsoncmg.com/common/sample.mp4";

Media media = new Media(MEDIA_URL);
MediaPlayer mediaPlayer = new MediaPlayer(media);
MediaView mediaView = new MediaView(mediaPlayer);
```

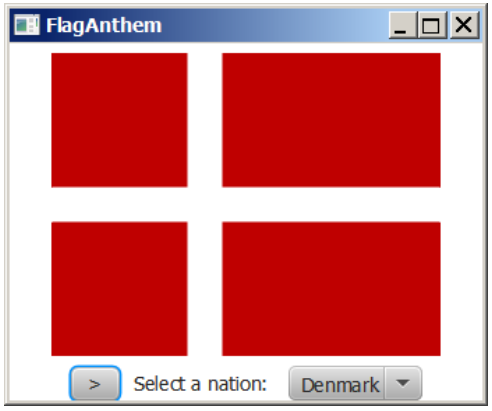


```
private static final String MEDIA_URL =
   "https://liveexample.pearsoncmg.com/common/sample.mp4";

 @Override // Override the start method in the Application class
 public void start(Stage primaryStage) {
   Media media = new Media(MEDIA_URL);
   MediaPlayer mediaPlayer = new MediaPlayer(media);
   MediaView mediaView = new MediaView(mediaPlayer);

   Button playButton = new Button(">");
   playButton.setOnAction(e -> {
     if (playButton.getText().equals(">")) {
       mediaPlayer.play();
       playButton.setText("||");
     } else {
       mediaPlayer.pause();
       playButton.setText(">");
     }
   });
```

```
mediaPlayer.volumeProperty().bind( slVolume.valueProperty().divide(100));
```

Case Study: National Flags and Anthems

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



FlagAnthem

Run

```
// Load images and audio
  for (int i = 0; i < NUMBER_OF_NATIONS; i++) {
    images[i] = new Image(URLBase + "/image/flag" + i + ".gif");
    mp[i] = new MediaPlayer(new Media(
      URLBase + "/audio/anthem/anthem" + i + ".mp3"));
  }

Button btPlayPause = new Button("||");
  btPlayPause.setOnAction(e -> {
    if (btPlayPause.getText().equals(">")) {
      btPlayPause.setText("||");
      mp[currentIndex].play();
    }
    else {
      btPlayPause.setText(">");
      mp[currentIndex].pause();
    } });
```