CHAPTER

16

JavaFX UI Controls and Multimedia

Objectives

- To create graphical user interfaces with various user-interface controls (§§16.2–16.11).
- To create a label with text and graphics 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 playing the national anthem (§16.14).

16.1 Introduction

JavaFX provides many UI controls for developing a comprehensive user interface.

GUI

A graphical user interface (GUI) makes a program user-friendly and easy to use. Creating a GUI requires creativity and knowledge of how UI controls work. Since the UI controls in JavaFX are very flexible and versatile, you can create a wide assortment of useful user interfaces for rich GUI applications.

Oracle provides tools for visually designing and developing GUIs. This enables the programmer to rapidly assemble the elements of a GUI with minimum coding. Tools, however, cannot do everything. You have to modify the programs they produce. Consequently, before you begin to use the visual tools, you must understand the basic concepts of JavaFX GUI programming.

Previous chapters used UI controls such as **`Button`**, **`Label`**, and **`TextField`**. This chapter introduces the frequently used UI controls in detail (see Figure 16.1).

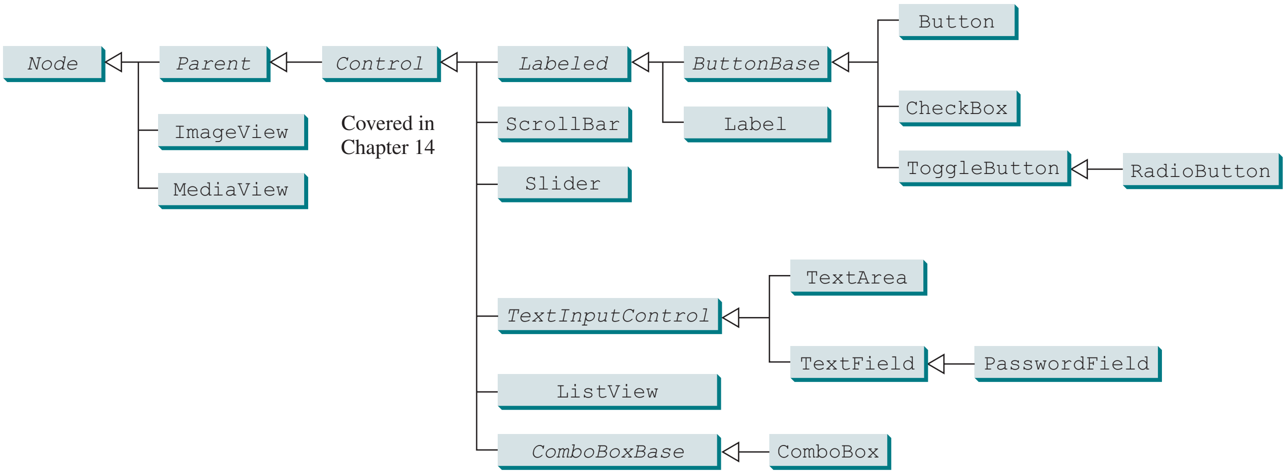


FIGURE 16.1 These UI controls are frequently used to create user interfaces.

naming convention for controls

Note

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`**, respectively.

16.2 Labeled and Label

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).*

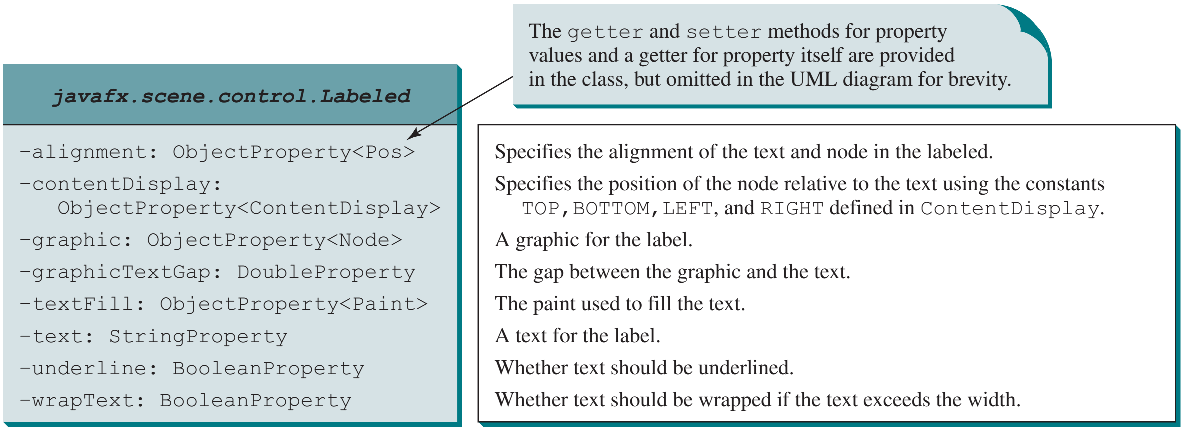
Labels and buttons share many common properties. These common properties are defined in the **`Labeled`** class, as shown in Figure 16.2.

A **`Label`** can be constructed using one of the three constructors shown in Figure 16.3.

The **`graphic`** property can be any node such as a shape, an image, or a control. Listing 16.1 gives an example that displays several labels with text and images in the label, as shown in Figure 16.4.

LISTING 16.1 LabelWithGraphic.java

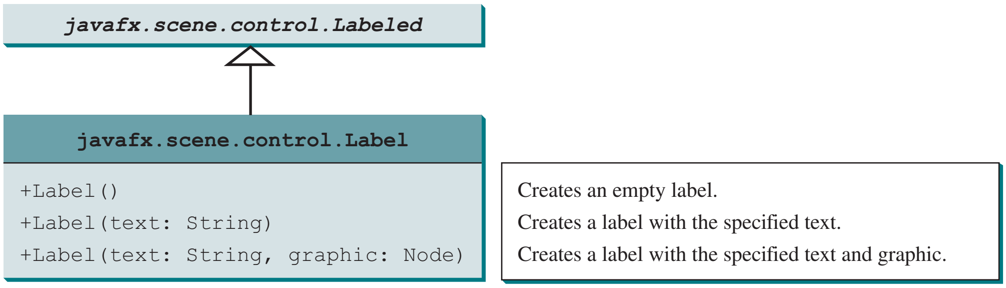
```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.ContentDisplay;
```


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 label.
The gap between the graphic and the text.
The paint used to fill the text.
A text for the label.
Whether text should be underlined.
Whether text should be wrapped if the text exceeds the width.

Figure 16.2 Labeled defines common properties for Label, Button, CheckBox, and RadioButton.


javafx.scene.control.Labeled
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.

Figure 16.3 Label is created to display a text or a node, or both.

```java
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.scene.shape.Ellipse;

public class LabelWithGraphic extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    ImageView us = new ImageView(new Image("image/us.gif"));
    Label lb1 = new Label("US\n50 States", us);
    lb1.setStyle("-fx-border-color: green; -fx-border-width: 2");
    lb1.setContentDisplay(ContentDisplay.BOTTOM);
    lb1.setTextFill(Color.RED);

    Label lb2 = new Label("Circle", new Circle(50, 50, 25));
    lb2.setContentDisplay(ContentDisplay.TOP);
    lb2.setTextFill(Color.ORANGE);

    Label lb3 = new Label("Rectangle", new Rectangle(10, 10, 50, 25));
    lb3.setContentDisplay(ContentDisplay.RIGHT);

    Label lb4 = new Label("Ellipse", new Ellipse(50, 50, 50, 25));
    lb4.setContentDisplay(ContentDisplay.LEFT);

```

create a label
set node position
create a label
set node position
create a label
create a label

create a label

add labels to pane

```
34       Ellipse ellipse = new Ellipse(50, 50, 50, 25);
35       ellipse.setStroke(Color.GREEN);
36       ellipse.setFill(Color.WHITE);
37       StackPane stackPane = new StackPane();
38       stackPane.getChildren().addAll(ellipse, new Label("JavaFX"));
39       Label lb5 = new Label("A pane inside a label", stackPane);
40       lb5.setContentDisplay(ContentDisplay.BOTTOM);
41
42       HBox pane = new HBox(20);
43       pane.getChildren().addAll(lb1, lb2, lb3, lb4, lb5);
44
45       // Create a scene and place it in the stage
46       Scene scene = new Scene(pane, 450, 150);
47       primaryStage.setTitle("LabelWithGraphic"); // Set the stage title
48       primaryStage.setScene(scene); // Place the scene in the stage
49       primaryStage.show(); // Display the stage
50     }
51   }
```

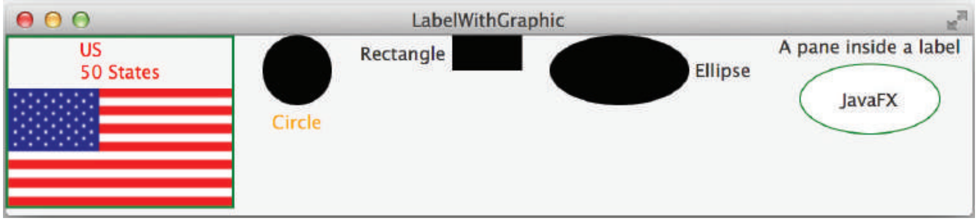


FIGURE 16.4 The program displays labels with texts and nodes. *Source*: booka/Fotolia.

The program creates a label with a text and an image (line 19). The text is `US\n50 States`, so it is displayed in two lines. Line 21 specifies that the image is placed at the bottom of the text.

The program creates a label with a text and a circle (line 24). The circle is placed on top of the text (line 25). The program creates a label with a text and a rectangle (line 28). The rectangle is placed on the right of the text (line 29). The program creates a label with a text and an ellipse (line 31). The ellipse is placed on the left of the text (line 32).

The program creates an ellipse (line 34), places it along with a label to a stack pane (line 38), and creates a label with a text and the stack pane as the node (line 39). As seen from this example, you can place any node in a label.

The program creates an `HBox` (line 42) and places all five labels into the `HBox` (line 43).

Check Point

16.2.1 How do you create a label with a node without a text?

16.2.2 How do you place a text on the right of the node in a label?

16.2.3 Can you display multiple lines of text in a label?

16.2.4 Can the text in a label be underlined?

16.3 Button

Key Point

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 as shown in Figure 16.5.

The `Labeled` class defines the common properties for labels and buttons. A button is just like a label, except that the button has the `onAction` property defined in the `ButtonBase` class, which sets a handler for handling a button's action.

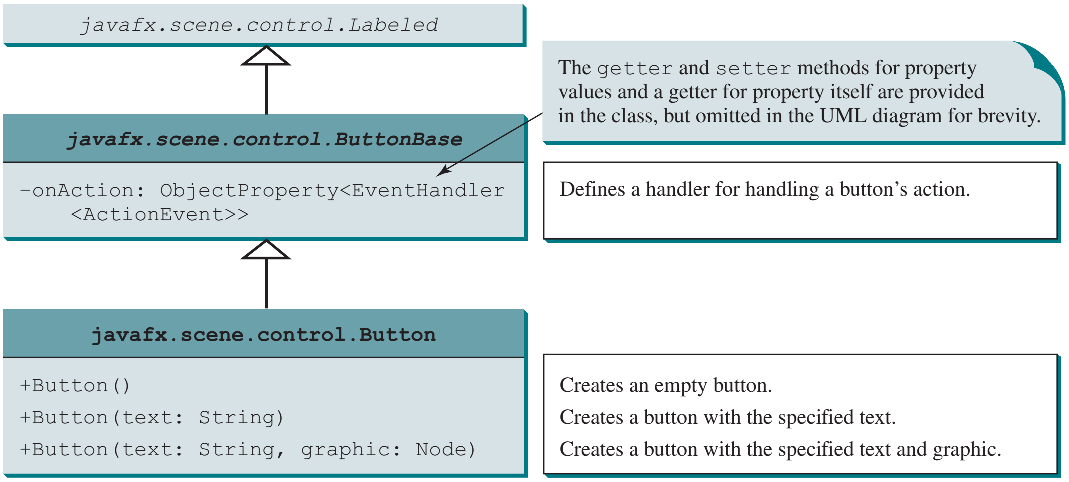

javafx.scene.control.Labeled
javafx.scene.control.ButtonBase
-onAction: ObjectProperty<EventHandler <ActionEvent>>
javafx.scene.control.Button
+Button()
+Button(text: String)
+Button(text: String, graphic: Node)
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.
Defines a handler for handling a button's action.
Creates an empty button.
Creates a button with the specified text.
Creates a button with the specified text and graphic.

FIGURE 16.5 ButtonBase extends Labeled and defines common features for all buttons.

Listing 16.2 gives a program that uses the buttons to control the movement of a text, as shown in Figure 16.6.

LISTING 16.2 ButtonDemo.java

```java
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.ImageView;
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 {
  protected Text text = new Text(50, 50, "JavaFX Programming");

  protected BorderPane getPane() {
    HBox paneForButtons = new HBox(20);
    Button btLeft = new Button("Left",
      new ImageView("image/left.gif"));
    Button btRight = new Button("Right",
      new ImageView("image/right.gif"));
    paneForButtons.getChildren().addAll(btLeft, btRight);
    paneForButtons.setAlignment(Pos.CENTER);
    paneForButtons.setStyle("-fx-border-color: green");

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

    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));

    return pane;
```

create a button

add buttons to pane

create a border pane

add buttons to the bottom

add an action handler

return a pane

set pane to scene

```
36    }
37
38    @Override // Override the start method in the Application class
39    public void start(Stage primaryStage) {
40      // Create a scene and place it in the stage
41      Scene scene = new Scene(getPane(), 450, 200);
42      primaryStage.setTitle("ButtonDemo"); // Set the stage title
43      primaryStage.setScene(scene); // Place the scene in the stage
44      primaryStage.show(); // Display the stage
45    }
46  }
```

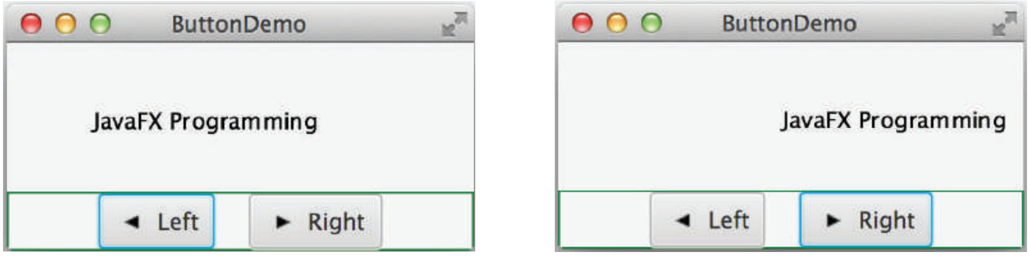


Figure 16.6 The program demonstrates using buttons. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

The program creates two buttons, `btLeft` and `btRight`, with each button containing a text and an image (lines 17–20). The buttons are placed in an `HBox` (line 21) and the `HBox` is placed in the bottom of a border pane (line 26). A text is created in line 13 and is placed in the center of the border pane (line 30). The action handler for `btLeft` moves the text to the left (line 32). The action handler for `btRight` moves the text to the right (line 33).

`getPane()` protected

The program purposely defines a protected `getPane()` method to return a pane (line 15). This method will be overridden by subclasses in the upcoming examples to add more nodes in the pane. The text is declared protected so it can be accessed by subclasses (line 13).

Check Point

16.3.1 How do you create a button with a text and a node? Can you apply all the methods for `Labeled` to `Button`?

16.3.2 Why is the `getPane()` method protected in Listing 16.2? Why is the data field `text` protected?

16.3.3 How do you set a handler for processing a button-clicked action?

16.4 CheckBox

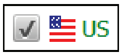
Key Point

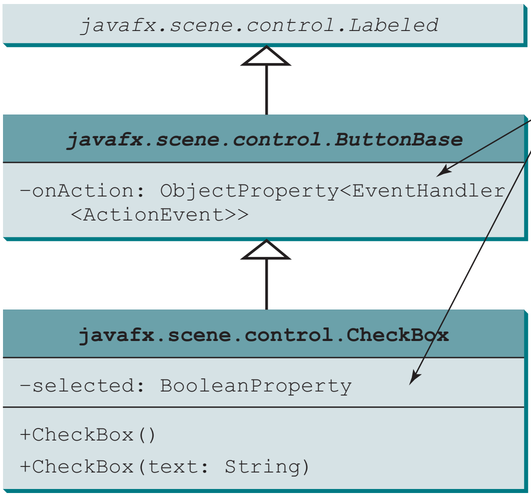
A `CheckBox` is used for the user to make a selection.

Like `Button`, `CheckBox` inherits all the properties such as `onAction`, `text`, `graphic`, `alignment`, `graphicTextGap`, `textFill`, and `contentDisplay` from `ButtonBase` and `Labeled`, as shown in Figure 16.7. In addition, it provides the `selected` property to indicate whether a check box is selected.

Here is an example of a check box with text `US`, a graphic image, green text color, black border, and initially selected.

```
CheckBox chkUS = new CheckBox("US");
chkUS.setGraphic(new ImageView("image/usIcon.gif"));
chkUS.setTextFill(Color.GREEN);
chkUS.setContentDisplay(ContentDisplay.LEFT);
chkUS.setStyle("-fx-border-color: black");
chkUS.setSelected(true);
chkUS.setPadding(new Insets(5, 5, 5, 5));
```




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

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.

Indicates whether this check box is checked.

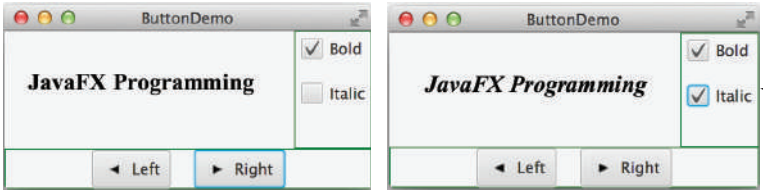
Creates an empty check box.

Creates a check box with the specified text.

Figure 16.7 CheckBox contains the properties inherited from ButtonBase and Labeled.

When a check box is clicked (checked or unchecked), it fires an ActionEvent. To see if a check box is selected, use the isSelected() method.

We now write a program that adds two check boxes named Bold and Italic to the preceding example to let the user specify whether the message is in bold or italic, as shown in Figure 16.8.



VBox containing two check boxes

Figure 16.8 The program demonstrates check boxes. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

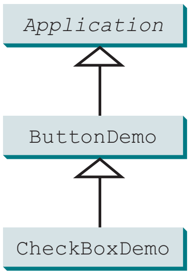
There are at least two approaches to writing this program. The first is to revise the preceding ButtonDemo class to insert the code for adding the check boxes and processing their events. The second is to define a subclass that extends ButtonDemo. Please implement the first approach as an exercise. Listing 16.3 gives the code to implement the second approach.

Listing 16.3 CheckBoxDemo.java

```java
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
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;

public class CheckBoxDemo extends ButtonDemo {
  @Override // Override the getPane() method in the super class
  protected BorderPane getPane() {
    BorderPane pane = super.getPane();

    Font fontBoldItalic = Font.font("Times New Roman",
      FontWeight.BOLD, FontPosture.ITALIC, 20);
```



override getPane()

invoke super.getPane()

create fonts

pane for check boxes

create check boxes

create a handler

set handler for action

return a pane

```
18        Font fontBold = Font.font("Times New Roman",
19          FontWeight.BOLD, FontPosture.REGULAR, 20);
20        Font fontItalic = Font.font("Times New Roman",
21          FontWeight.NORMAL, FontPosture.ITALIC, 20);
22        Font fontNormal = Font.font("Times New Roman",
23          FontWeight.NORMAL, FontPosture.REGULAR, 20);
24
25        text.setFont(fontNormal);
26
27        VBox paneForCheckBoxes = new VBox(20);
28        paneForCheckBoxes.setPadding(new Insets(5, 5, 5, 5));
29        paneForCheckBoxes.setStyle("-fx-border-color: green");
30        CheckBox chkBold = new CheckBox("Bold");
31        CheckBox chkItalic = new CheckBox("Italic");
32        paneForCheckBoxes.getChildren().addAll(chkBold, chkItalic);
33        pane.setRight(paneForCheckBoxes);
34
35        EventHandler<ActionEvent> handler = e -> {
36          if (chkBold.isSelected() && chkItalic.isSelected()) {
37            text.setFont(fontBoldItalic); // Both check boxes checked
38          }
39          else if (chkBold.isSelected()) {
40            text.setFont(fontBold); // The Bold check box checked
41          }
42          else if (chkItalic.isSelected()) {
43            text.setFont(fontItalic); // The Italic check box checked
44          }
45          else {
46            text.setFont(fontNormal); // Both check boxes unchecked
47          }
48        };
49
50        chkBold.setOnAction(handler);
51        chkItalic.setOnAction(handler);
52
53        return pane; // Return a new pane
54      }
55
56      public static void main(String[] args) {
57        launch(args);
58      }
59    }
```

CheckBoxDemo extends ButtonDemo and overrides the getPane() method (line 13). The new getPane() method invokes the super.getPane() method from the ButtonDemo class to obtain a border pane that contains the buttons and a text (line 14). The check boxes are created and added to paneForCheckBoxes (lines 30–32). paneForCheckBoxes is added to the border pane (lines 33).

The handler for processing the action event on check boxes is created in lines 35–48. It sets the appropriate font based on the status of the check boxes.

The start method for this JavaFX program is defined in ButtonDemo and inherited in CheckBoxDemo. Therefore, when you run CheckBoxDemo, the start method in ButtonDemo is invoked. Since the getPane() method is overridden in CheckBoxDemo, the method in CheckBoxDemo is invoked from line 41 in Listing 16.2, ButtonDemo.java. For additional information, see CheckPoint question 16.4.1.

Check Point

16.4.1 What is the output of the following code?

```
public class Test {
  public static void main(String[] args) {
    Test test = new Test();
    test.new B().start();
  }

  class A {
    public void start() {
      System.out.println(getP());
    }

    public int getP() {
      return 1;
    }
  }

  class B extends A {
    public int getP() {
      return 2 + super.getP();
    }
  }
}
```

16.4.2 How do you test if a check box is selected?

16.4.3 Can you apply all the methods for `Labeled` to `CheckBox`?

16.4.4 Can you set a node for the `graphic` property in a check box?

16.5 RadioButton

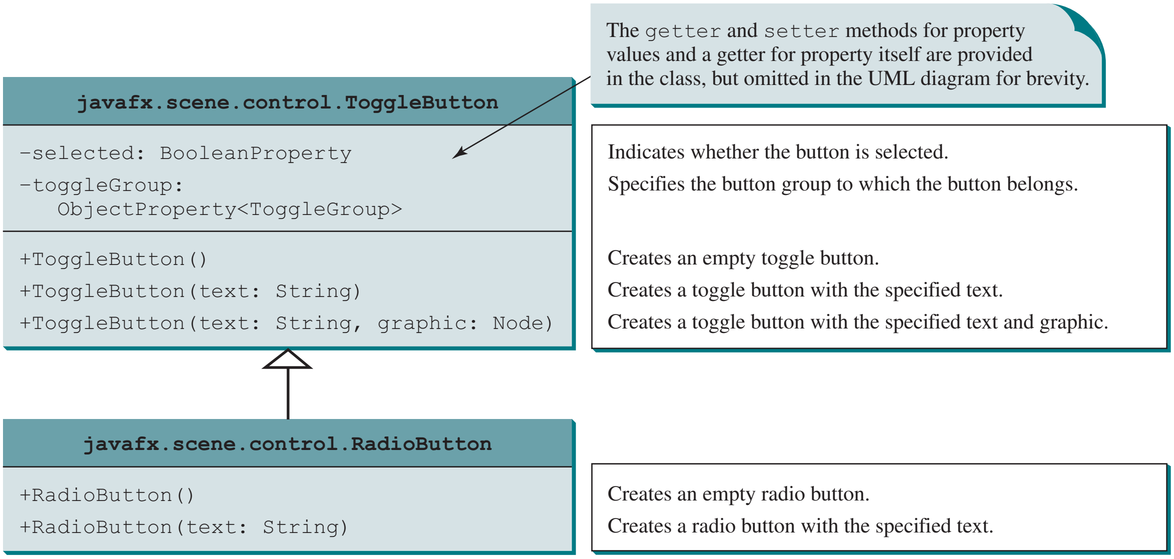
Radio buttons, also known as option buttons, *enable you to choose a single item from a group of choices.*

Key Point

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).

option buttons

`RadioButton` is a subclass of `ToggleButton`. The difference between a radio button and a toggle button is that a radio button displays a circle, but a toggle button is rendered similar to a button. The UML diagrams for `ToggleButton` and `RadioButton` are shown in Figure 16.9.



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

Indicates whether the button is selected.

-toggleGroup: ObjectProperty<ToggleGroup>

Specifies the button group to which the button belongs.

+ToggleButton()

Creates an empty toggle button.

+ToggleButton(text: String)

Creates a toggle button with the specified text.

+ToggleButton(text: String, graphic: Node)

Creates a toggle button with the specified text and graphic.

javafx.scene.control.RadioButton

+RadioButton()

Creates an empty radio button.

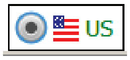
+RadioButton(text: String)

Creates a radio button with the specified text.

FIGURE 16.9 `ToggleButton` and `RadioButton` are specialized buttons for making selections.

Here is an example of a radio button with text US, a graphic image, green text color, black border, and initially selected.

```
Radiobutton rbUS = new RadioButton("US");
rbUS.setGraphic(new ImageView("image/usIcon.gif"));
rbUS.setTextFill(Color.GREEN);
rbUS.setContentDisplay(ContentDisplay.LEFT);
rbUS.setStyle("-fx-border-color: black");
rbUS.setSelected(true);
rbUS.setPadding(new Insets(5, 5, 5, 5));
```



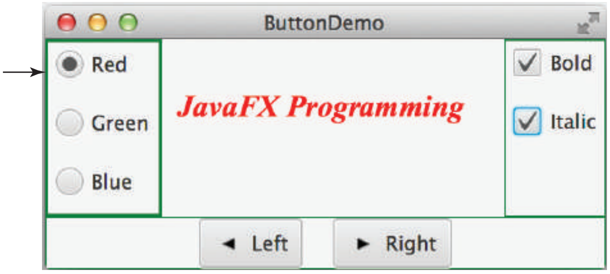
To group radio buttons, you need to create an instance of ToggleGroup and set a radio button's toggleGroup property to join the group, as follows:

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

This code creates a button group for radio buttons rbRed, rbGreen, and rbBlue so buttons rbRed, rbGreen, and rbBlue are selected mutually exclusively. Without grouping, these buttons would be independent.

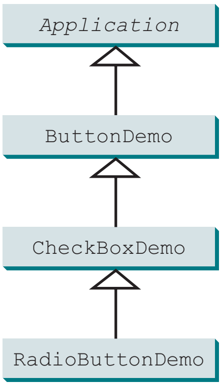
When a radio button is changed (selected or deselected), it fires an ActionEvent. To see if a radio button is selected, use the isSelected() method.

We now give a program that adds three radio buttons named Red, Green, and Blue to the preceding example to let the user choose the color of the message, as shown in Figure 16.10.



VBox containing three radio buttons

FIGURE 16.10 The program demonstrates using radio buttons. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.



Again, there are at least two approaches to writing this program. The first is to revise the preceding CheckBoxDemo class to insert the code for adding the radio buttons and processing their events. The second is to define a subclass that extends CheckBoxDemo. Listing 16.4 gives the code to implement the second approach.

LISTING 16.4 RadioButtonDemo.java

```
import javafx.geometry.Insets;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;

public class RadioButtonDemo extends CheckBoxDemo {
  @Override // Override the getPane() method in the super class
  protected BorderPane getPane() {
    BorderPane pane = super.getPane();

```

override getPane()
invoke super.getPane()

```
13      VBox paneForRadioButtons = new VBox(20);
14      paneForRadioButtons.setPadding(new Insets(5, 5, 5, 5));
15      paneForRadioButtons.setStyle
16          ("-fx-border-width: 2px; -fx-border-color: green");
17
18      RadioButton rbRed = new RadioButton("Red");
19      RadioButton rbGreen = new RadioButton("Green");
20      RadioButton rbBlue = new RadioButton("Blue");
21      paneForRadioButtons.getChildren().addAll(rbRed, rbGreen, rbBlue);
22      pane.setLeft(paneForRadioButtons);
23
24      ToggleGroup group = new ToggleGroup();
25      rbRed.setToggleGroup(group);
26      rbGreen.setToggleGroup(group);
27      rbBlue.setToggleGroup(group);
28
29      rbRed.setOnAction(e -> {
30        if (rbRed.isSelected()) {
31          text.setFill(Color.RED);
32        }
33      });
34
35      rbGreen.setOnAction(e -> {
36        if (rbGreen.isSelected()) {
37          text.setFill(Color.GREEN);
38        }
39      });
40
41      rbBlue.setOnAction(e -> {
42        if (rbBlue.isSelected()) {
43          text.setFill(Color.BLUE);
44        }
45      });
46
47      return pane;
48    }
49
50    public static void main(String[] args) {
51      launch(args);
52    }
53  }
```

pane for radio buttons

create radio buttons

add to border pane

group radio buttons

handle radio button

return border pane

RadioButtonDemo extends CheckBoxDemo and overrides the getPane() method (line 10). The new getPane() method invokes the getPane() method from the CheckBoxDemo class to create a border pane that contains the check boxes, buttons, and a text (line 11). This border pane is returned from invoking super.getPane(). The radio buttons are created and added to paneForRadioButtons (lines 18–21). paneForRadioButtons is added to the border pane (line 22).

The radio buttons are grouped together in lines 24–27. The handlers for processing the action event on radio buttons are created in lines 29–45. It sets the appropriate color based on the status of the radio buttons.

The start method for this JavaFX program is defined in ButtonDemo and inherited in CheckBoxDemo then in RadioButtonDemo. Thus, when you run RadioButtonDemo, the start method in ButtonDemo is invoked. Since the getPane() method is overridden in RadioButtonDemo, the method in RadioButtonDemo is invoked from line 41 in Listing 16.2, ButtonDemo.java.

Check Point

16.5.1 How do you test if a radio button is selected?

16.5.2 Can you apply all the methods for Labeled to RadioButton?

16.5.3 Can you set any node in the graphic property in a radio button?

16.5.4 How do you group radio buttons?`

Key Point

16.6 TextField

A text field can be used to enter or display a string.

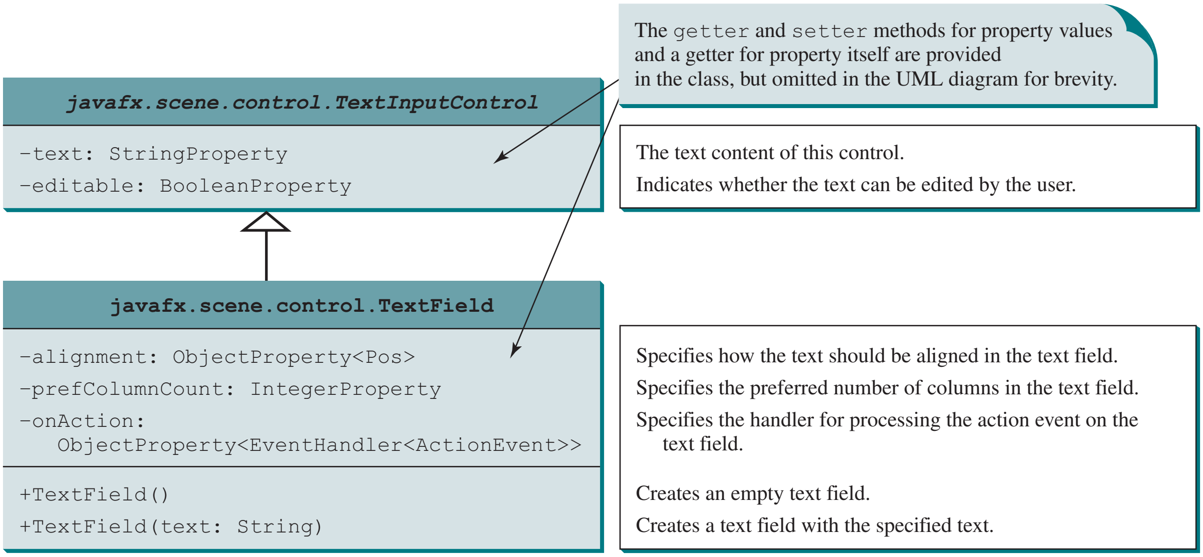
TextField is a subclass of **TextInputControl**. Figure 16.11 lists the properties and constructors in **TextField**.

Here is an example of creating a noneditable text field with red text color, a specified font, and right horizontal alignment:

```
TextField tfMessage = new TextField("T-Storm");
tfMessage.setEditable(false);
tfMessage.setStyle("-fx-text-fill: red");
tfMessage.setFont(Font.font("Times", 20));
tfMessage.setAlignment(Pos.BASELINE_RIGHT);
```

When you move the cursor in the text field and press the *Enter* key, it fires an **ActionEvent**.

Listing 16.5 gives a program that adds a text field to the preceding example to let the user set a new message, as shown in Figure 16.12.



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
-editable: BooleanProperty

The text content of this control.
Indicates whether the text can be edited by the user.

javafx.scene.control.TextField

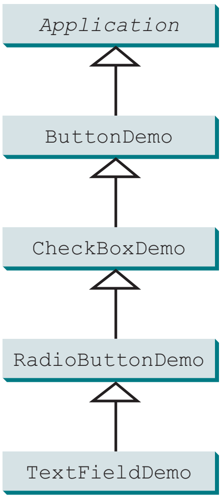
-alignment: ObjectProperty<Pos>
-prefColumnCount: IntegerProperty
-onAction: ObjectProperty<EventHandler<ActionEvent>>

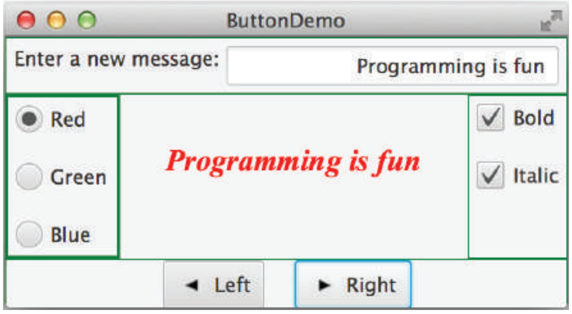
+TextField()
+TextField(text: String)

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.

FIGURE 16.11 **TextField** enables the user to enter or display a string.





Enter a new message: Programming is fun

● Red ○ Green ○ Blue ☑ Bold ☑ Italic

FIGURE 16.12 The program demonstrates using text fields. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

Listing 16.5 TextFieldDemo.java

```java
 1 import javafx.geometry.Insets;
 2 import javafx.geometry.Pos;
 3 import javafx.scene.control.Label;
 4 import javafx.scene.control.TextField;
 5 import javafx.scene.layout.BorderPane;
 6
 7 public class TextFieldDemo extends RadioButtonDemo {
 8   @Override // Override the getPane() method in the super class
 9   protected BorderPane getPane() {
10     BorderPane pane = super.getPane();
11
12     BorderPane paneForTextField = new BorderPane();
13     paneForTextField.setPadding(new Insets(5, 5, 5, 5));
14     paneForTextField.setStyle("-fx-border-color: green");
15     paneForTextField.setLeft(new Label("Enter a new message: "));
16
17     TextField tf = new TextField();
18     tf.setAlignment(Pos.BOTTOM_RIGHT);
19     paneForTextField.setCenter(tf);
20     pane.setTop(paneForTextField);
21
22     tf.setOnAction(e -> text.setText(tf.getText()));
23
24     return pane;
25   }
26
27   public static void main(String[] args) {
28     launch(args);
29   }
30 }
```

override getPane()
invoke super.getPane()
pane for label and text field
create text field
add to border pane
handle text field action
return border pane

TextFieldDemo extends **RadioButtonDemo** (line 7) and adds a label and a text field to let the user enter a new text (lines 12–20). After you set a new text in the text field and press the *Enter* key, a new message is displayed (line 22). Pressing the *Enter* key on the text field triggers an action event.

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 ******.

PasswordField

Check Point

16.6.1 Can you disable editing of a text field?

16.6.2 Can you apply all the methods for **TextInputControl** to **TextField**?

16.6.3 Can you set a node as the **graphic** property in a text field?

16.6.4 How do you align the text in a text field to the right?

16.7 TextArea

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

Key Point

If you want to let the user enter multiple lines of text, you may create several instances of **TextField**. A better alternative, however, is to use **TextArea**, which enables the user to enter multiple lines of text. Figure 16.13 lists the properties and constructors in **TextArea**.

Here is an example of creating a text area with **5** rows and **20** columns, wrapped to the next line, **red** text color, and **Courier** font **20** pixels.

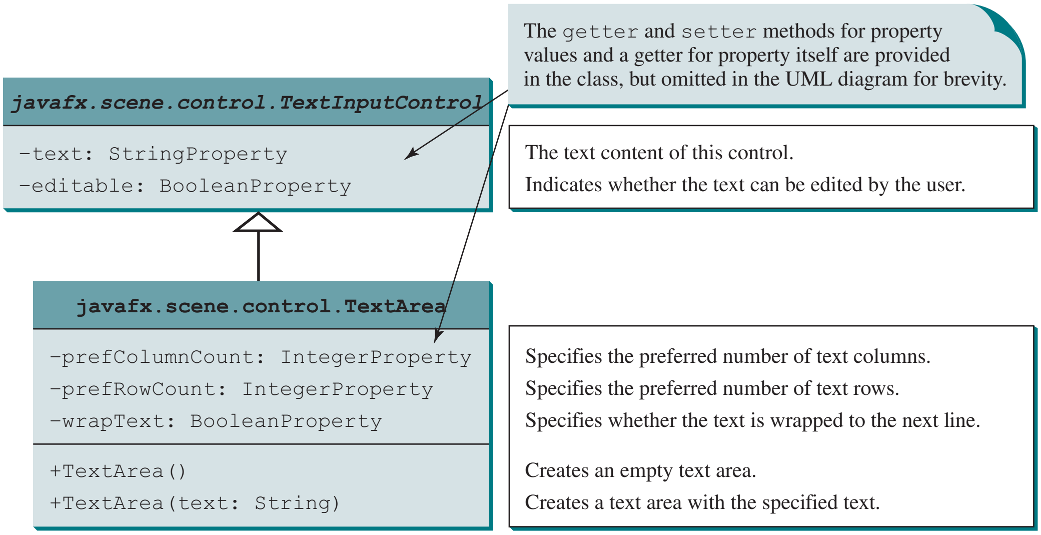

javafx.scene.control.TextInputControl
-text: StringProperty
-editable: BooleanProperty
javafx.scene.control.TextArea
-prefColumnCount: IntegerProperty
-prefRowCount: IntegerProperty
-wrapText: BooleanProperty
+TextArea()
+TextArea(text: String)
The getter and setter methods for property values and a 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.
Specifies the preferred number of text columns.
Specifies the preferred number of text rows.
Specifies whether the text is wrapped to the next line.
Creates an empty text area.
Creates a text area with the specified text.

Figure 16.13 TextArea enables the user to enter or display multiple lines of characters.

```
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("Times", 20));
```

TextArea provides scrolling, but often it is useful to create a ScrollPane object to hold an instance of TextArea and let ScrollPane handle scrolling for TextArea, as follows:

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

ScrollPane

Tip

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

We now give a program that displays an image and a short text in a label, and a long text in a text area, as shown in Figure 16.14.

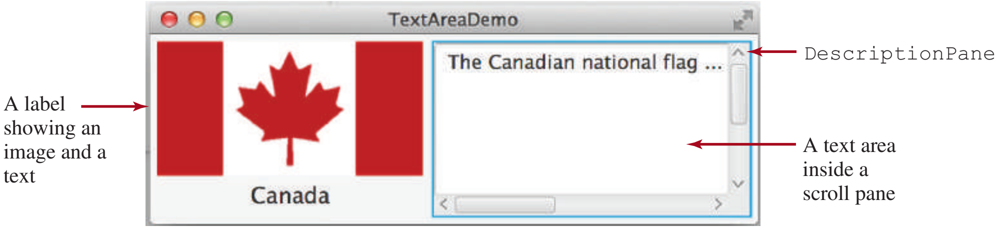

TextAreaDemo
Canada
The Canadian national flag ...
A label showing an image and a text
DescriptionPane
A text area inside a scroll pane

Figure 16.14 The program displays an image in a label, a title in a label, and text in the text area. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

Here are the major steps in the program:

1. Define a class named DescriptionPane that extends BorderPane, as shown in Listing 16.6. This class contains a text area inside a scroll pane and a label for displaying an image icon and a title. The class DescriptionPane will be reused in later examples.

2. Define a class named TextAreaDemo that extends Application, as shown in Listing 16.7. Create an instance of DescriptionPane and add it to the scene. The relationship between DescriptionPane and TextAreaDemo is shown in Figure 16.15.

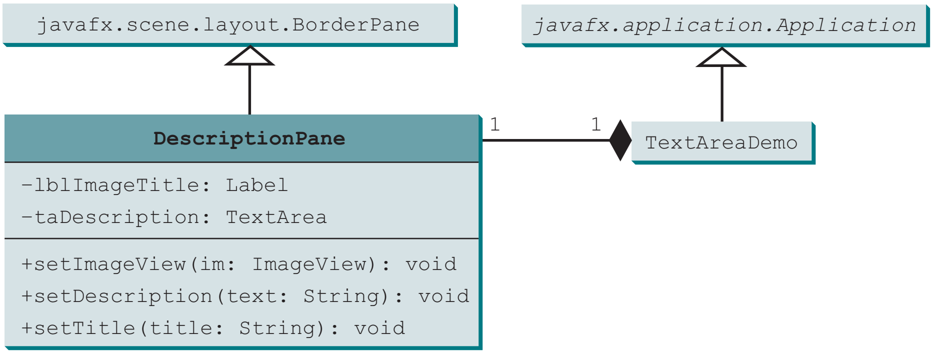

javafx.scene.layout.BorderPane
javafx.application.Application
DescriptionPane
-lblImageTitle: Label
-taDescription: TextArea
+setImageView(im: ImageView): void
+setDescription(text: String): void
+setTitle(title: String): void
1
1
TextAreaDemo

FIGURE 16.15 TextAreaDemo uses DescriptionPane to display an image, title, and text description of a national flag.

LISTING 16.6 DescriptionPane.java

```
import javafx.geometry.Insets;
import javafx.scene.control.Label;
import javafx.scene.control.ContentDisplay;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.scene.text.Font;

public class DescriptionPane extends BorderPane {
  /** Label for displaying an image and a title */
  private Label lblImageTitle = new Label();

  /** Text area for displaying text */
  private TextArea taDescription = new TextArea();

  public DescriptionPane() {
    // Center the icon and text and place the text under the icon
    lblImageTitle.setContentDisplay(ContentDisplay.TOP);
    lblImageTitle.setPrefSize(200, 100);

    // Set the font in the label and the text field
    lblImageTitle.setFont(new Font("SansSerif", 16));
    taDescription.setFont(new Font("Serif", 14));

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

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

    // Place label and scroll pane in the border pane
    setLeft(lblImageTitle);
    setCenter(scrollPane);
    setPadding(new Insets(5, 5, 5, 5));
  }

  /** Set the title */
```

label

text area

label properties

wrap text

read only

scroll pane

```
39    public void setTitle(String title) {
40      lblImageTitle.setText(title);
41    }
42
43    /** Set the image view */
44    public void setImageView(ImageView icon) {
45      lblImageTitle.setGraphic(icon);
46    }
47
48    /** Set the text description */
49    public void setDescription(String text) {
50      taDescription.setText(text);
51    }
52  }
```

The text area is inside a **ScrollPane** (line 30), which provides scrolling functions for the text area.

The **wrapText** property is set to **true** (line 26) so the line is automatically wrapped when the text cannot fit in one line. The text area is set as noneditable (line 27), so you cannot edit the description in the text area.

It is not necessary to define a separate class for **DescriptionPane** in this example. However, this class was defined for reuse in the next section, where you will use it to display a description pane for various images.

Listing 16.7 TextAreaDemo.java

create descriptionPane

set title

set image

add descriptionPane to scene

```
 1  import javafx.application.Application;
 2  import javafx.stage.Stage;
 3  import javafx.scene.Scene;
 4  import javafx.scene.image.ImageView;
 5
 6  public class TextAreaDemo extends Application {
 7    @Override // Override the start method in the Application class
 8    public void start(Stage primaryStage) {
 9      // Declare and create a description pane
10      DescriptionPane descriptionPane = new DescriptionPane();
11
12      // Set title, text, and image in the description pane
13      descriptionPane.setTitle("Canada");
14      String description = "The Canadian national flag ... ";
15      descriptionPane.setImageView(new ImageView("image/ca.gif"));
16      descriptionPane.setDescription(description);
17
18      // Create a scene and place it in the stage
19      Scene scene = new Scene(descriptionPane, 450, 200);
20      primaryStage.setTitle("TextAreaDemo"); // Set the stage title
21      primaryStage.setScene(scene); // Place the scene in the stage
22      primaryStage.show(); // Display the stage
23    }
24  }
```

The program creates an instance of **DescriptionPane** (line 10) and sets the title (line 13), image (line 15), and text (line 16) in the description pane. **DescriptionPane** is a subclass of **Pane**. **DescriptionPane** contains a label for displaying an image, a title, and a text area for displaying a description of the image.

Check Point

16.7.1 How do you create a text area with **10** rows and **20** columns?

16.7.2 How do you obtain the text from a text area?

16.7.3 Can you disable editing of a text area?

16.7.4 What method do you use to wrap text to the next line in a text area?

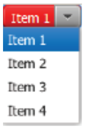
16.8 ComboBox

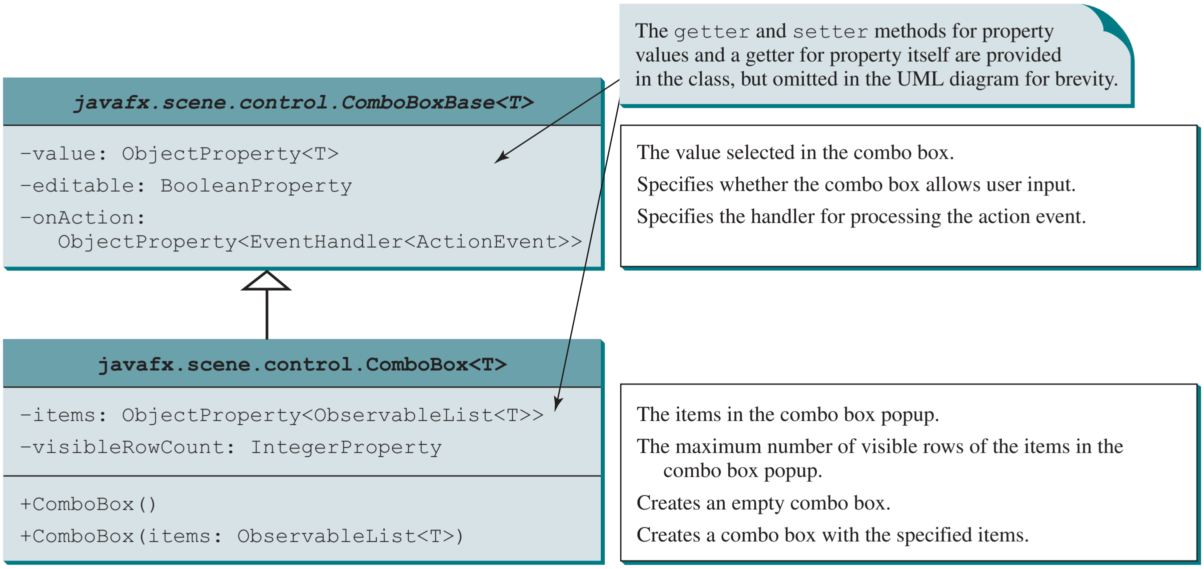
A combo box, also known as a choice list or drop-down list, contains a list of items from which the user can choose.

A combo box is useful for limiting a user's range of choices and avoids the cumbersome validation of data input. Figure 16.16 lists several frequently used properties and constructors in `ComboBox`. `ComboBox` is defined as a generic class like the `ArrayList` class. The generic type `T` specifies the element type for the elements stored in a combo box.

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: #EB0D1B");
cbo.setValue("item 1");
```





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.

FIGURE 16.16 `ComboBox` enables the user to select an item from a list of items.

`ComboBox` inherits from `ComboBoxBase`. `ComboBox` can fire an `ActionEvent`. Whenever an item is selected, an `ActionEvent` is fired. `ObservableList` is a subinterface of `java.util.List`. Therefore, you can apply all the methods defined in `List` for an `ObservableList`. For convenience, JavaFX provides the static method `FXCollections.observableArrayList(arrayOfElements)` for creating an `ObservableList` from an array of elements.

Listing 16.8 gives a program that lets the user view an image and a description of a country's flag by selecting the country from a combo box, as shown in Figure 16.17.

Here are the major steps in the program:

1. Create the user interface.
   Create a combo box with country names as its selection values. Create a `DescriptionPane` object (the `DescriptionPane` class was introduced in the preceding section). Place the combo box at the top of the border pane, and the description pane in the center of the border pane.

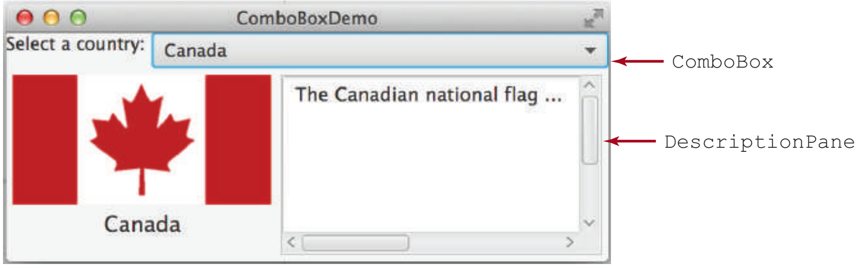


FIGURE 16.17 Information about a country, including an image and a description of its flag, is displayed when the country is selected in the combo box. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

2. Process the event.
   Create a handler for handling action event from the combo box to set the flag title, image, and text in the description pane for the selected country name.

LISTING 16.8 ComboBoxDemo.java

```java
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;

public class ComboBoxDemo extends Application {
  // Declare an array of Strings for flag titles
  private String[] flagTitles = {"Canada", "China", "Denmark",
    "France", "Germany", "India", "Norway", "United Kingdom",
    "United States of America"};

  // Declare an ImageView array for the national flags of 9 countries
  private ImageView[] flagImage = {new ImageView("image/ca.gif"),
    new ImageView("image/china.gif"),
    new ImageView("image/denmark.gif"),
    new ImageView("image/fr.gif"),
    new ImageView("image/germany.gif"),
    new ImageView("image/india.gif"),
    new ImageView("image/norway.gif"),
    new ImageView("image/uk.gif"), new ImageView("image/us.gif")};

  // Declare an array of strings for flag descriptions
  private String[] flagDescription = new String[9];

  // Declare and create a description pane
  private DescriptionPane descriptionPane = new DescriptionPane();

  // Create a combo box for selecting countries
  private ComboBox<String> cbo = new ComboBox<>(); // flagTitles;

  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Set text description
```

countries

image views

description

combo box

```
39      flagDescription[0] = "The Canadian national flag ... ";
40      flagDescription[1] = "Description for China ... ";
41      flagDescription[2] = "Description for Denmark ... ";
42      flagDescription[3] = "Description for France ... ";
43      flagDescription[4] = "Description for Germany ... ";
44      flagDescription[5] = "Description for India ... ";
45      flagDescription[6] = "Description for Norway ... ";
46      flagDescription[7] = "Description for UK ... ";
47      flagDescription[8] = "Description for US ... ";
48
49      // Set the first country (Canada) for display
50      setDisplay(0);
51
52      // Add combo box and description pane to the border pane
53      BorderPane pane = new BorderPane();
54
55      BorderPane paneForComboBox = new BorderPane();
56      paneForComboBox.setLeft(new Label("Select a country: "));
57      paneForComboBox.setCenter(cbo);
58      pane.setTop(paneForComboBox);
59      cbo.setPrefWidth(400);
60      cbo.setValue("Canada");
61
62      ObservableList<String> items =
63        FXCollections.observableArrayList(flagTitles);
64      cbo.getItems().addAll(items);
65      pane.setCenter(descriptionPane);
66
67      // Display the selected country
68      cbo.setOnAction(e -> setDisplay(items.indexOf(cbo.getValue())));
69
70      // Create a scene and place it in the stage
71      Scene scene = new Scene(pane, 450, 170);
72      primaryStage.setTitle("ComboBoxDemo"); // Set the stage title
73      primaryStage.setScene(scene); // Place the scene in the stage
74      primaryStage.show(); // Display the stage
75    }
76
77    /** Set display information on the description pane */
78    public void setDisplay(int index) {
79      descriptionPane.setTitle(flagTitles[index]);
80      descriptionPane.setImageView(flagImage[index]);
81      descriptionPane.setDescription(flagDescription[index]);
82    }
83  }
```

set combo box value

observable list

add to combo box

The program stores the flag information in three arrays: **flagTitles**, **flagImage**, and **flagDescription** (lines 13–28). The array **flagTitles** contains the names of nine countries, the array **flagImage** contains image views of each of the nine countries' flags, and the array **flagDescription** contains descriptions of the flags.

The program creates an instance of **DescriptionPane** (line 31), which was presented in Listing 16.6, DescriptionPane.java. The program creates a combo box with values from **flagTitles** (lines 62 and 63). The **getItems()** method returns a list from the combo box (line 64) and the **addAll** method adds multiple items into the list.

When the user selects an item in the combo box, the action event triggers the execution of the handler. The handler finds the selected index (line 68) and invokes the **setDisplay(int index)** method to set its corresponding flag title, flag image, and flag description on the pane (lines 78–82).

16.8.1 How do you create a combo box and add three items to it?

16.8.2 How do you retrieve an item from a combo box? How do you retrieve a selected item from a combo box?

16.8.3 How do you get the number of items in a combo box? How do you retrieve an item at a specified index in a combo box?

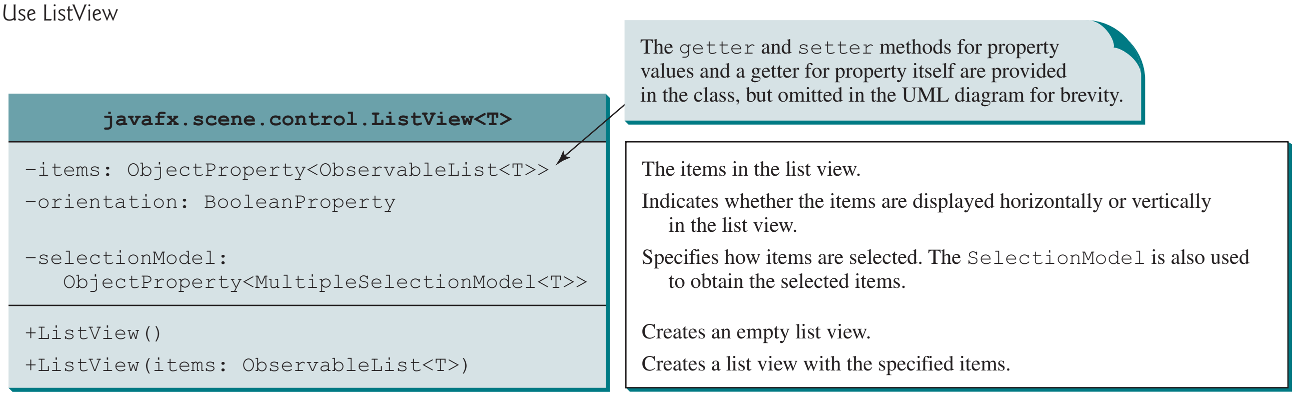
16.8.4 What events would a `ComboBox` fire upon selecting a new item?

16.9 ListView

A list view 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.

VideoNote
Use ListView

Figure 16.18 lists several frequently used properties and constructors in `ListView`. `ListView` is defined as a generic class like the `ArrayList` class. The generic type `T` specifies the element type for the elements stored in a list view.



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.ListView<T>

-items: ObjectProperty<ObservableList<T>>
-orientation: BooleanProperty

-selectionModel:
   ObjectProperty<MultipleSelectionModel<T>>

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

The items in the list view.

Indicates whether the items are displayed horizontally or vertically in the list view.

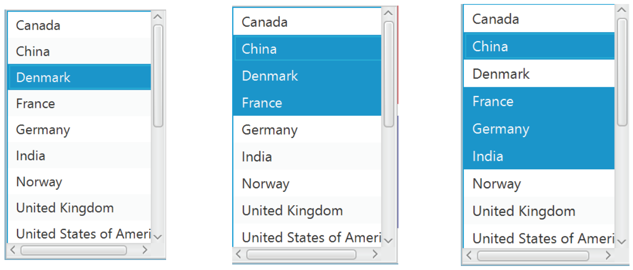
Specifies how items are selected. The `SelectionModel` is also used to obtain the selected items.

Creates an empty list view.

Creates a list view with the specified items.

FIGURE 16.18 `ListView` enables the user to select one or multiple items from a list of items.

The `getSelectionModel()` method returns an instance of `SelectionModel`, which contains the methods for setting a selection mode and obtaining selected indices and items. 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`. Figure 16.19a shows a single selection and Figures 16.19b and c show multiple selections.



(a) Single selection (b) Multiple selection (c) Multiple selection

FIGURE 16.19 `SelectionMode` has two selection modes: single selection and multiple-interval selection. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

The following statements create a list view of six items with multiple selections allowed:

```
ObservableList<String> items =
  FXCollections.observableArrayList("Item 1", "Item 2",
    "Item 3", "Item 4", "Item 5", "Item 6");
ListView<String> lv = new ListView<>(items);
lv.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
```

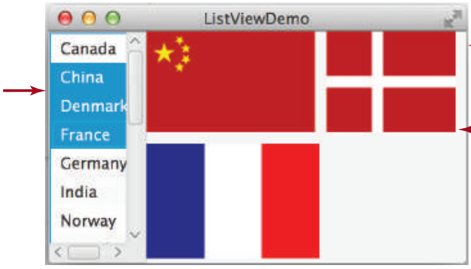
The selection model in a list view has the `selectedItemProperty` property, which is an instance of `Observable`. As discussed in Section 15.10, you can add a listener to this property for handling the property change as follows:

```
lv.getSelectionModel().selectedItemProperty().addListener(
  new InvalidationListener() {
    public void invalidated(Observable ov) {
      System.out.println("Selected indices: "
        + lv.getSelectionModel().getSelectedIndices());
      System.out.println("Selected items: "
        + lv.getSelectionModel().getSelectedItems());
    }
  });
```

This anonymous inner class can be simplified using a lambda expression as follows:

```
lv.getSelectionModel().selectedItemProperty().addListener(ov -> {
  System.out.println("Selected indices: "
    + lv.getSelectionModel().getSelectedIndices());
  System.out.println("Selected items: "
    + lv.getSelectionModel().getSelectedItems());
});
```

Listing 16.9 gives a program that lets users select the countries in a list and displays the flags of the selected countries in the image views. Figure 16.20 shows a sample run of the program.



`ListView` inside a scroll pane

`FlowPane`

An `ImageView` is displayed

Figure 16.20 When the countries in the list are selected, corresponding images of their flags are displayed in the image views. *Source*: booka/Fotolia.

Here are the major steps in the program:

1. Create the user interface.
   Create a list view with nine country names as selection values and place the list view inside a scroll pane. Place the scroll pane on the left of a border pane. Create nine image views to be used to display the countries' flag images. Create a flow pane to hold the image views and place the pane in the center of the border pane.

2. Process the event.
   Create a listener to implement the `invalidated` method in the `InvalidationListener` interface to place the selected countries' flag image views in the pane.

Listing 16.9 ListViewDemo.java

```java
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.collections.FXCollections;
import javafx.scene.Scene;
import javafx.scene.control.ListView;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.SelectionMode;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.FlowPane;

public class ListViewDemo extends Application {
  // Declare an array of Strings for flag titles
  private String[] flagTitles = {"Canada", "China", "Denmark",
    "France", "Germany", "India", "Norway", "United Kingdom",
    "United States of America"};

  // Declare an ImageView array for the national flags of 9 countries
  private ImageView[] ImageViews = {
    new ImageView("image/ca.gif"),
    new ImageView("image/china.gif"),
    new ImageView("image/denmark.gif"),
    new ImageView("image/fr.gif"),
    new ImageView("image/germany.gif"),
    new ImageView("image/india.gif"),
    new ImageView("image/norway.gif"),
    new ImageView("image/uk.gif"),
    new ImageView("image/us.gif")
  };

  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    ListView<String> lv = new ListView<>
      (FXCollections.observableArrayList(flagTitles));
    lv.setPrefSize(400, 400);
    lv.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

    // Create a pane to hold image views
    FlowPane imagePane = new FlowPane(10, 10);
    BorderPane pane = new BorderPane();
    pane.setLeft(new ScrollPane(lv));
    pane.setCenter(imagePane);

    lv.getSelectionModel().selectedItemProperty().addListener(
      ov -> {
        imagePane.getChildren().clear();
        for (Integer i: lv.getSelectionModel().getSelectedIndices()) {
          imagePane.getChildren().add(ImageViews[i]);
        }
    });

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 450, 170);
    primaryStage.setTitle("ListViewDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}
```

create a list view

set list view properties

place list view in pane

listen to item selected

add image views of selected items

The program creates an array of strings for countries (lines 14–16) and an array of nine image views for displaying flag images for nine countries (lines 19–29) in the same order as in the array of countries. The items in the list view are from the array of countries (line 34). Thus, the index 0 of the image view array corresponds to the first country in the list view.

The list view is placed in a scroll pane (line 41) so it can be scrolled when the number of items in the list extends beyond the viewing area.

By default, the selection mode of the list view is single. The selection mode for the list view is set to multiple (line 36), which allows the user to select multiple items in the list view. When the user selects countries in the list view, the listener's handler (lines 44–50) is executed, which gets the indices of the selected items and adds their corresponding image views to the flow pane.

Check Point

16.9.1 How do you create an observable list with an array of strings?

16.9.2 How do you set the orientation in a list view?

16.9.3 What selection modes are available for a list view? What is the default selection mode? How do you set a selection mode?

16.9.4 How do you obtain the selected items and selected indices?

16.10 ScrollBar

Key Point

ScrollBar is a control that enables the user to select from a range of values.

Figure 16.21 shows a scroll bar. Normally, the user changes the value of a scroll bar by making a gesture with the mouse. For example, the user can drag the scroll bar's thumb, click on the scroll bar track, or the scroll bar's left or right buttons.

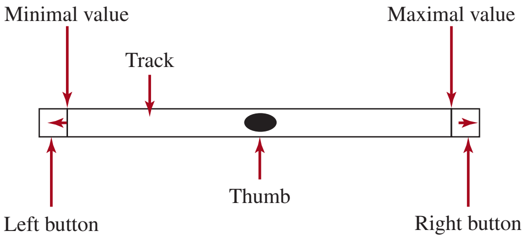
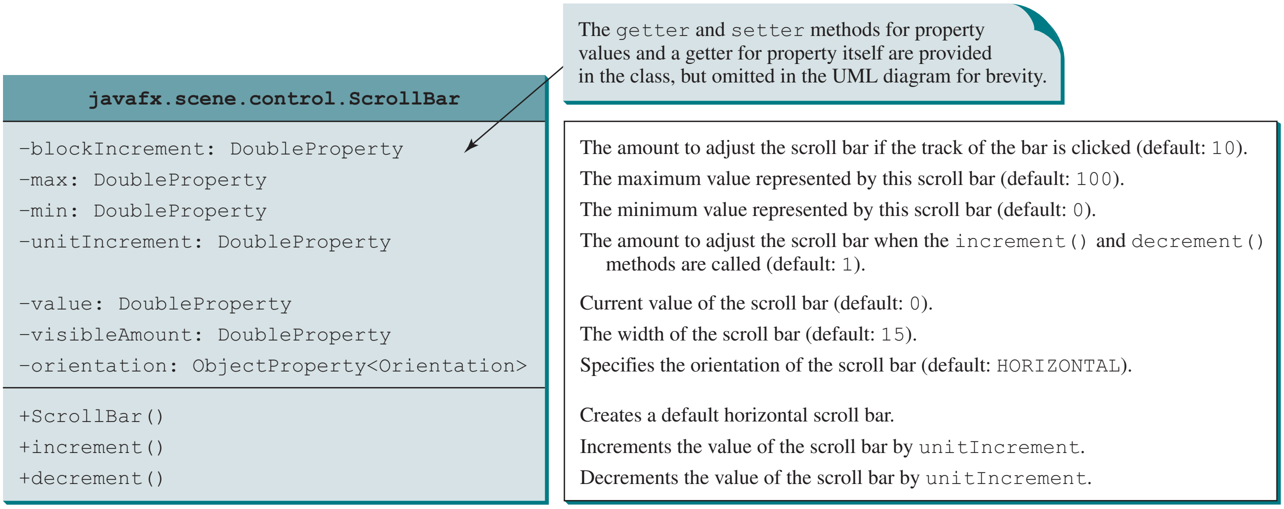


Figure 16.21 A scroll bar graphically represents a range of values.

ScrollBar has the following properties, as shown in Figure 16.22.



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.

Figure 16.22 ScrollBar enables the user to select from a range of values.

Note

The width of the scroll bar's track corresponds to **max + visibleAmount**. When a scroll bar is set to its maximum value, the left side of the bubble is at **max**, and the right side is at **max + visibleAmount**.

When the user changes the value of the scroll bar, it notifies the listener of the change. You can register a listener on the scroll bar's **valueProperty** for responding to this change as follows:

```
ScrollBar sb = new ScrollBar();
sb.valueProperty().addListener(ov -> {
  System.out.println("old value: " + oldVal);
  System.out.println("new value: " + newVal);
});
```

Listing 16.10 gives a program that uses horizontal and vertical scroll bars to move a text displayed on a pane. The horizontal scroll bar is used to move the text to the left and the right, and the vertical scroll bar to move it up and down. A sample run of the program is shown in Figure 16.23.

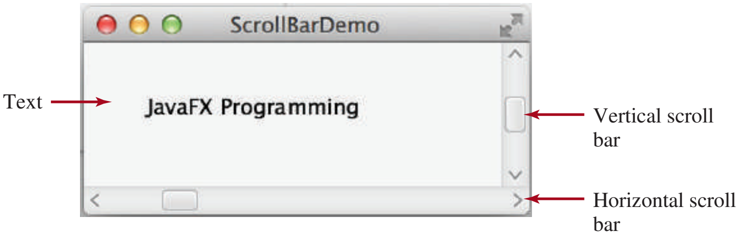


FIGURE 16.23 The scroll bars move the message on a pane horizontally and vertically. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

Here are the major steps in the program:

1. Create the user interface.
   Create a **Text** object and place it in a pane and place the pane in the center of the border pane. Create a vertical scroll bar and place it on the right of the border pane. Create a horizontal scroll bar and place it at the bottom of the border pane.

2. Process the event.
   Create listeners to move the text according to the bar movement in the scroll bars upon the change of the **value** property.

LISTING 16.10 ScrollBarDemo.java

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.geometry.Orientation;
import javafx.scene.Scene;
import javafx.scene.control.ScrollBar;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;
import javafx.scene.text.Text;

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

    ScrollBar sbHorizontal = new ScrollBar();
    ScrollBar sbVertical = new ScrollBar();
    sbVertical.setOrientation(Orientation.VERTICAL);

    // Create a text in a pane
```

horizontal scroll bar
vertical scroll bar

```
20     Pane paneForText = new Pane();
21     paneForText.getChildren().add(text);
22 
23     // Create a border pane to hold text and scroll bars
24     BorderPane pane = new BorderPane();
25     pane.setCenter(paneForText);
26     pane.setBottom(sbHorizontal);
27     pane.setRight(sbVertical);
28 
29     // Listener for horizontal scroll bar value change
30     sbHorizontal.valueProperty().addListener(ov ->
31       text.setX(sbHorizontal.getValue() * paneForText.getWidth() /
32         sbHorizontal.getMax()));
33 
34     // Listener for vertical scroll bar value change
35     sbVertical.valueProperty().addListener(ov ->
36       text.setY(sbVertical.getValue() * paneForText.getHeight() /
37         sbVertical.getMax()));
38 
39     // Create a scene and place it in the stage
40     Scene scene = new Scene(pane, 450, 170);
41     primaryStage.setTitle("ScrollBarDemo"); // Set the stage title
42     primaryStage.setScene(scene); // Place the scene in the stage
43     primaryStage.show(); // Display the stage
44   }
45 }
```

add text to a pane

border pane

set new location for text

set new location for text

The program creates a text (line 13) and two scroll bars (**sbHorizontal** and **sbVertical**) (lines 15 and 16). The text is placed in a pane (line 21) that is then placed in the center of the border pane (line 25). If the text were directly placed in the center of the border pane, the position of the text could not be changed by resetting its *x* and *y* properties. The **sbHorizontal** and **sbVertical** are placed on the right and at the bottom of the border pane (lines 26 and 27), respectively.

You can specify the properties of the scroll bar. By default, the property value is **100** for **max**, **0** for **min**, **10** for **blockIncrement**, and **15** for **visibleAmount**.

A listener is registered to listen for the **sbHorizontal value** property change (lines 30–32). When the value of the scroll bar changes, the listener is notified by invoking the handler to set a new *x* value for the text that corresponds to the current value of **sbHorizontal** (lines 31 and 32).

A listener is registered to listen for the **sbVertical value** property change (lines 35–37). When the value of the scroll bar changes, the listener is notified by invoking the handler to set a new *y* value for the text that corresponds to the current value of **sbVertical** (lines 36 and 37).

Alternatively, the code in lines 30–37 can be replaced by using binding properties as follows:

```
text.xProperty().bind(sbHorizontal.valueProperty().
  multiply(paneForText.widthProperty()).
  divide(sbHorizontal.maxProperty()));

text.yProperty().bind(sbVertical.valueProperty().multiply(
  paneForText.heightProperty().divide(
  sbVertical.maxProperty())));
```

Check Point

16.10.1 How do you create a horizontal scroll bar? How do you create a vertical scroll bar?

16.10.2 How do you write the code to respond to the **value** property change of a scroll bar?

16.10.3 How do you get the value from a scroll bar? How do you get the maximum value from a scroll bar?

16.11 Slider

Key Point

VideoNote
Use Slider

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

Figure 16.24 shows two sliders. `Slider` lets the user graphically select a value by sliding a knob within a bounded interval. The slider can show both major and minor tick marks between them. The number of pixels between the tick marks is specified by the `majorTickUnit` and `minorTickUnit` properties. Sliders can be displayed horizontally or vertically, with or without ticks, and with or without labels.

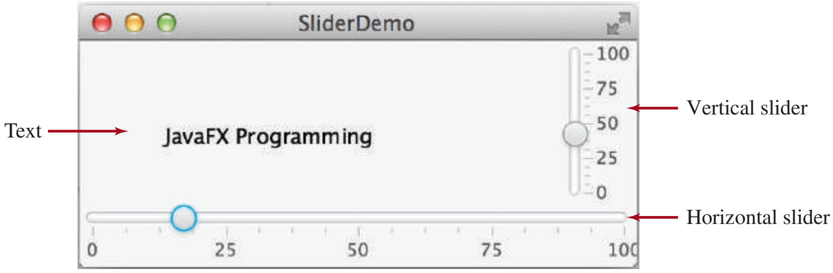
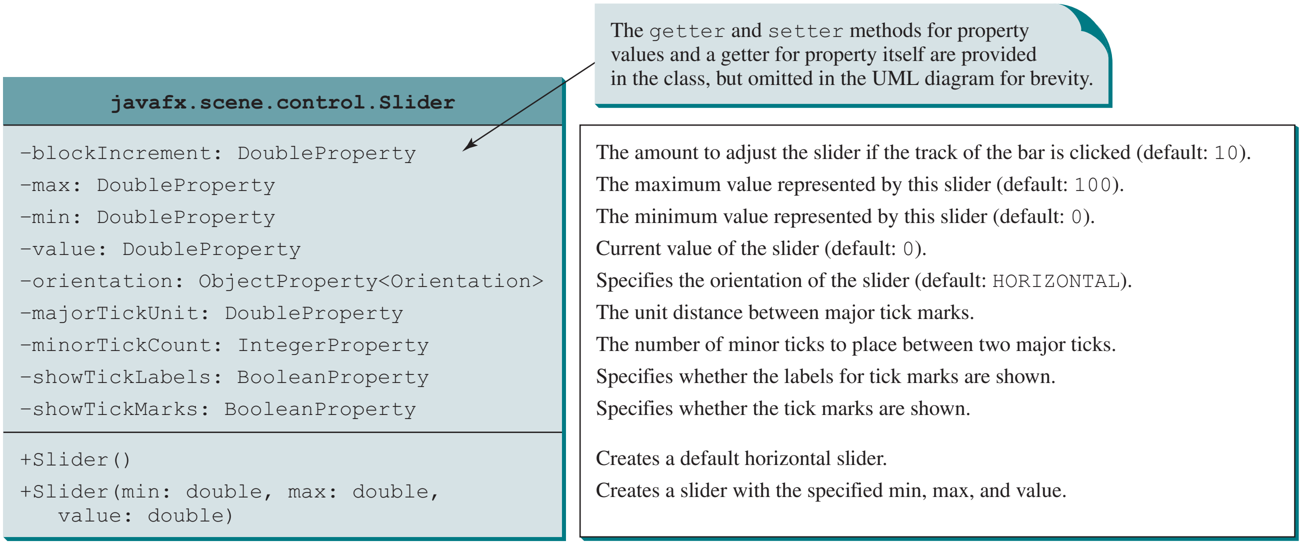


FIGURE 16.24 The sliders move the message on a pane horizontally and vertically. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

The frequently used constructors and properties in `Slider` are shown in Figure 16.25.



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.

FIGURE 16.25 `Slider` enables the user to select from a range of values.

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.

You can add a listener to listen for the `value` property change in a slider in the same way as in a scroll bar. We now rewrite the program in the preceding section using the sliders to move a text displayed on a pane in Listing 16.11. A sample run of the program is shown in Figure 16.24.

LISTING 16.11 SliderDemo.java

```
 1  import javafx.application.Application;
 2  import javafx.stage.Stage;
 3  import javafx.geometry.Orientation;
 4  import javafx.scene.Scene;
 5  import javafx.scene.control.Slider;
 6  import javafx.scene.layout.BorderPane;
 7  import javafx.scene.layout.Pane;
 8  import javafx.scene.text.Text;
 9
10  public class SliderDemo 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      Slider slHorizontal = new Slider();
16      slHorizontal.setShowTickLabels(true);
17      slHorizontal.setShowTickMarks(true);
18
19      Slider slVertical = new Slider();
20      slVertical.setOrientation(Orientation.VERTICAL);
21      slVertical.setShowTickLabels(true);
22      slVertical.setShowTickMarks(true);
23      slVertical.setValue(100);
24
25      // Create a text in a pane
26      Pane paneForText = new Pane();
27      paneForText.getChildren().add(text);
28
29      // Create a border pane to hold text and scroll bars
30      BorderPane pane = new BorderPane();
31      pane.setCenter(paneForText);
32      pane.setBottom(slHorizontal);
33      pane.setRight(slVertical);
34
35      slHorizontal.valueProperty().addListener(ov ->
36        text.setX(slHorizontal.getValue() * paneForText.getWidth() /
37          slHorizontal.getMax()));
38
39      slVertical.valueProperty().addListener(ov ->
40        text.setY((slVertical.getMax() - slVertical.getValue())
41          * paneForText.getHeight() / slVertical.getMax()));
42
43      // Create a scene and place it in the stage
44      Scene scene = new Scene(pane, 450, 170);
45      primaryStage.setTitle("SliderDemo"); // Set the stage title
46      primaryStage.setScene(scene); // Place the scene in the stage
47      primaryStage.show(); // Display the stage
48    }
49  }
```

horizontal slider

set slider properties

vertical slider

set slider properties

add text to a pane

border pane

set new location for text

set new location for text

Slider is similar to ScrollBar but has more features. As shown in this example, you can specify labels, major ticks, and minor ticks on a Slider (lines 16 and 17).

A listener is registered to listen for the slHorizontal value property change (lines 35–37) and another one is for the sbVertical value property change (lines 39–41). When the value of the slider changes, the listener is notified by invoking the handler to set a new position for the text (lines 36 and 37 and 40 and 41). Note since the value of a vertical slider decreases from top to bottom, the corresponding *y* value for the text is adjusted accordingly.

The code in lines 35–41 can be replaced by using binding properties as follows:

```
text.xProperty().bind(slHorizontal.valueProperty().
  multiply(paneForText.widthProperty()).
  divide(slHorizontal.maxProperty()));

text.yProperty().bind((slVertical.maxProperty().subtract(
  slVertical.valueProperty()).multiply(
  paneForText.heightProperty().divide(
  slVertical.maxProperty()))));
```

Listing 15.17 gives a program that displays a bouncing ball. You can add a slider to control the speed of the ball movement, as shown in Figure 16.26. The new program is given in Listing 16.12.

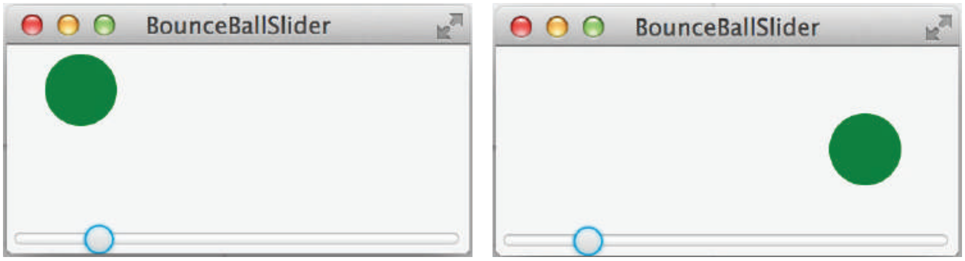


Figure 16.26 You can increase or decrease the speed of the ball using a slider. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

Listing 16.12 BounceBallSlider.java

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Slider;
import javafx.scene.layout.BorderPane;

public class BounceBallSlider extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    BallPane ballPane = new BallPane();
    Slider slSpeed = new Slider();
    slSpeed.setMax(20);
    ballPane.rateProperty().bind(slSpeed.valueProperty());

    BorderPane pane = new BorderPane();
    pane.setCenter(ballPane);
    pane.setBottom(slSpeed);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 250, 250);
    primaryStage.setTitle("BounceBallSlider"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}
```

create a ball pane
create a slider
set max value for slider
bind rate with slider value

create a border pane
add ball pane to center
add slider to the bottom

The `BallPane` class defined in Listing 15.17 animates a ball bouncing in a pane. The `rateProperty()` method in `BallPane` returns a property value for the animation rate. The animation stops if the rate is 0. If the rate is greater than 20, the animation will be too fast. Therefore, we purposely set the rate to a value between 0 and 20. This value is bound to the slider value (line 13). Thus, the slider max value is set to 20 (line 12).

Check Point

16.11.1 How do you create a horizontal slider? How do you create a vertical slider?

16.11.2 How do you add a listener to handle the property value change of a slider?

16.11.3 How do you get the value from a slider? How do you get the maximum value from a slider?

16.12 Case Study: Developing a Tic-Tac-Toe Game

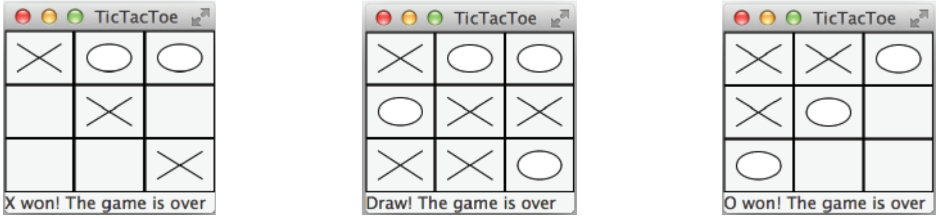
Key Point

This section develops a program for playing tic-tac-toe game.

From the many examples in this and earlier chapters, you have learned about objects, classes, arrays, class inheritance, GUI, and event-driven programming. Now it is time to put what you have learned to work in developing comprehensive projects. In this section, we will develop a JavaFX program with which to play the popular game of tic-tac-toe.

VideoNote
Tic-Tac-Toe

Two players take turns marking an available cell in a 3 × 3 grid with their respective tokens (either X or O). When one player has placed three tokens in a horizontal, vertical, or diagonal row on the grid, the game is over and that player has won. A draw (no winner) occurs when all the cells on the grid have been filled with tokens and neither player has achieved a win. Figure 16.27 shows the representative sample runs of the game.



X won! The game is over

Draw! The game is over

O won! The game is over

(a) The X player won the game

(b) Draw—no winners

(c) The O player won the game

FIGURE 16.27 Two players play a tic-tac-toe game. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

All the examples you have seen so far show simple behaviors that are easy to model with classes. The behavior of the tic-tac-toe game is somewhat more complex. To define classes that model the behavior, you need to study and understand the game.

Assume all the cells are initially empty, and that the first player takes the X token and the second player the O token. To mark a cell, the player points the mouse to the cell and clicks it. If the cell is empty, the token (X or O) is displayed. If the cell is already filled, the player's action is ignored.

From the preceding description, it should be obvious that a cell is a GUI object that handles the mouse-click event and displays tokens. There are many choices for this object. We will use a pane to model a cell and to display a token (X or O). How do you know the state of the cell (empty, X, or O)? You use a property named `token` of the `char` type in the `Cell` class. The `Cell` class is responsible for drawing the token when an empty cell is clicked, so you need to write the code for listening to the mouse-clicked action and for painting the shapes for tokens X and O. The `Cell` class can be defined as shown in Figure 16.28.

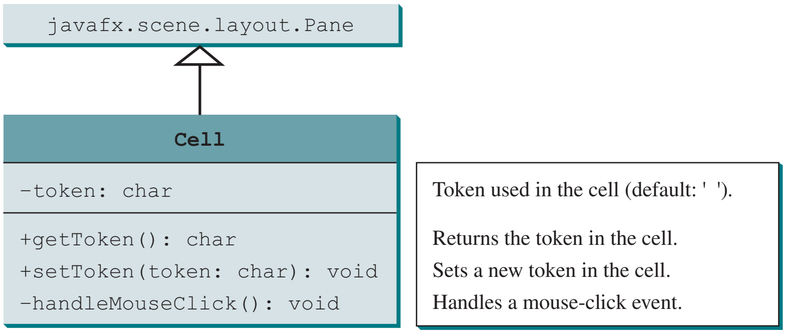

javafx.scene.layout.Pane
Cell
-token: char
+getToken(): char
+setToken(token: char): void
-handleMouseClick(): void
Token used in the cell (default: ' ').
Returns the token in the cell.
Sets a new token in the cell.
Handles a mouse-click event.

FIGURE 16.28 The Cell class displays the token in a cell.

The tic-tac-toe board consists of nine cells, created using new Cell[3][3]. To determine which player's turn it is, you can introduce a variable named whoseTurn of the char type. whoseTurn is initially 'X', then changes to 'O', and subsequently changes between 'X' and 'O' whenever a new cell is occupied. When the game is over, set whoseTurn to ' '.

How do you know whether the game is over, whether there is a winner, and who is the winner, if any? You can define a method named isWon(char token) to check whether a specified token has won and a method named isFull() to check whether all the cells are occupied.

Clearly, two classes emerge from the foregoing analysis. One is the Cell class, which handles operations for a single cell; the other is the TicTacToe class, which plays the whole game and deals with all the cells. The relationship between these two classes is shown in Figure 16.29.

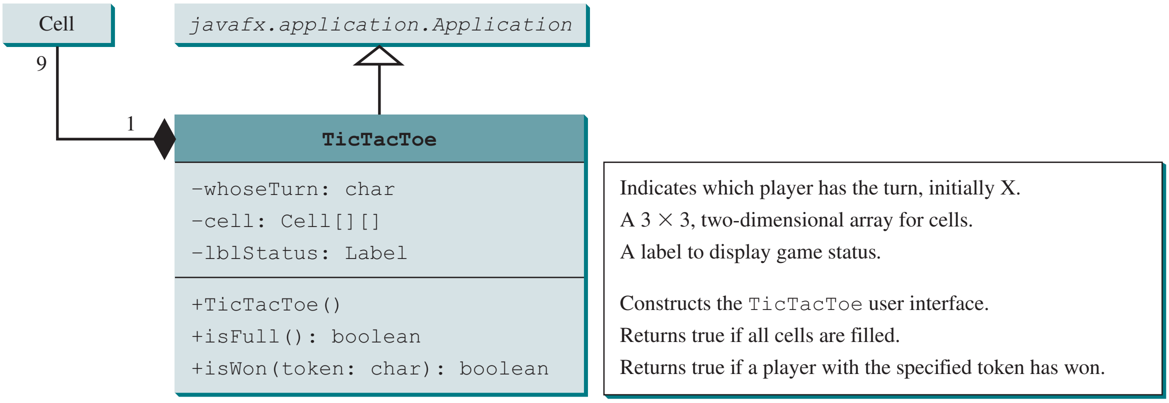

Cell
9
1
javafx.application.Application
TicTacToe
-whoseTurn: char
-cell: Cell[][]
-lblStatus: Label
+TicTacToe()
+isFull(): boolean
+isWon(token: char): boolean
Indicates which player has the turn, initially X.
A 3 × 3, two-dimensional array for cells.
A label to display game status.
Constructs the TicTacToe user interface.
Returns true if all cells are filled.
Returns true if a player with the specified token has won.

FIGURE 16.29 The TicTacToe class contains nine cells.

Since the Cell class is only to support the TicTacToe class, it can be defined as an inner class in TicTacToe. The complete program is given in Listing 16.13.

LISTING 16.13 TicTacToe.java

```java
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.shape.Ellipse;

public class TicTacToe extends Application {
  // Indicate which player has a turn, initially it is the X player
  private char whoseTurn = 'X';

  // Create and initialize cell
  private Cell[][] cell = new Cell[3][3];

  // Create and initialize a status label
  private Label lblStatus = new Label("X's turn to play");

  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Pane to hold cell
    GridPane pane = new GridPane();
    for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++)
        pane.add(cell[i][j] = new Cell(), j, i);

    BorderPane borderPane = new BorderPane();
    borderPane.setCenter(pane);
    borderPane.setBottom(lblStatus);

    // Create a scene and place it in the stage
    Scene scene = new Scene(borderPane, 450, 170);
    primaryStage.setTitle("TicTacToe"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /** Determine if the cell are all occupied */
  public boolean isFull() {
    for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++)
        if (cell[i][j].getToken() == ' ')
          return false;

    return true;
  }

  /** Determine if the player with the specified token wins */
  public boolean isWon(char token) {
    for (int i = 0; i < 3; i++)
      if (cell[i][0].getToken() == token
          && cell[i][1].getToken() == token
          && cell[i][2].getToken() == token) {
        return true;
      }

    for (int j = 0; j < 3; j++)
      if (cell[0][j].getToken() == token
          && cell[1][j].getToken() == token
          && cell[2][j].getToken() == token) {
        return true;
      }
```

main class TicTacToe

hold nine cells

create a cell

tic-tac-toe cells in center

label at bottom

check isFull

check rows

check columns

```java

```

check major diagonal

```java
    if (cell[0][0].getToken() == token
        && cell[1][1].getToken() == token
        && cell[2][2].getToken() == token) {
      return true;
    }

```

check subdiagonal

```java
    if (cell[0][2].getToken() == token
        && cell[1][1].getToken() == token
        && cell[2][0].getToken() == token) {
      return true;
    }

    return false;
  }

  // An inner class for a cell
```

inner class Cell

```java
  public class Cell extends Pane {
    // Token used for this cell
    private char token = ' ';

    public Cell() {
      setStyle("-fx-border-color: black");
      this.setPrefSize(2000, 2000);
```

register listener

```java
      this.setOnMouseClicked(e -> handleMouseClick());
    }

    /** Return token */
    public char getToken() {
      return token;
    }

    /** Set a new token */
    public void setToken(char c) {
      token = c;

```

display X

```java
      if (token == 'X') {
        Line line1 = new Line(10, 10,
          this.getWidth() - 10, this.getHeight() - 10);
        line1.endXProperty().bind(this.widthProperty().subtract(10));
        line1.endYProperty().bind(this.heightProperty().subtract(10));
        Line line2 = new Line(10, this.getHeight() - 10,
          this.getWidth() - 10, 10);
        line2.startYProperty().bind(
          this.heightProperty().subtract(10));
        line2.endXProperty().bind(this.widthProperty().subtract(10));

        // Add the lines to the pane
        this.getChildren().addAll(line1, line2);
      }
```

display O

```java
      else if (token == 'O') {
        Ellipse ellipse = new Ellipse(this.getWidth() / 2,
          this.getHeight() / 2, this.getWidth() / 2 - 10,
          this.getHeight() / 2 - 10);
        ellipse.centerXProperty().bind(
          this.widthProperty().divide(2));
        ellipse.centerYProperty().bind(
          this.heightProperty().divide(2));
        ellipse.radiusXProperty().bind(
          this.widthProperty().divide(2).subtract(10));
```

```
126         ellipse.radiusYProperty().bind(
127           this.heightProperty().divide(2).subtract(10));
128         ellipse.setStroke(Color.BLACK);
129         ellipse.setFill(Color.WHITE);
130
131         getChildren().add(ellipse); // Add the ellipse to the pane
132       }
133     }
134
135     /* Handle a mouse click event */
136     private void handleMouseClick() {
137       // If cell is empty and game is not over
138       if (token == ' ' && whoseTurn != ' ') {
139         setToken(whoseTurn); // Set token in the cell
140
141         // Check game status
142         if (isWon(whoseTurn)) {
143           lblStatus.setText(whoseTurn + " won! The game is over");
144           whoseTurn = ' '; // Game is over
145         }
146         else if (isFull()) {
147           lblStatus.setText("Draw! The game is over");
148           whoseTurn = ' '; // Game is over
149         }
150         else {
151           // Change the turn
152           whoseTurn = (whoseTurn == 'X') ? 'O' : 'X';
153           // Display whose turn
154           lblStatus.setText(whoseTurn + "'s turn");
155         }
156       }
157     }
158   }
159 }
```

handle mouse click

The **TicTacToe** class initializes the user interface with nine cells placed in a grid pane (lines 25–28). A label named **lblStatus** is used to show the status of the game (line 20). The variable **whoseTurn** (line 14) is used to track the next type of token to be placed in a cell. The methods **isFull** (lines 42–49) and **isWon** (lines 52–80) are for checking the status of the game.

Since **Cell** is an inner class in **TicTacToe**, the variable **whoseTurn** and methods **isFull** and **isWon** defined in **TicTacToe** can be referenced from the **Cell** class. The inner class makes programs simple and concise. If **Cell** were not defined as an inner class of **TicTacToe**, you would have to pass an object of **TicTacToe** to **Cell** in order for the variables and methods in **TicTacToe** to be used in **Cell**.

The listener for the mouse-click action is registered for the cell (line 90). If an empty cell is clicked and the game is not over, a token is set in the cell (line 138). If the game is over, **whoseTurn** is set to **' '** (lines 144 and 148). Otherwise, **whoseTurn** is alternated to a new turn (line 152).

Tip

Use an incremental approach in developing and testing a Java project of this kind. For example, this program can be divided into five steps:

incremental development and testing

1. Lay out the user interface and display a fixed token X on a cell.
2. Enable the cell to display a fixed token X upon a mouse click.
3. Coordinate between the two players so as to display tokens X and O alternately.
4. Check whether a player wins, or whether all the cells are occupied without a winner.
5. Implement displaying a message on the label upon each move by a player.

Check Point

16.12.1 When the game starts, what value is in `whoseTurn`? When the game is over, what value is in `whoseTurn`?

16.12.2 What happens when the user clicks on an empty cell if the game is not over? What happens when the user clicks on an empty cell if the game is over?

16.12.3 How does the program check whether a player wins? How does the program check whether all cells are filled?

16.13 Video and Audio

Key Point

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.

VideoNote
Use Media, MediaPlayer, and MediaView

Media (video and audio) is essential in developing rich GUI applications. JavaFX provides the `Media`, `MediaPlayer`, and `MediaView` classes for working with media. Currently, JavaFX supports MP3, AIFF, WAV, and MPEG-4 audio formats and FLV and MPEG-4 video formats.

The `Media` class represents a media source with properties `duration`, `width`, and `height`, as shown in Figure 16.30. You can construct a `Media` object from an Internet URL string.

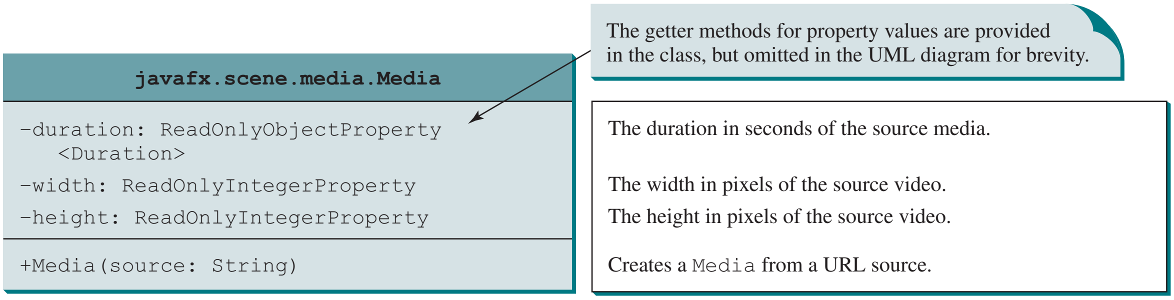

The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.
javafx.scene.media.Media
-duration: ReadOnlyObjectProperty <Duration>
-width: ReadOnlyIntegerProperty
-height: ReadOnlyIntegerProperty
+Media(source: String)
The duration in seconds of the source media.
The width in pixels of the source video.
The height in pixels of the source video.
Creates a Media from a URL source.

FIGURE 16.30 `Media` represents a media source such as a video or an audio.

The `MediaPlayer` class plays and controls the media with properties such as `autoPlay`, `currentCount`, `cycleCount`, `mute`, `volume`, and `totalDuration`, as shown in Figure 16.31. You can construct a `MediaPlayer` object from a media and use the `pause()` and `play()` methods to pause and resume playing.

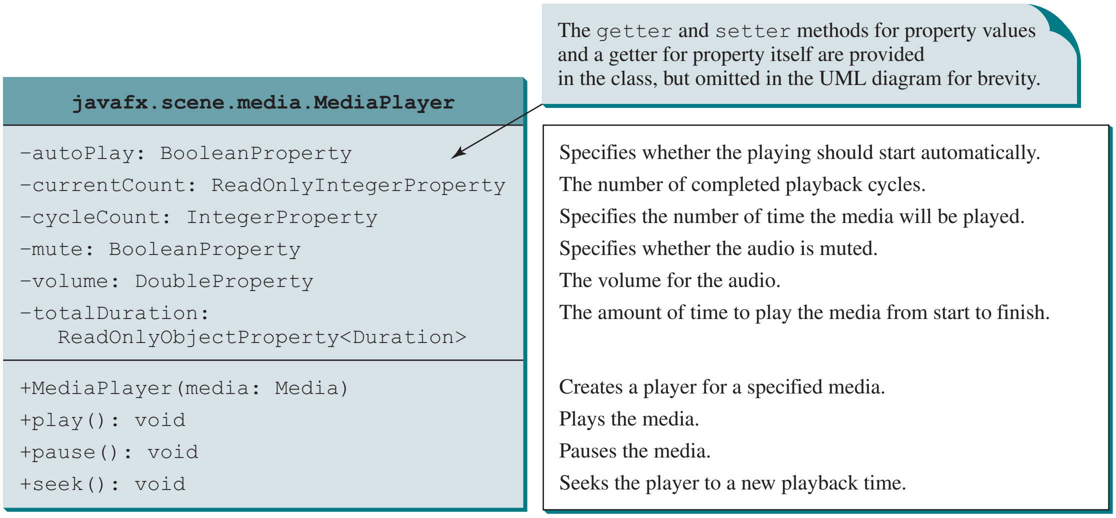
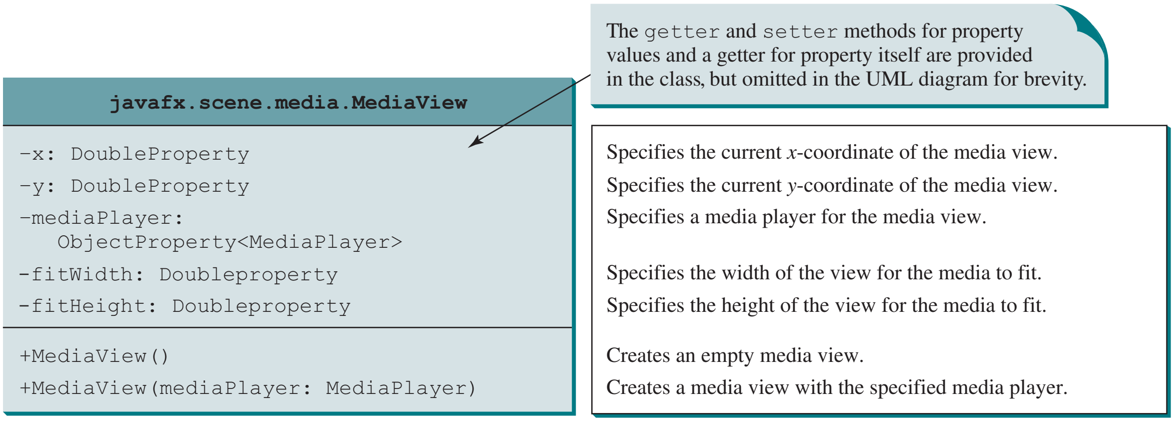

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.
javafx.scene.media.MediaPlayer
-autoPlay: BooleanProperty
-currentCount: ReadOnlyIntegerProperty
-cycleCount: IntegerProperty
-mute: BooleanProperty
-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.

FIGURE 16.31 `MediaPlayer` plays and controls a media.

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, as shown in Figure 16.32.


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.

FIGURE 16.32 MediaView provides the properties for viewing the media.

Listing 16.14 gives an example that displays a video in a view, as shown in Figure 16.33. 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.

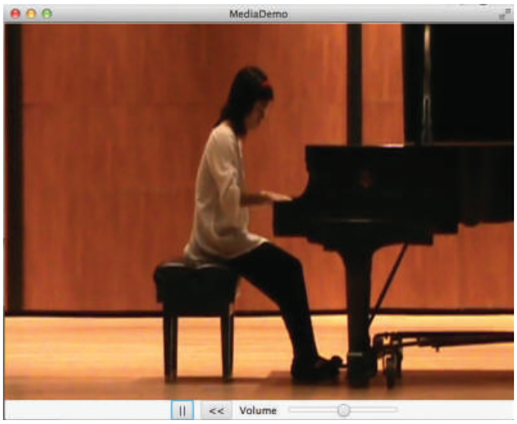


FIGURE 16.33 The program controls and plays a video.

LISTING 16.14 MediaDemo.java

```java
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Region;
import javafx.scene.media.Media;
```

create a media
create a media player
create a media view
create a play/pause button
add handler for button action
play media
pause media
create a rewind button
create a handler for rewinding
create a slider for volume
set current volume
bind volume with slider
add buttons, slider to hBox
place media view in a pane

```
12 import javafx.scene.media.MediaPlayer;
13 import javafx.scene.media.MediaView;
14 import javafx.util.Duration;
15
16 public class MediaDemo extends Application {
17   private static final String MEDIA_URL =
18     "http://liveexample.pearsoncmg.com/common/sample.mp4";
19
20   @Override // Override the start method in the Application class
21   public void start(Stage primaryStage) {
22     Media media = new Media(MEDIA_URL);
23     MediaPlayer mediaPlayer = new MediaPlayer(media);
24     MediaView mediaView = new MediaView(mediaPlayer);
25
26     Button playButton = new Button(">");
27     playButton.setOnAction(e -> {
28       if (playButton.getText().equals(">")) {
29         mediaPlayer.play();
30         playButton.setText("||");
31       } else {
32         mediaPlayer.pause();
33         playButton.setText(">");
34       }
35     });
36
37     Button rewindButton = new Button("<<");
38     rewindButton.setOnAction(e -> mediaPlayer.seek(Duration.ZERO));
39
40     Slider slVolume = new Slider();
41     slVolume.setPrefWidth(150);
42     slVolume.setMaxWidth(Region.USE_PREF_SIZE);
43     slVolume.setMinWidth(30);
44     slVolume.setValue(50);
45     mediaPlayer.volumeProperty().bind(
46       slVolume.valueProperty().divide(100));
47
48     HBox hBox = new HBox(10);
49     hBox.setAlignment(Pos.CENTER);
50     hBox.getChildren().addAll(playButton, rewindButton,
51       new Label("Volume"), slVolume);
52
53     BorderPane pane = new BorderPane();
54     pane.setCenter(mediaView);
55     pane.setBottom(hBox);
56
57     // Create a scene and place it in the stage
58     Scene scene = new Scene(pane, 650, 500);
59     primaryStage.setTitle("MediaDemo"); // Set the stage title
60     primaryStage.setScene(scene); // Place the scene in the stage
61     primaryStage.show(); // Display the stage
62   }
63 }
```

The source of the media is a URL string defined in lines 17 and 18. The program creates a `Media` object from this URL (line 22), a `MediaPlayer` from the `Media` object (line 23), and a `MediaView` from the `MediaPlayer` object (line 24). The relationship among these three objects is shown in Figure 16.34.

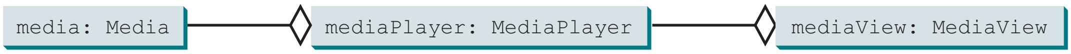


Figure 16.34 The media represents the source, the media player controls the playing, and the media view displays the video.

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

A play button is created (line 26) to play/pause the media (line 29). The button's text is changed to `||` (line 30) if the button's current text is `>` (line 28). If the button's current text is `||`, it is changed to `>` (line 33) and the player is paused (line 32).

A rewind button is created (line 37) to reset the playback time to the beginning of the media stream by invoking `seek(Duration.ZERO)` (line 38).

A slider is created (line 40) to set the volume. The media player's volume property is bound to the slider (lines 45 and 46).

The buttons and slider are placed in an `HBox` (lines 48–51) and the media view is placed in the center of the border pane (line 54) and the `HBox` is placed at the bottom of the border pane (line 55).

Check Point

16.13.1 How do you create a `Media` from a URL? How do you create a `MediaPlayer`? How do you create a `MediaView`?

16.13.2 If the URL is typed as liveexample.pearsoncmg.com/common/sample.mp4 without http:// in front of it, will it work?

16.13.3 Can you place a `Media` in multiple `MediaPlayer`s? Can you place a `MediaPlayer` in multiple `MediaView`s? Can you place a `MediaView` in multiple `Pane`s?

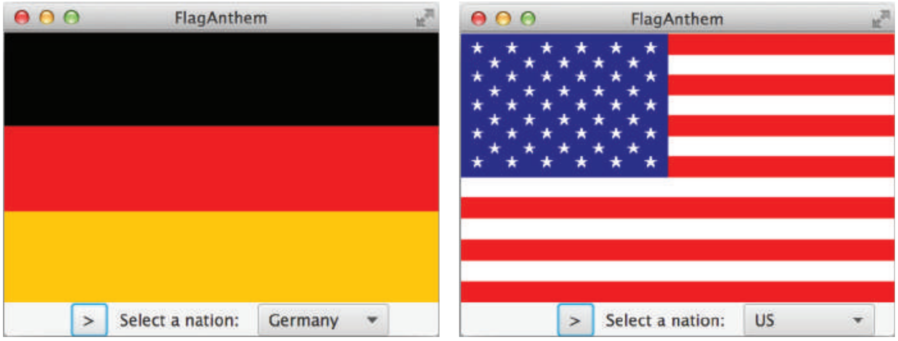
16.14 Case Study: National Flags and Anthems

Key Point

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

The images for seven national flags, named **flag0.gif**, **flag1.gif**, . . . , **flag6.gif** for Denmark, Germany, China, India, Norway, the United Kingdom, and the United States are stored under http://liveexample.pearsoncmg.com/common/image. The audio consists of national anthems for these seven nations, named **anthem0.mp3**, **anthem1.mp3**, . . . , **anthem6.mp3**. They are stored under http://liveexample.pearsoncmg.com/common/audio.

The program enables the user to select a nation from a combo box, then displays its flag and plays its anthem. The user can suspend the audio by clicking the `||` button, and resume it by clicking the `<` button, as shown in Figure 16.35.



Select a nation: Germany

Select a nation: US

Figure 16.35 The program displays a national flag and plays its anthem. *Source*: booka/Fotolia.

The program is given in Listing 16.15.

LISTING 16.15 FlagAnthem.java

```java
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.stage.Stage;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;

public class FlagAnthem extends Application {
  private final static int NUMBER_OF_NATIONS = 7;
  private final static String URLBase =
    "https://liveexample.pearsoncmg.com/common";
  private int currentIndex = 0;

  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    Image[] images = new Image[NUMBER_OF_NATIONS];
    MediaPlayer[] mp = new MediaPlayer[NUMBER_OF_NATIONS];

    // 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();
      }
    });

    ImageView imageView = new ImageView(images[currentIndex]);
    ComboBox<String> cboNation = new ComboBox<>();
    ObservableList<String> items = FXCollections.observableArrayList
      ("Denmark", "Germany", "China", "India", "Norway", "UK", "US");
    cboNation.getItems().addAll(items);
    cboNation.setValue(items.get(0));
    cboNation.setOnAction(e -> {
      mp[currentIndex].stop();
      currentIndex = items.indexOf(cboNation.getValue());
      imageView.setImage(images[currentIndex]);
      mp[currentIndex].play();
```

URLBase for image and audio

track current image/audio

image array

media player array

load image

load audio

create play button

handle button action

play audio

pause audio

create image view

create combo box

create observable list

process combo selection

choose a new nation

play audio

```
58        btPlayPause.setText("||");
59      });
60
61      HBox hBox = new HBox(10);
62      hBox.getChildren().addAll(btPlayPause,
63        new Label("Select a nation: "), cboNation);
64      hBox.setAlignment(Pos.CENTER);
65
66      // Create a pane to hold nodes
67      BorderPane pane = new BorderPane();
68      pane.setCenter(imageView);
69      pane.setBottom(hBox);
70
71      // Create a scene and place it in the stage
72      Scene scene = new Scene(pane, 350, 270);
73      primaryStage.setTitle("FlagAnthem"); // Set the stage title
74      primaryStage.setScene(scene); // Place the scene in the stage
75      primaryStage.show(); // Display the stage
76      mp[currentIndex].play(); // Play the current selected anthem
77    }
78  }
```

The program loads the image and audio from the Internet (lines 29–33). A play/pause button is created to control the playing of the audio (line 35). When the button is clicked, if the button's current text is `>` (line 37), its text is changed to `||` (line 38) and the player is paused (line 39); If the button's current text is `||`, it is changed to `>` (line 42) and the player is paused (line 43).

An image view is created to display a flag image (line 47). A combo box is created for selecting a nation (line 48–51). When a new country name in the combo box is selected, the current audio is stopped (line 54), the newly selected nation's image is displayed (line 56) and the new anthem is played (line 57).

JavaFX also provides the `AudioClip` class for creating auto clips. An `AudioClip` object can be created using `new AudioClip(URL)`. An audio clip stores the audio in memory. `AudioClip` is more efficient for playing a small audio clip in the program than using `MediaPlayer`. `AudioClip` has the similar methods as in the `MediaPlayer` class.

Check Point

16.14.1 In Listing 16.15, which code sets the initial image icon and which code plays the audio?

16.14.2 In Listing 16.15, what does the program do when a new nation is selected in the combo box?

Chapter Summary

1. The abstract `Labeled` class is the base class for `Label`, `Button`, `CheckBox`, and `RadioButton`. It defines properties `alignment`, `contentDisplay`, `text`, `graphic`, `graphicTextGap`, `textFill`, `underline`, and `wrapText`.

2. The abstract `ButtonBase` class is the base class for `Button`, `CheckBox`, and `RadioButton`. It defines the `onAction` property for specifying a handler for action events.

3. The abstract `TextInputContorl` class is the base class for `TextField` and `TextArea`. It defines the properties `text` and `editable`.

4. A `TextField` fires an action event when clicking the *Enter* key with the text field focused. A `TextArea` is often used for editing a multiline text.

5. `ComboBox<T>` and `ListView<T>` are generic classes for storing elements of type `T`. The elements in a combo box or a list view are stored in an observable list.

6. A **ComboBox** fires an action event when a new item is selected.

7. You can set a single item or multiple items selection for a **ListView** and add a listener for processing selected items.

8. You can use a **ScrollBar** or **Slider** to select a range of values and add a listener to the **value** property to respond to the change of the value.

9. JavaFX provides the **Media** class for loading a media, the **MediaPlayer** class for controlling a media, and the **MediaView** for displaying a media.

Quiz

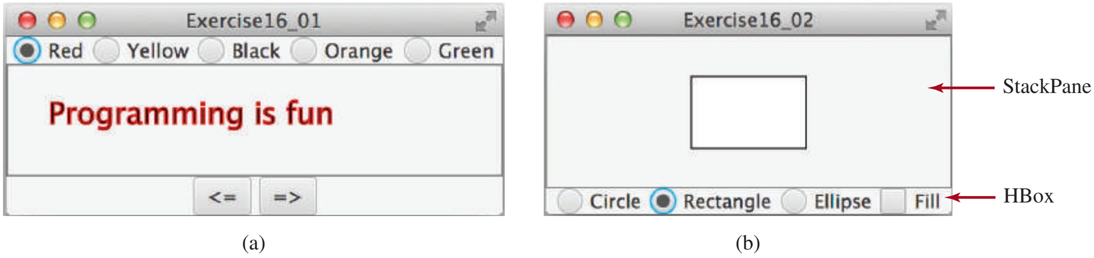
Answer the quiz for this chapter online at the book Companion Website.

MyProgrammingLab™

Programming Exercises

Sections 16.2–16.5

***16.1** *(Use radio buttons)* Write a GUI program as shown in Figure 16.36a. You can use buttons to move the message to the left and right and use the radio buttons to change the color for the message displayed.

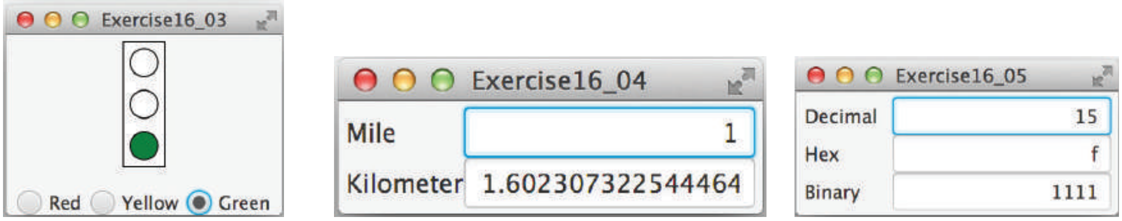


(a) (b)

Figure 16.36 (a) The <= and => buttons move the message, and the radio buttons change the color for the message. (b) The program displays a circle, rectangle, and ellipse when you select a shape type. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

***16.2** *(Select geometric figures)* Write a program that draws various figures, as shown in Figure 16.36b. The user selects a figure from a radio button and uses a check box to specify whether it is filled.

****16.3** *(Traffic lights)* Write a program that simulates a traffic light. The program lets the user select one of three lights: red, yellow, or green. When a radio button is selected, the light is turned on. Only one light can be on at a time (see Figure 16.37a). No light is on when the program starts.



Mile 1
Kilometer 1.602307322544464

Decimal 15
Hex f
Binary 1111

(a) (b) (c)

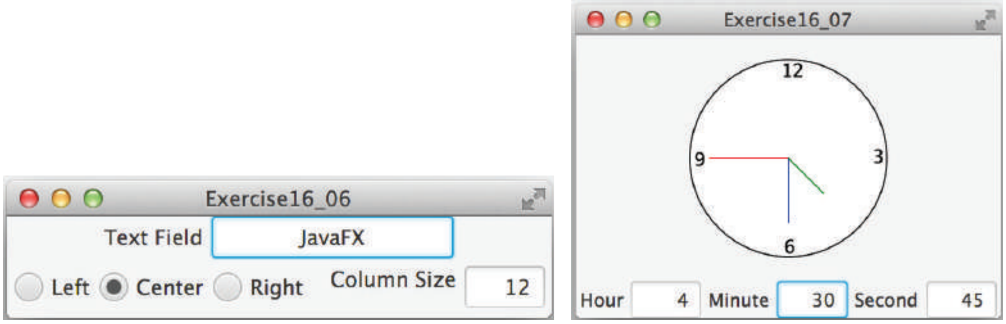
FIGURE 16.37 (a) The radio buttons are grouped to let you turn only one light on at a time. (b) The program converts miles to kilometers and vice versa. (c) The program converts among decimal, hex, and binary numbers. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

***16.4** *(Create a miles/kilometers converter)* Write a program that converts miles and kilometers, as shown in Figure 16.37b. If you enter a value in the Mile text field and press the *Enter* key, the corresponding kilometer measurement is displayed in the Kilometer text field. Likewise, if you enter a value in the Kilometer text field and press the *Enter* key, the corresponding miles is displayed in the Mile text field.

***16.5** *(Convert numbers)* Write a program that converts among decimal, hex, and binary numbers, as shown in Figure 16.37c. When you enter a decimal value in the decimal-value text field and press the *Enter* key, its corresponding hex and binary numbers are displayed in the other two text fields. Likewise, you can enter values in the other fields and convert them accordingly. (*Hint*: Use the `Integer.parseInt(s, radix)` method to parse a string to a decimal and use `Integer.toHexString(decimal)` and `Integer.toBinaryString(decimal)` to obtain a hex number or a binary number from a decimal.)

***16.6** *(Demonstrate* `TextField` *properties)* Write a program that sets the horizontal-alignment and column-size properties of a text field dynamically, as shown in Figure 16.38a.

VideoNote
Use radio buttons and text fields



Text Field JavaFX
Column Size 12

Hour 4 Minute 30 Second 45

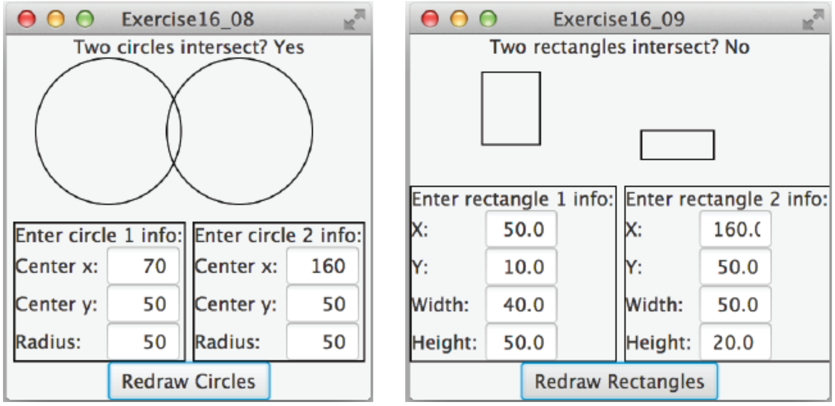
(a) (b)

FIGURE 16.38 (a) You can set a text field's properties for the horizontal alignment and column size dynamically. (b) The program displays the time specified in the text fields. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

***16.7** *(Set clock time)* Write a program that displays a clock and sets the time with the input from three text fields, as shown in Figure 16.38b. Use the `ClockPane` in Listing 14.21. Resize the clock to the center of the pane.

****16.8** *(Geometry: two circles intersect?)* Write a program that enables the user to specify the location and size of the circles, and displays whether the two circles intersect, as shown in Figure 16.39a. Enable the user to point the mouse inside a circle and drag it. As the circle is being dragged, the circle's center coordinates in the text fields are updated.

****16.9** *(Geometry: two rectangles intersect?)* Write a program that enables the user to specify the location and size of the rectangles and displays whether the two rectangles intersect, as shown in Figure 16.39b. Enable the user to point the mouse inside a rectangle and drag it. As the rectangle is being dragged, the rectangle's center coordinates in the text fields are updated.



Exercise16_08

Two circles intersect? Yes

Enter circle 1 info:
Center x: 70
Center y: 50
Radius: 50

Enter circle 2 info:
Center x: 160
Center y: 50
Radius: 50

Exercise16_09

Two rectangles intersect? No

Enter rectangle 1 info:
X: 50.0
Y: 10.0
Width: 40.0
Height: 50.0

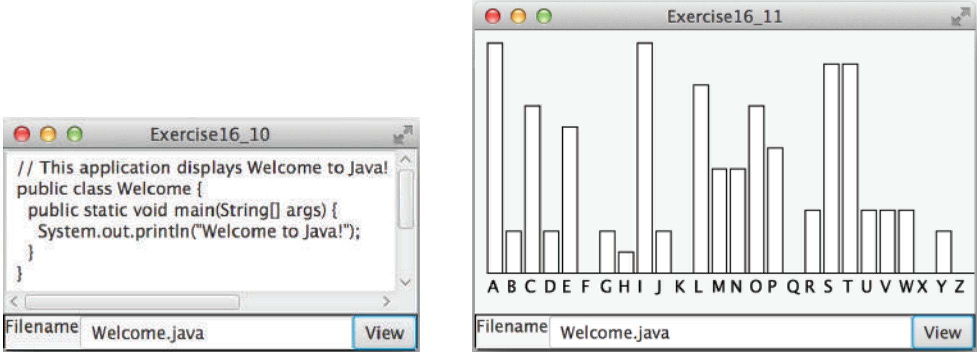
Enter rectangle 2 info:
X: 160.(
Y: 50.0
Width: 50.0
Height: 20.0

(a) (b)

FIGURE 16.39 Check whether two circles and two rectangles are overlapping. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

Sections 16.6–16.8

****16.10** *(Text viewer)* Write a program that displays a text file in a text area, as shown in Figure 16.40a. The user enters a file name in a text field and clicks the *View* button; the file is then displayed in a text area.



Exercise16_10

```
// This application displays Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Filename Welcome.java

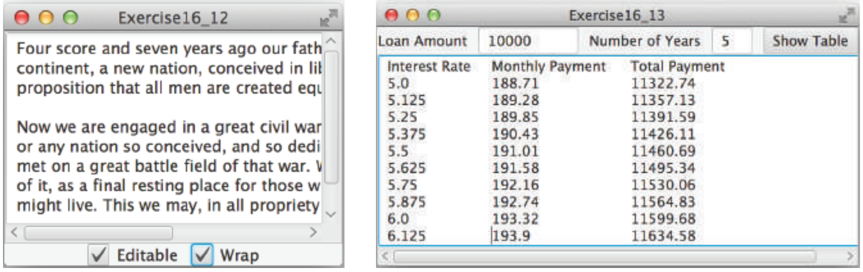
Filename Welcome.java

(a) (b)

FIGURE 16.40 (a) The program displays the text from a file in a text area. (b) The program displays a histogram that shows the occurrences of each letter in the file. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

****16.11** *(Create a histogram* for *occurrences of letters)* Write a program that reads a file and displays a histogram to show the occurrences of each letter in the file, as shown in Figure 16.40b. The file name is entered from a text field. Pressing the *Enter* key on the text field causes the program to start to read, process the file, and display the histogram. The histogram is displayed in the center of the window. Define a class named `Histogram` that extends `Pane`. The class contains the property `counts` that is an array of 26 elements. `counts[0]` stores the number of `A`, `counts[1]` the number of `B`, and so on. The class also contains a setter method for setting a new `counts` and displaying the histogram for the new `counts`.

***16.12** *(Demonstrate `TextArea` properties)* Write a program that demonstrates the properties of a text area. The program uses a check box to indicate whether the text is wrapped onto next line, as shown in Figure 16.41a.



```
Four score and seven years ago our fath
continent, a new nation, conceived in lib
proposition that all men are created equ

Now we are engaged in a great civil war
or any nation so conceived, and so dedi
met on a great battle field of that war. V
of it, as a final resting place for those w
might live. This we may, in all propriety
```

☑ Editable ☑ Wrap

(a)

Loan Amount 10000 Number of Years 5 Show Table

```
Interest Rate   Monthly Payment   Total Payment
5.0             188.71            11322.74
5.125           189.28            11357.13
5.25            189.85            11391.59
5.375           190.43            11426.11
5.5             191.01            11460.69
5.625           191.58            11495.34
5.75            192.16            11530.06
5.875           192.74            11564.83
6.0             193.32            11599.68
6.125           193.9             11634.58
```

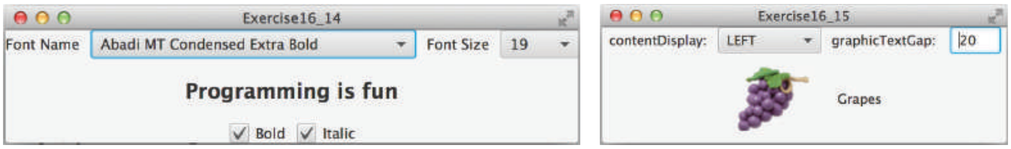
(b)

FIGURE 16.41 (a) You can set the options to enable text editing and text wrapping. (b) The program displays a table for monthly payments and total payments on a given loan based on various interest rates. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

***16.13** *(Compare loans with various interest rates)* Rewrite Programming Exercise 5.21 to create a GUI, as shown in Figure 16.41b. Your program should let the user enter the loan amount and loan period in the number of years from text fields, and it should display the monthly and total payments for each interest rate starting from `5`% to `8`%, with increments of one-eighth, in a text area.

****16.14** *(Select a font)* Write a program that can dynamically change the font of a text in a label displayed on a stack pane. The text can be displayed in bold and italic at the same time. You can select the font name or font size from combo boxes, as shown in Figure 16.42a. The available font names can be obtained using `Font.getFontNames()`. The combo box for the font size is initialized with numbers from `1` to `100`.

VideoNote
Set fonts



Font Name: Abadi MT Condensed Extra Bold Font Size: 19

Programming is fun

☑ Bold ☑ Italic

(a)

contentDisplay: LEFT graphicTextGap: 20

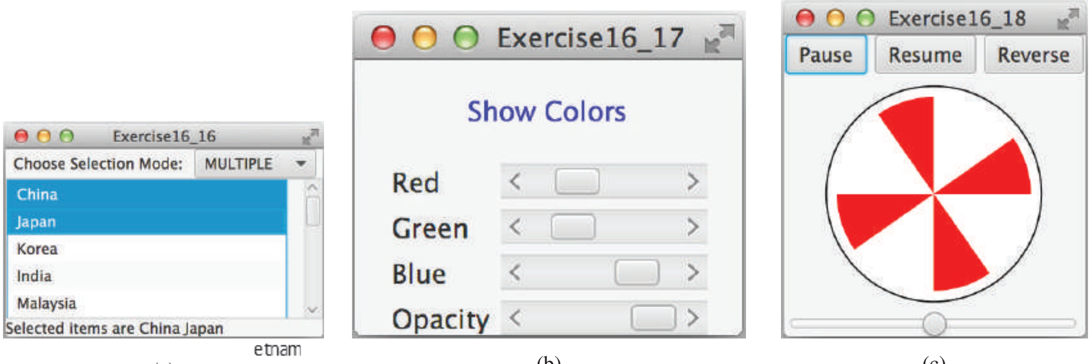
Grapes

(b)

FIGURE 16.42 You can dynamically set the font for the message. (b) You can set the alignment and text-position properties of a label dynamically. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

****16.15** *(Demonstrate* `Label` *properties)* Write a program to let the user dynamically set the properties `contentDisplay` and `graphicTextGap`, as shown in Figure 16.42b.

***16.16** *(Use* `ComboBox` *and* `ListView`*)* Write a program that demonstrates selecting items in a list. The program uses a combo box to specify a selection mode, as shown in Figure 16.43a. When you select items, they are displayed in a label below the list.



Selected items are China Japan

(a) (b) (c)

Figure 16.43 (a) You can choose single or multiple selection modes in a list. (b) The color changes in the text as you adjust the scroll bars. (c) The program simulates a running fan. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

Sections 16.6–16.8

****16.17** *(Use* `ScrollBar` *and* `Slider`*)* Write a program that uses scroll bars or sliders to select the color for a text, as shown in Figure 16.43b. Four horizontal scroll bars are used for selecting the colors: red, green, blue, and opacity percentages.

****16.18** *(Simulation: a running fan)* Rewrite Programming Exercise 15.28 to add a slider to control the speed of the fan, as shown in Figure 16.43c.

****16.19** *(Control a group of fans)* Write a program that displays three fans in a group, with control buttons to start and stop all of them, as shown in Figure 16.44.

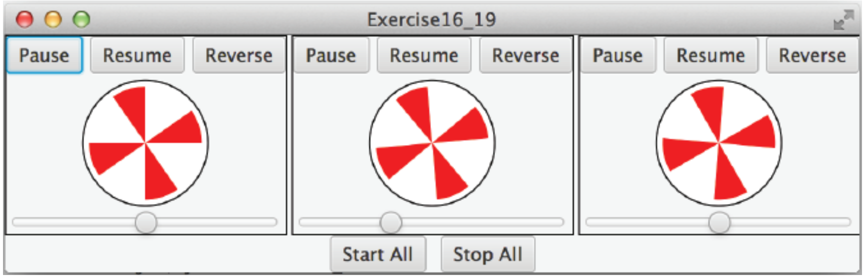
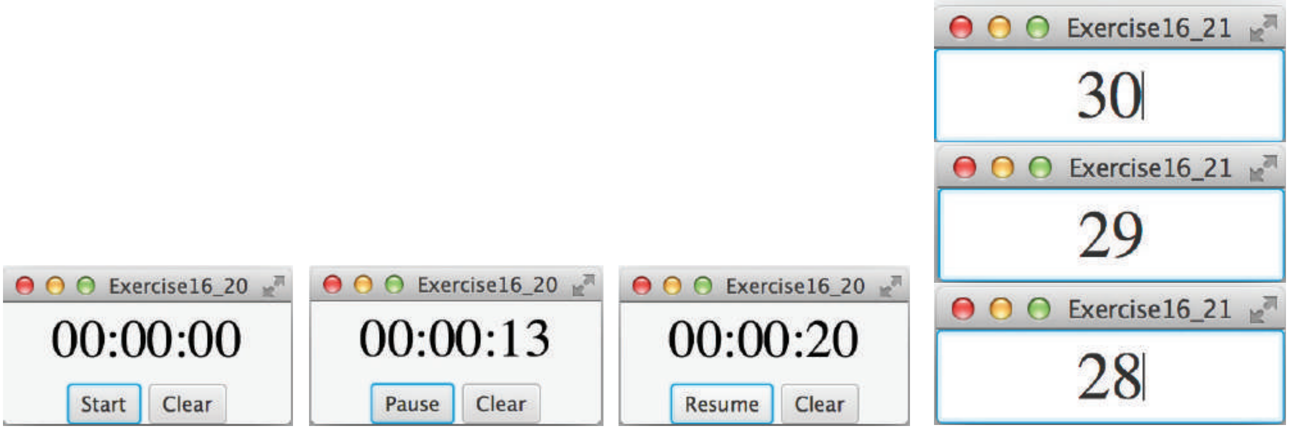


Figure 16.44 The program runs and controls a group of fans. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

***16.20** *(Count-up stopwatch)* Write a program that simulates a stopwatch, as shown in Figure 16.45a. When the user clicks the *Start* button, the button's label is changed to *Pause,* as shown in Figure 16.45b. When the user clicks the *Pause* button, the button's label is changed to *Resume,* as shown in Figure 16.45c. The *Clear* button resets the count to 0 and resets the button's label to *Start.*



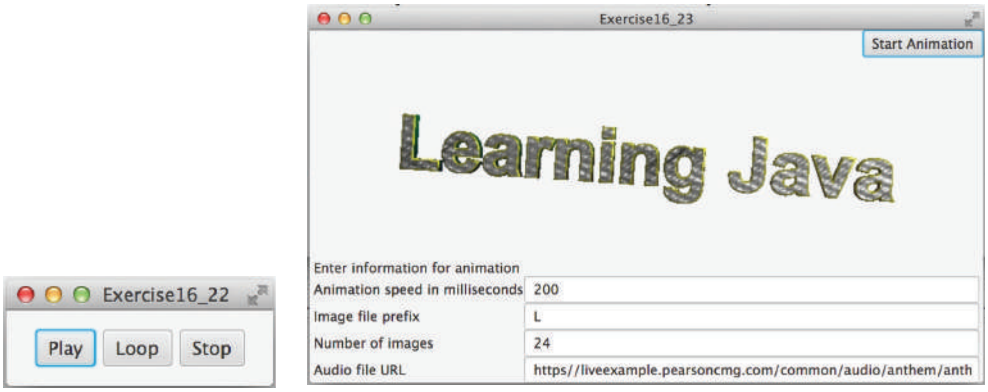
(a) (b) (c) (d)

FIGURE 16.45 (a–c) The program counts up the time. (d) The program counts down the time. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

***16.21** *(Count-down stopwatch)* Write a program that allows the user to enter time in seconds in the text field and press the *Enter* key to count down the seconds, as shown in Figure 16.45d. The remaining seconds are redisplayed every second. When the seconds are expired, the program starts to play music continuously.

16.22 *(Play, loop, and stop a sound clip)* Write a program that meets the following requirements:

- Get an audio file from the class directory using `AudioClip`.
- Place three buttons labeled *Play*, *Loop*, and *Stop*, as shown in Figure 16.46a.
- If you click the *Play* button, the audio file is played once. If you click the *Loop* button, the audio file keeps playing repeatedly. If you click the *Stop* button, the playing stops.



Enter information for animation
Animation speed in milliseconds 200
Image file prefix L
Number of images 24
Audio file URL https//liveexample.pearsoncmg.com/common/audio/anthem/anth

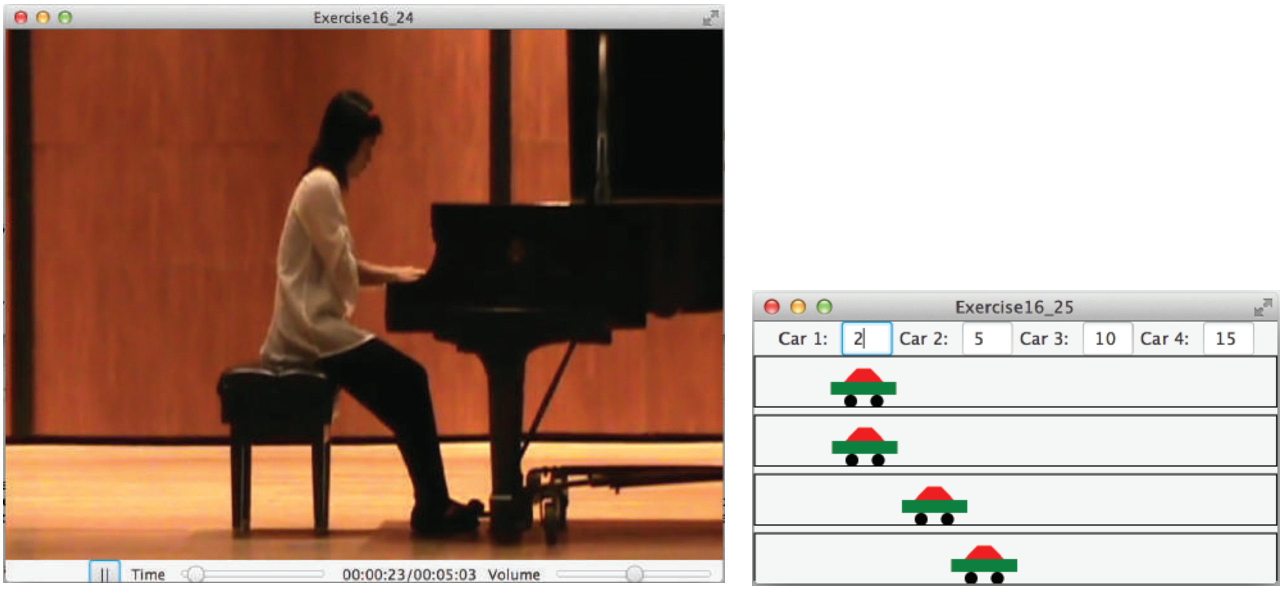
(a) (b)

FIGURE 16.46 (a) Click *Play* to play an audio clip once, click *Loop* to play an audio repeatedly, and click *Stop* to terminate playing. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission. (b) The program lets the user specify image files, an audio file, and the animation speed.

****16.23** *(Create an image animator with audio)* Create animation in Figure 16.46b to meet the following requirements:

- Allow the user to specify the animation speed in a text field.
- Get the number of images and image's file-name prefix from the user. For example, if the user enters **n** for the number of images and **L** for the image prefix, then the files are **L1.gif**, **L2.gif**, and so on, to **L*n*.gif**. Assume the images are stored in the **image** directory, a subdirectory of the program's class directory. The animation displays the images one after the other.
- Allow the user to specify an audio file URL. The audio is played while the animation runs.

****16.24** *(Revise Listing 16.14 MediaDemo.java)* Add a slider to enable the user to set the current time for the video and a label to display the current time and the total time for the video. As shown in Figure 16.47a, the total time is 5 minutes and 3 seconds and the current time is 3 minutes and 58 seconds. As the video plays, the slider value and current time are continuously updated.



Exercise16_24

Time 00:00:23/00:05:03 Volume

Exercise16_25

Car 1: 2 Car 2: 5 Car 3: 10 Car 4: 15

(a)

(b)

FIGURE 16.47 (a) A slider for current video time and a label to show the current time and total time are added. (b) You can set the speed for each car. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

****16.25** *(Racing cars)* Write a program that simulates four cars racing, as shown in Figure 16.47b. You can set the speed for each car, with a maximum of 100.

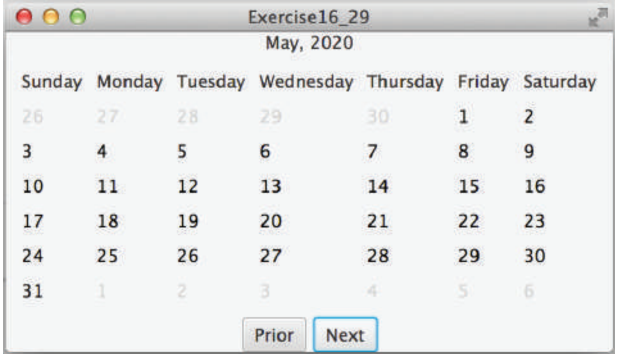
****16.26** *(Simulation: raise flag and play anthem)* Write a program that displays a flag rising up, as shown in Figure 15.15. As the national flag rises, play the national anthem. (You may use a flag image and anthem audio file from Listing 16.15.)

Comprehensive

****16.27** *(Display country flag and flag description)* Listing 16.8, ComboBoxDemo.java, gives a program that lets the user view a country's flag image and description by selecting the country from a combo box. The description is a string coded in the program. Rewrite the program to read the text description from a file. Suppose the descriptions are stored in the files **description0.txt**, . . . , and **description8.txt** under the **text** directory for the nine countries Canada, China, Denmark, France, Germany, India, Norway, the United Kingdom, and the United States, in this order.

****16.28** *(Slide show)* Programming Exercise 15.30 developed a slide show using images. Rewrite that program to develop a slide show using text files. Suppose that 10 text files named **slide0.txt**, **slide1.txt**, . . . , **slide9.txt** are stored in the **text** directory. Each slide displays the text from one file. Each slide is shown for one second, and the slides are displayed in order. When the last slide finishes, the first slide is redisplayed, and so on. Use a text area to display the slide.

*****16.29** *(Display a calendar)* Write a program that displays the calendar for the current month. You can use the *Prior* and *Next* buttons to show the calendar of the previous or next month. Display the dates in the current month in black and display the dates in the previous month and next month in gray, as shown in Figure 16.48.

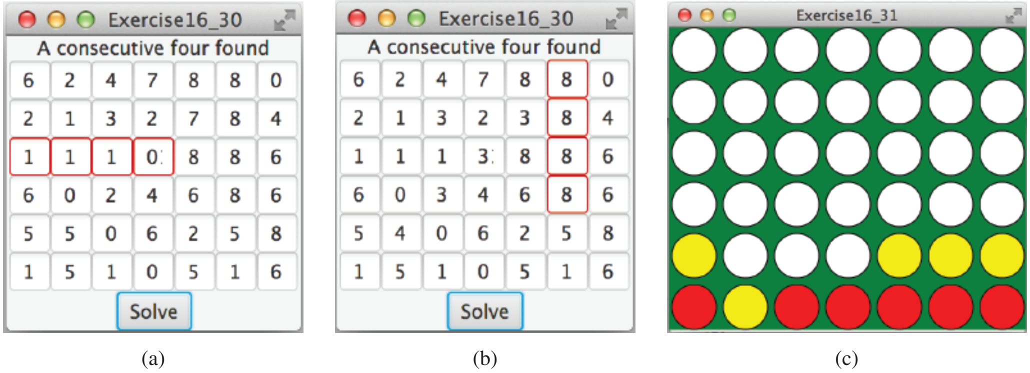


May, 2020

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

FIGURE 16.48 The program displays the calendar for the current month. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

****16.30** *(Pattern recognition: consecutive four equal numbers)* Write a GUI program for Programming Exercise 8.19, as shown in Figures 16.49a–b. Let the user enter the numbers in the text fields in a grid of 6 rows and 7 columns. The user can click the *Solve* button to highlight a sequence of four equal numbers, if it exists. Initially, the values in the text fields are randomly filled with numbers from 0 to 9.



A consecutive four found

6	2	4	7	8	8	0
2	1	3	2	7	8	4
1	1	1	0:	8	8	6
6	0	2	4	6	8	6
5	5	0	6	2	5	8
1	5	1	0	5	1	6

(a)

A consecutive four found

6	2	4	7	8	8	0
2	1	3	2	3	8	4
1	1	1	3:	8	8	6
6	0	3	4	6	8	6
5	4	0	6	2	5	8
1	5	1	0	5	1	6

(b)

(c)

FIGURE 16.49 (a and b) Clicking the *Solve* button highlights the four consecutive numbers in a row, a column, or a diagonal. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission. (c) The program enables two players to play the connect-four game.

***16.31 *(Game: connect four)* Programming Exercise 8.20 enables two players to play the connect-four game on the console. Rewrite a GUI version for the program, as shown in Figure 16.49c. The program enables two players to place red and yellow discs in turn. To place a disk, the player needs to click an available cell. An *available cell* is unoccupied and its downward neighbor is occupied. The program flashes the four winning cells if a player wins, and reports no winners if all cells are occupied with no winners.