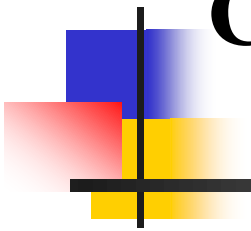


CCS3104
ADVANCED PROGRAMMING
Chapter 3 Advanced UI Controls and Multimedia

JavaFX UI Controls, Video and Audio

Motivations

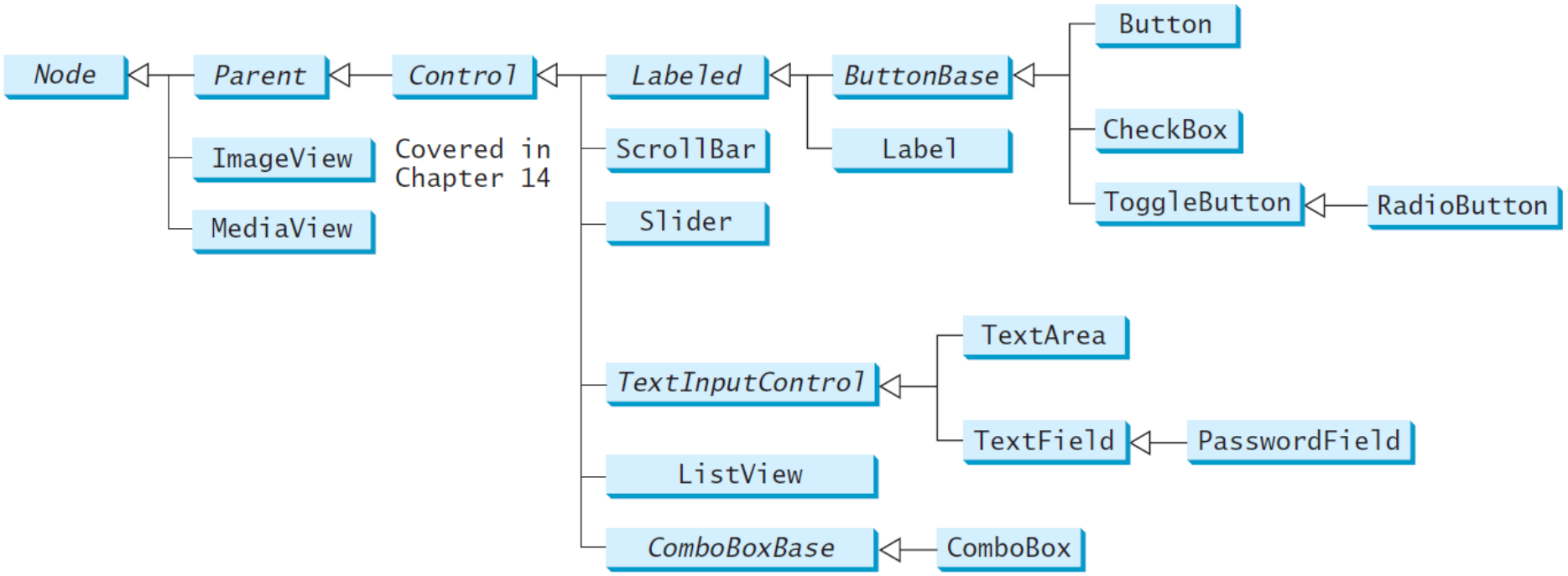
A graphical user interface (GUI) makes a system user-friendly and easy to use. Creating a GUI requires creativity and knowledge of how GUI components work. Since the GUI components in Java are very flexible and versatile, you can create a wide assortment of useful user interfaces.

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

Objectives

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

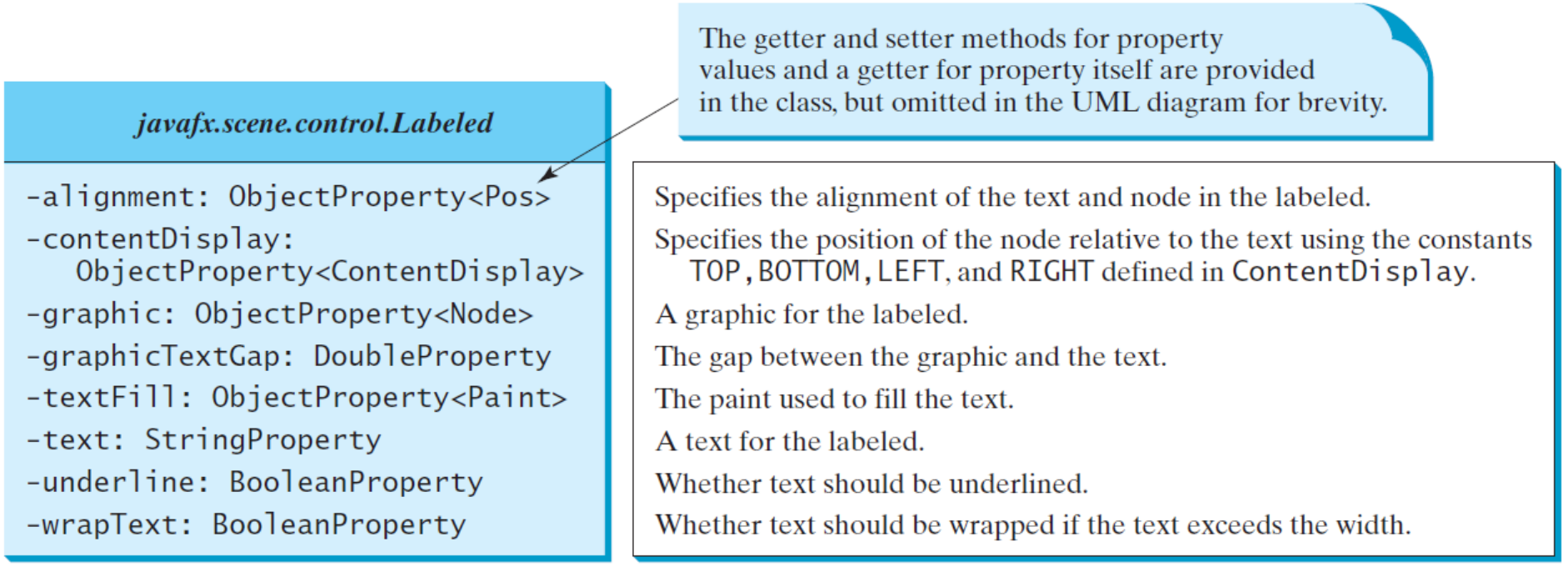
Frequently Used UI Controls



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

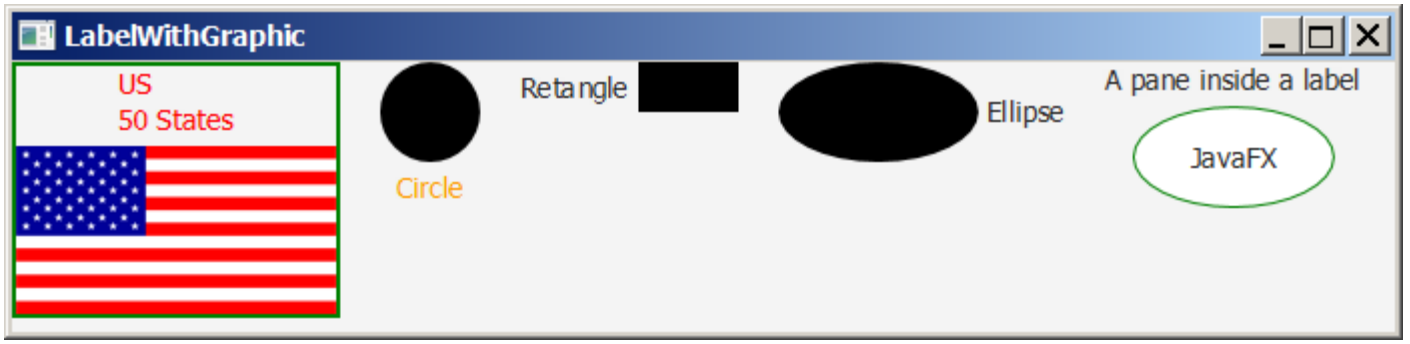
Labeled

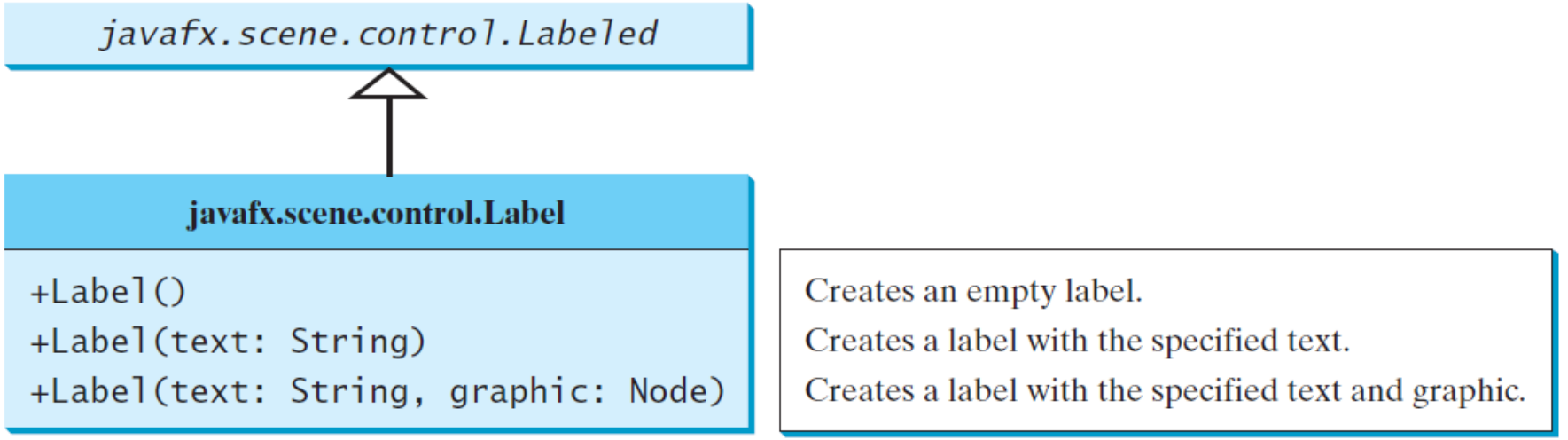
A *label* is a display area for a short text, a node, or both. It is often used to label other controls (usually text fields). Labels and buttons share many common properties. These common properties are defined in the **Labeled** class.



Label

The Label class defines labels.



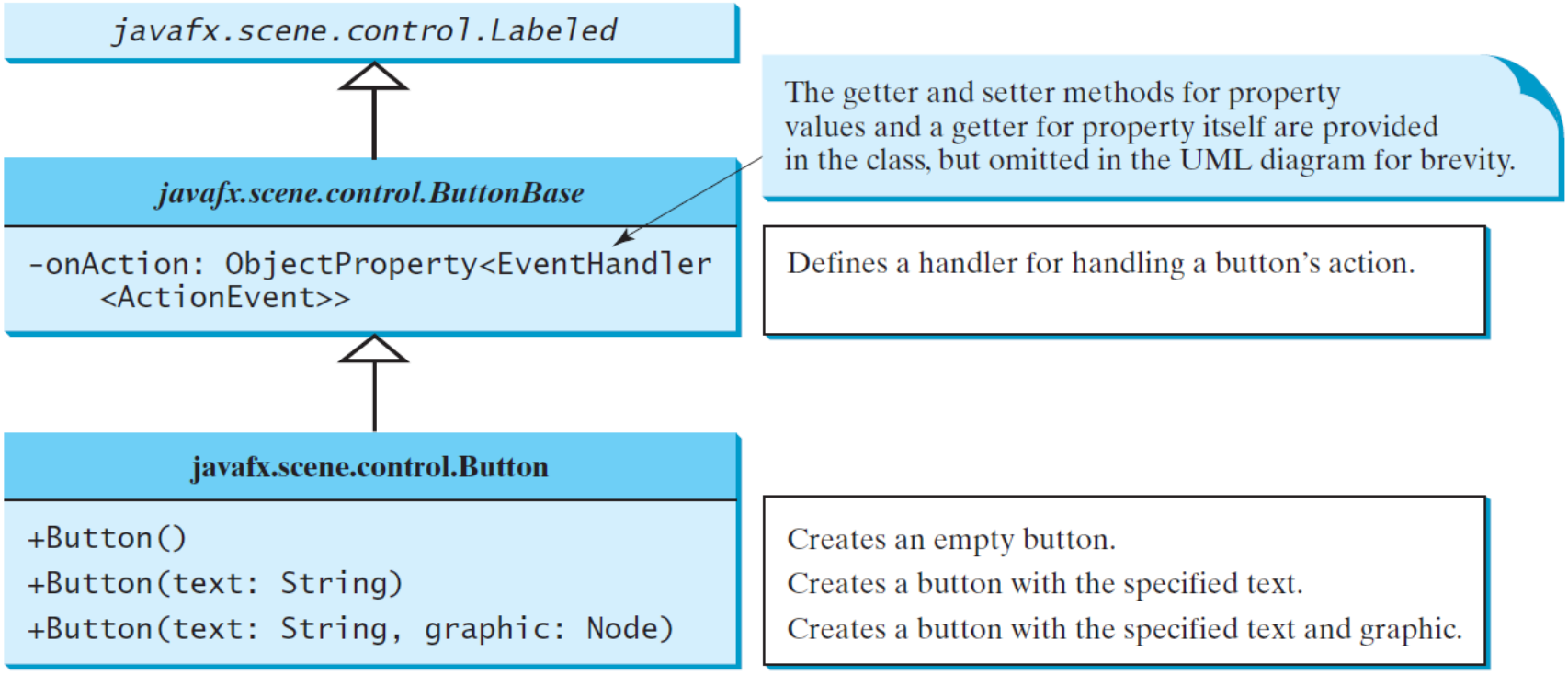


LabelWithGraphic Run

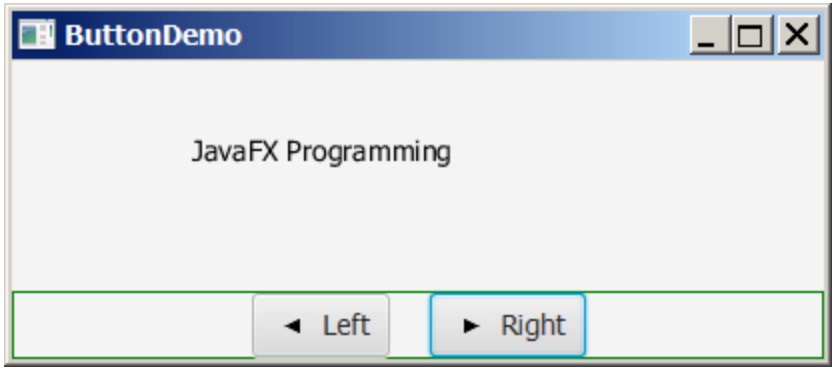
```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.ContentDisplay;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
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("Retangle", 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);

    Ellipse ellipse = new Ellipse(50, 50, 50, 25);
    ellipse.setStroke(Color.GREEN);
    ellipse.setFill(Color.WHITE);
    StackPane stackPane = new StackPane();
    stackPane.getChildren().addAll(ellipse, new Label("JavaFX"));
    Label lb5 = new Label("A pane inside a label", stackPane);
    lb5.setContentDisplay(ContentDisplay.BOTTOM);

    HBox pane = new HBox(20);
    pane.getChildren().addAll(lb1, lb2, lb3, lb4, lb5);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 450, 150);
    primaryStage.setTitle("LabelWithGraphic"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

ButtonBase and Button

A *button* is a control that triggers an action event when clicked. JavaFX provides regular buttons, toggle buttons, check box buttons, and radio buttons. The common features of these buttons are defined in **ButtonBase** and **Labeled** classes.



Button Example



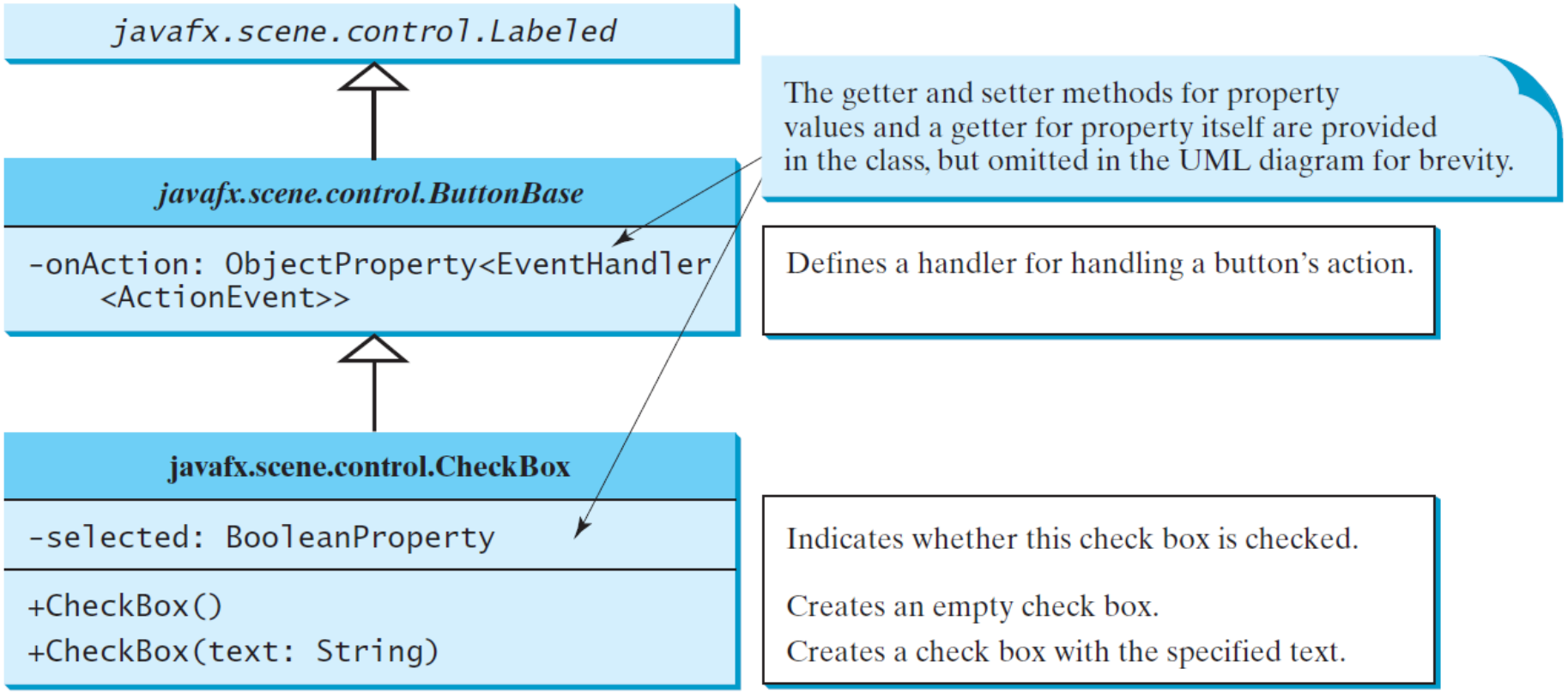
ButtonDemo Run

```
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;
  }
```

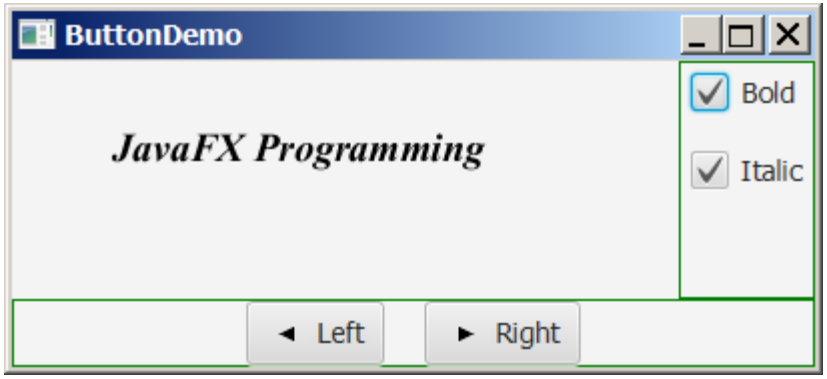
```
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a scene and place it in the stage
    Scene scene = new Scene(getPane(), 450, 200);
    primaryStage.setTitle("ButtonDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

CheckBox

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



CheckBox Example



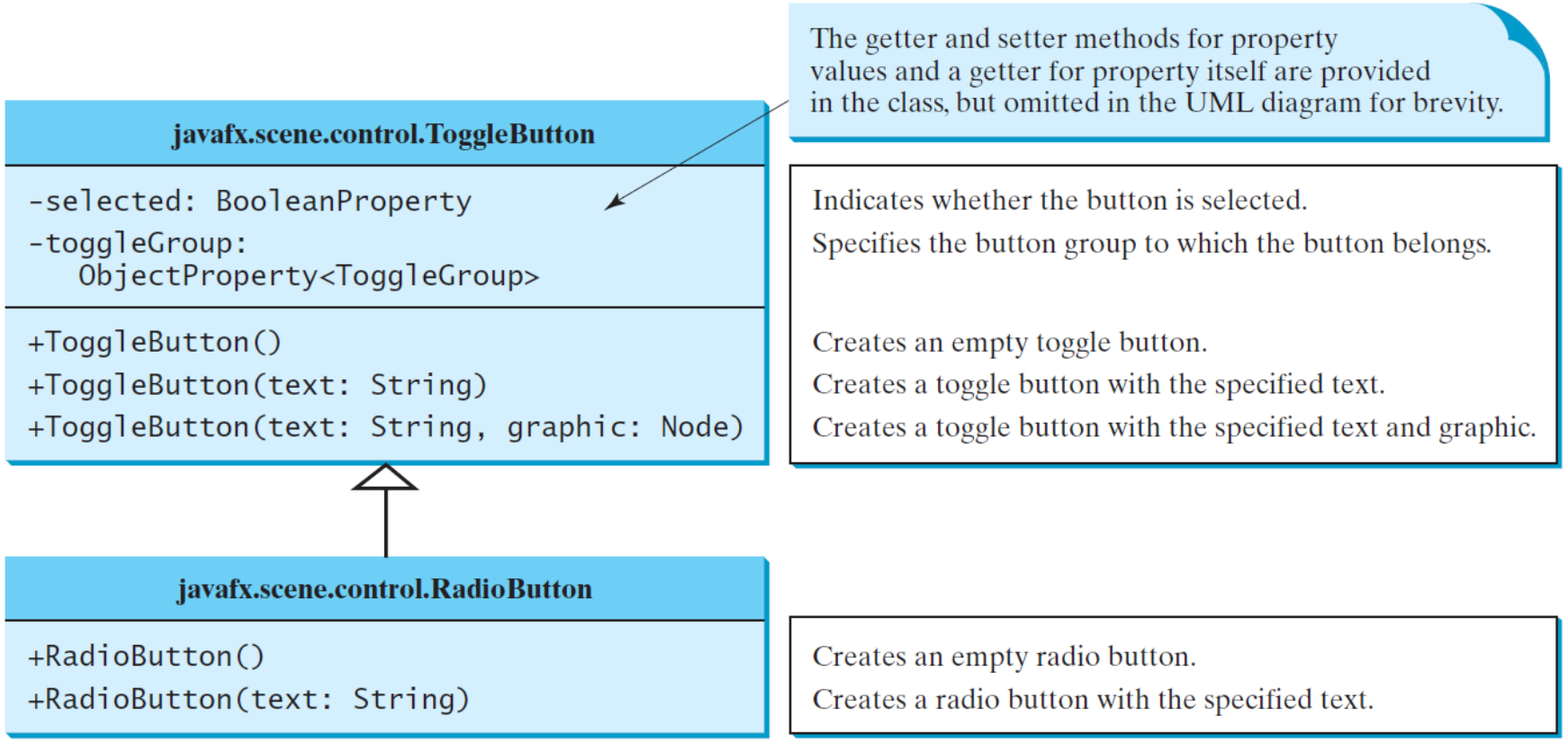
CheckBoxDemo Run

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

    text.setFont(fontNormal);

    VBox paneForCheckBoxes = new VBox(20);
    paneForCheckBoxes.setPadding(new Insets(5, 5, 5, 5));
    paneForCheckBoxes.setStyle("-fx-border-color: green");
    CheckBox chkBold = new CheckBox("Bold");
    CheckBox chkItalic = new CheckBox("Italic");
    paneForCheckBoxes.getChildren().addAll(chkBold, chkItalic);
    pane.setRight(paneForCheckBoxes);
```

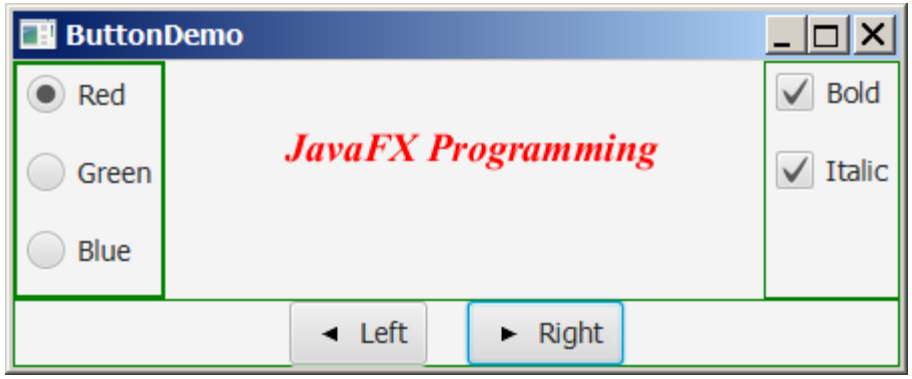
```
    EventHandler<ActionEvent> handler = e -> {
      if (chkBold.isSelected() && chkItalic.isSelected()) {
        text.setFont(fontBoldItalic); // Both check boxes checked
      }
      else if (chkBold.isSelected()) {
        text.setFont(fontBold); // The Bold check box checked
      }
      else if (chkItalic.isSelected()) {
        text.setFont(fontItalic); // The Italic check box checked
      }
      else {
        text.setFont(fontNormal); // Both check boxes unchecked
      }
    };

    chkBold.setOnAction(handler);
    chkItalic.setOnAction(handler);

    return pane; // Return a new pane
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

RadioButton

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



RadioButton Example



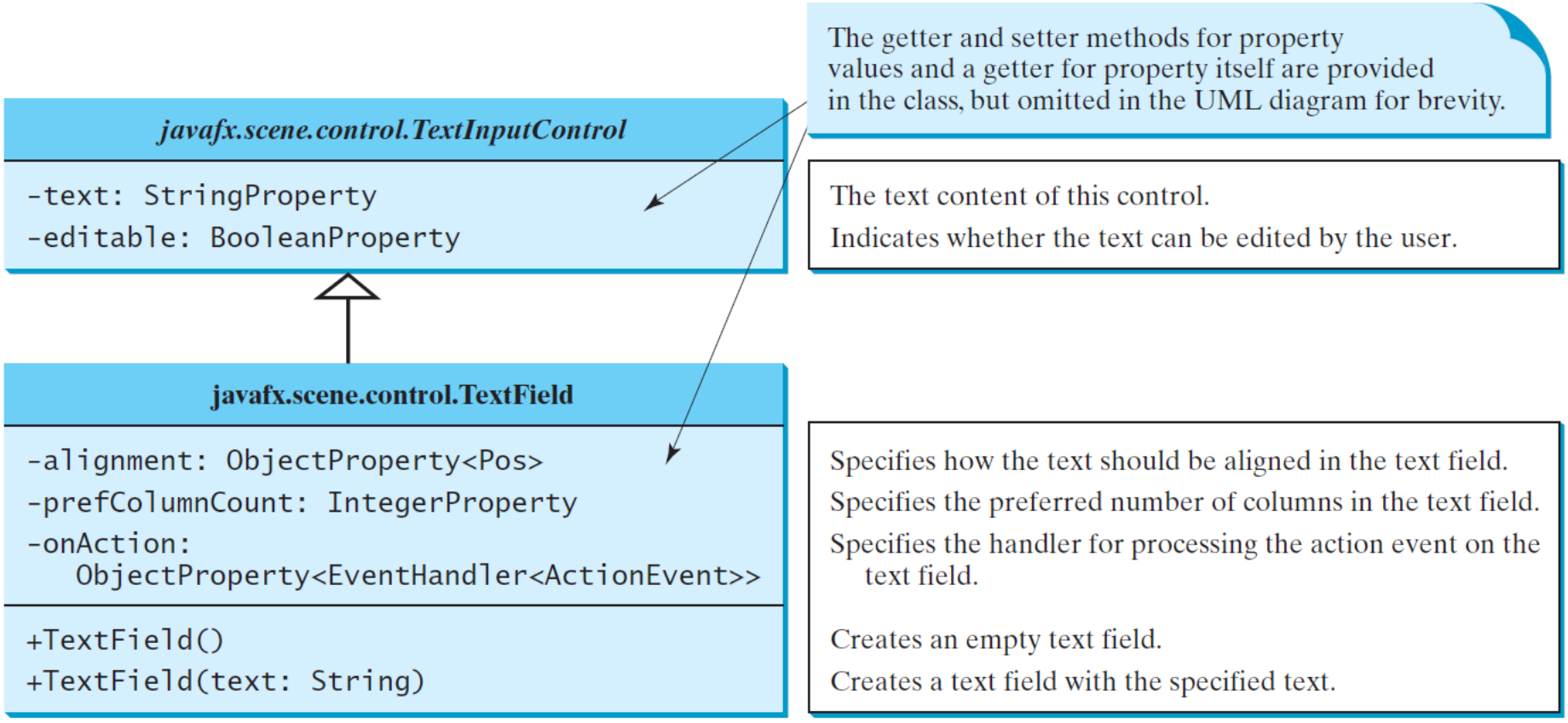
RadioButtonDemo Run

```
import static javafx.application.Application.launch;
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();

    VBox paneForRadioButtons = new VBox(20);
    paneForRadioButtons.setPadding(new Insets(5, 5, 5, 5));
    paneForRadioButtons.setStyle("-fx-border-color: green");
    paneForRadioButtons.setStyle
      ("-fx-border-width: 2px; -fx-border-color: green");
    RadioButton rbRed = new RadioButton("Red");
    RadioButton rbGreen = new RadioButton("Green");
    RadioButton rbBlue = new RadioButton("Blue");
    paneForRadioButtons.getChildren().addAll(rbRed, rbGreen,
rbBlue);
    pane.setLeft(paneForRadioButtons);

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

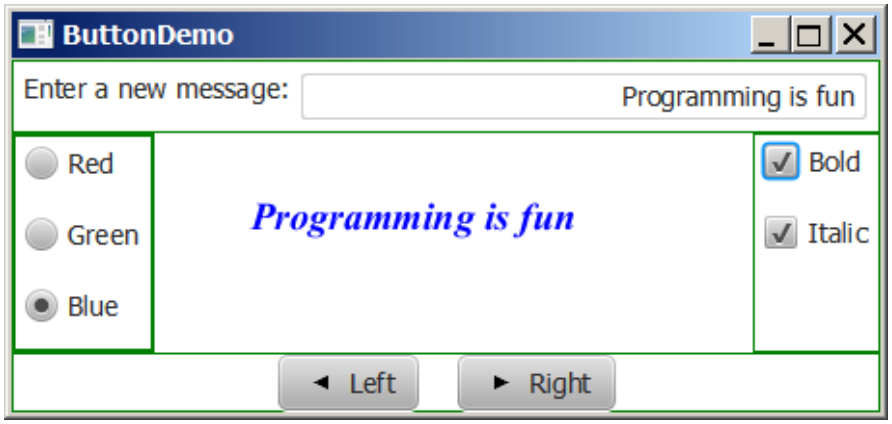
```
    rbRed.setOnAction(e -> {
      if (rbRed.isSelected()) {
        text.setFill(Color.RED);
      }
    });

    rbGreen.setOnAction(e -> {
      if (rbGreen.isSelected()) {
        text.setFill(Color.GREEN);
      }
    });

    rbBlue.setOnAction(e -> {
      if (rbBlue.isSelected()) {
        text.setFill(Color.BLUE);
      }
    });

    return pane;
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

TextField

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



TextField Example

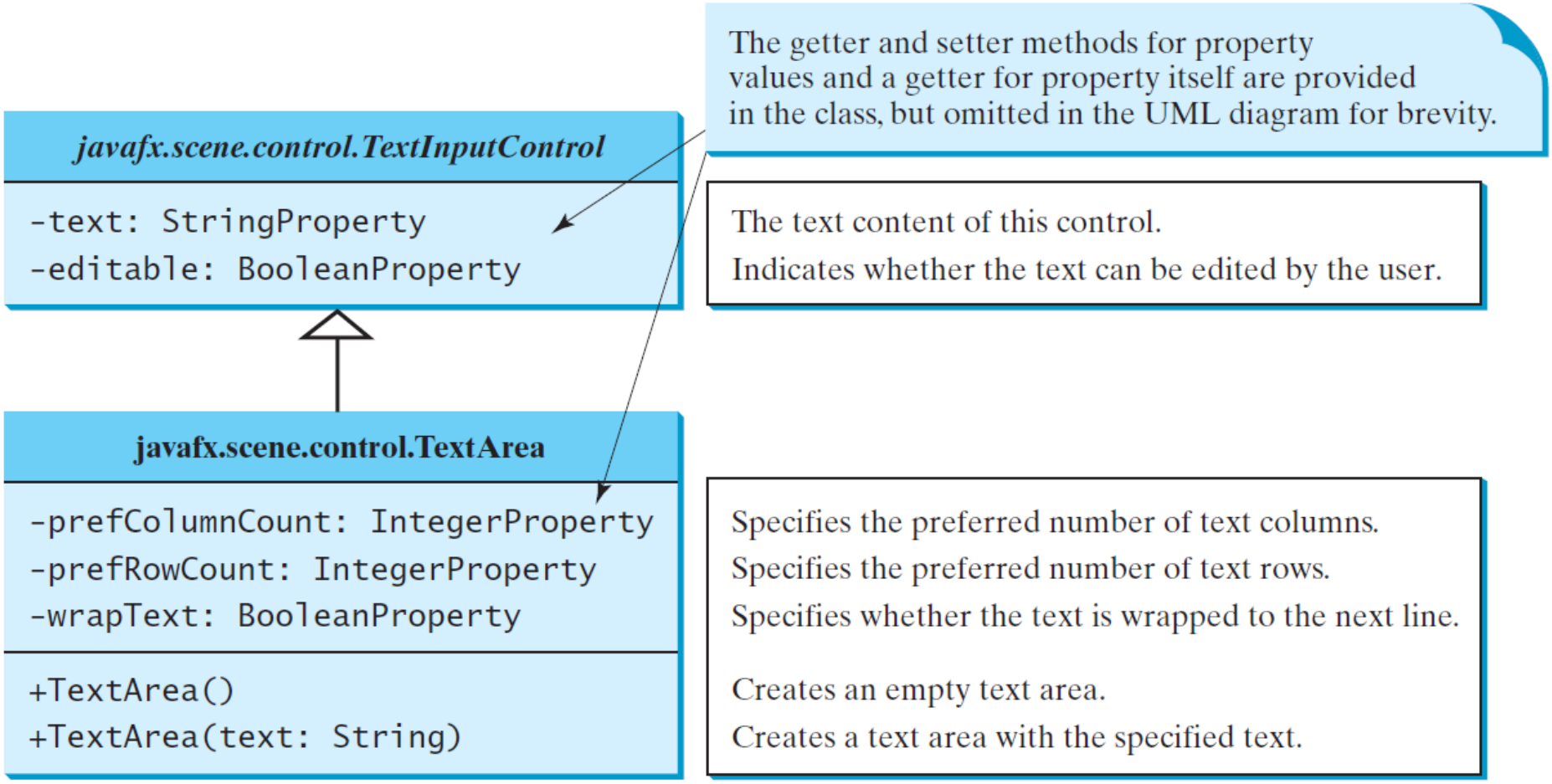


TextFieldDemo Run

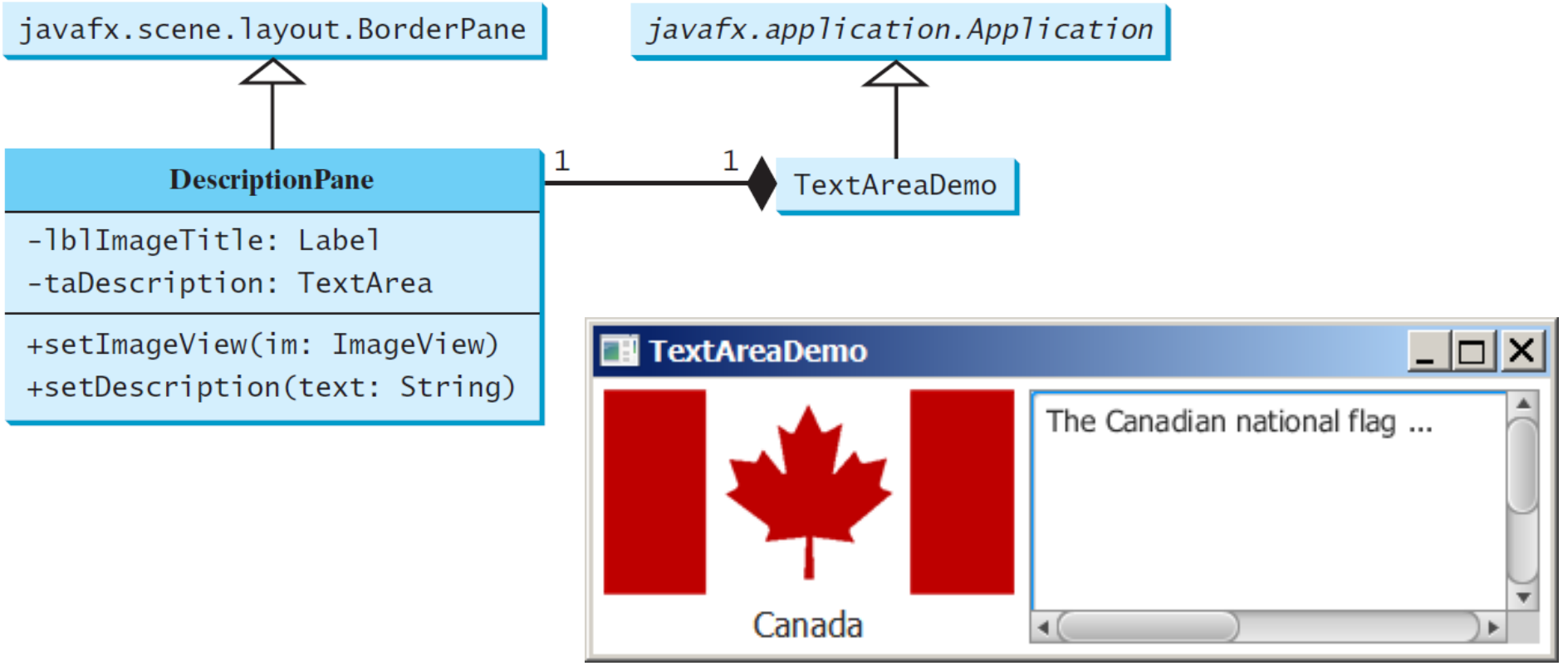
```
import static javafx.application.Application.launch;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;

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

    BorderPane paneForTextField = new BorderPane();
    paneForTextField.setPadding(new Insets(5, 5, 5, 5));
    paneForTextField.setStyle("-fx-border-color: green");
    paneForTextField.setLeft(new Label("Enter a new message: "));

    TextField tf = new TextField();
    tf.setAlignment(Pos.BOTTOM_RIGHT);
    paneForTextField.setCenter(tf);
    pane.setTop(paneForTextField);

    tf.setOnAction(e -> text.setText(tf.getText()));

    return pane;
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

TextArea

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



TextArea Example



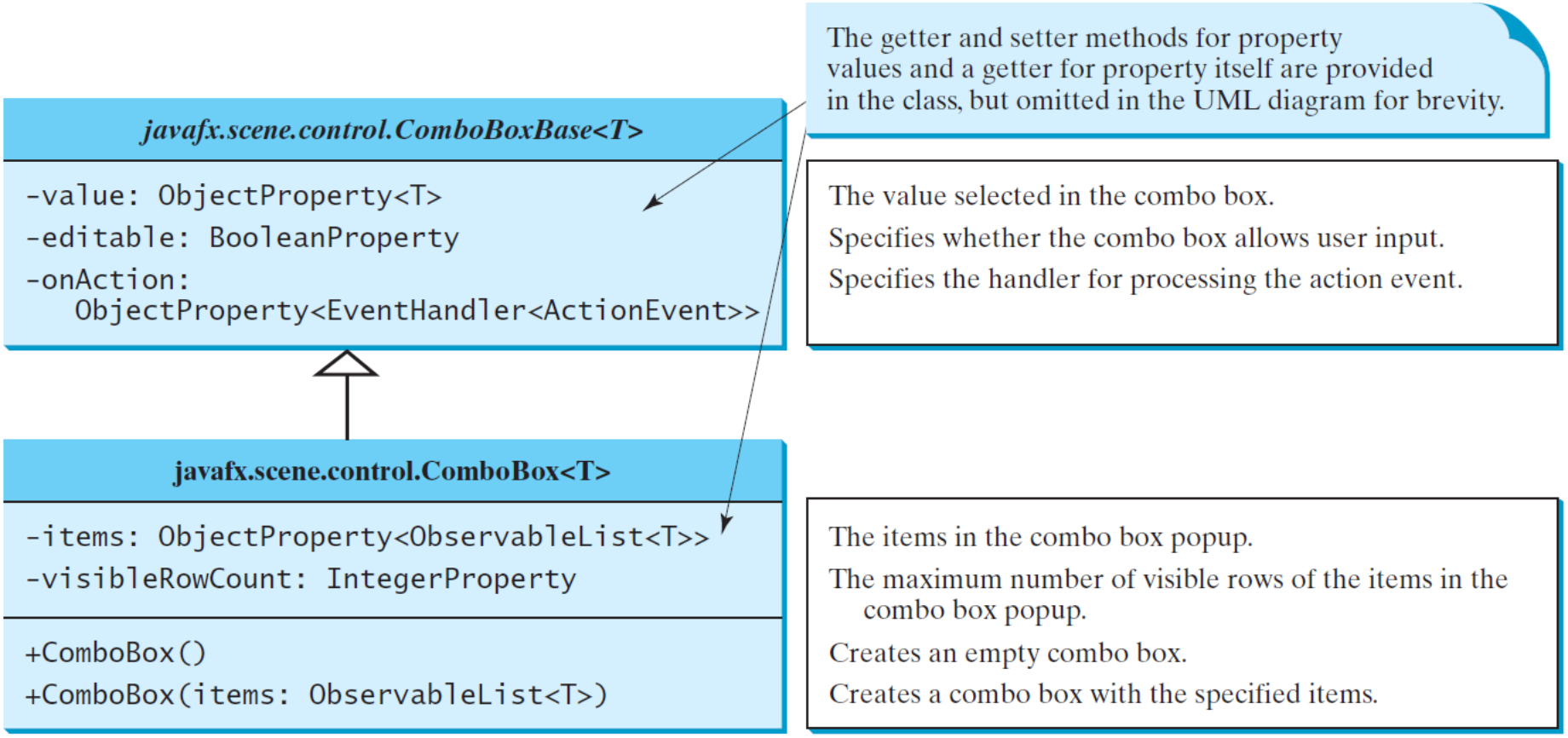
DescriptionPane TextAreaDemo Run

```
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 */
  public void setTitle(String title) {
    lblImageTitle.setText(title);
  }

  /** Set the image view */
  public void setImageView(ImageView icon) {
    lblImageTitle.setGraphic(icon);
  }

  /** Set the text description */
  public void setDescription(String text) {
    taDescription.setText(text);
  }
}
```

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.image.ImageView;

public class TextAreaDemo extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Declare and create a description pane
    DescriptionPane descriptionPane = new DescriptionPane();

    // Set title, text and image in the description pane
    descriptionPane.setTitle("Canada");
    String description = "The Canadian national flag ...";
    descriptionPane.setImageView(new ImageView("image/ca.gif"));
    descriptionPane.setDescription(description);

    // Create a scene and place it in the stage
    Scene scene = new Scene(descriptionPane, 450, 200);
    primaryStage.setTitle("TextAreaDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

ComboBox

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



ComboBox Example

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

ComboBoxDemo Run

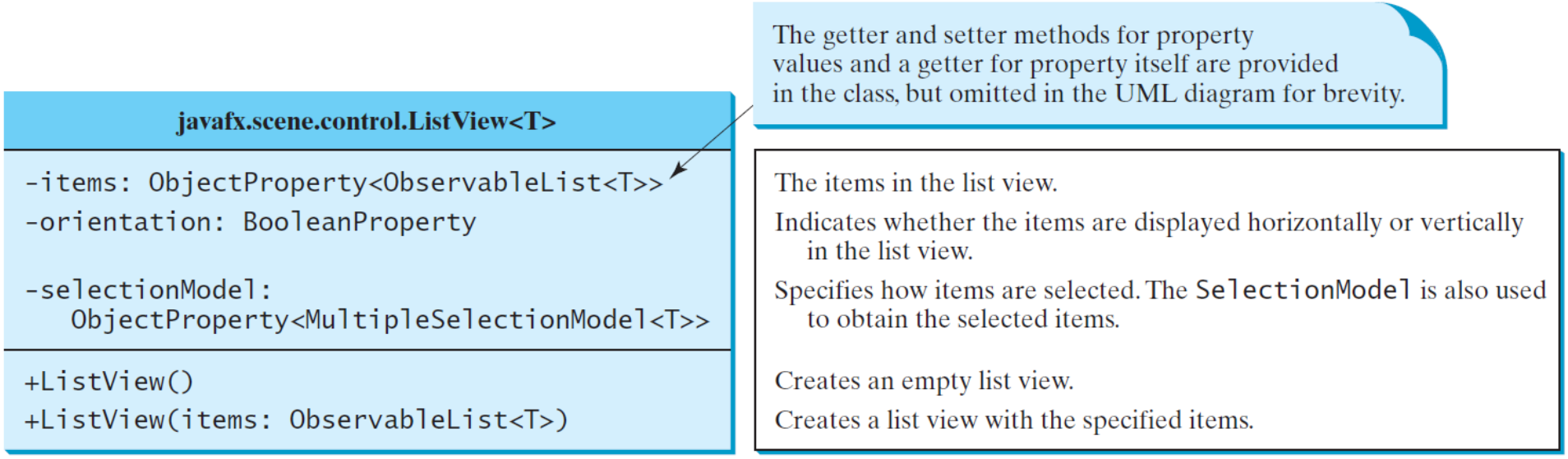
```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;

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
   flagDescription[0] = "The Canadian national flag ...";
   flagDescription[1] = "Description for China ... ";
   flagDescription[2] = "Description for Denmark ... ";
   flagDescription[3] = "Description for France ... ";
   flagDescription[4] = "Description for Germany ... ";
   flagDescription[5] = "Description for India ... ";
   flagDescription[6] = "Description for Norway ... ";
   flagDescription[7] = "Description for UK ... ";
   flagDescription[8] = "Description for US ... ";

   // Set the first country (Canada) for display
   setDisplay(0);

   // Add combo box and description pane to the border pane
   BorderPane pane = new BorderPane();

   BorderPane paneForComboBox = new BorderPane();
   paneForComboBox.setLeft(new Label("Select a country: "));
   paneForComboBox.setCenter(cbo);
   pane.setTop(paneForComboBox);
   cbo.setPrefWidth(400);
   cbo.setValue("Canada");

   ObservableList<String> items =
     FXCollections.observableArrayList(flagTitles);
   cbo.getItems().addAll(items);
   pane.setCenter(descriptionPane);
```

```
// Display the selected country
  cbo.setOnAction(e -> setDisplay(items.indexOf(cbo.getValue())));

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

 /** Set display information on the description pane */
 public void setDisplay(int index) {
  descriptionPane.setTitle(flagTitles[index]);
  descriptionPane.setImageView(flagImage[index]);
  descriptionPane.setDescription(flagDescription[index]);
 }

 /**
  * The main method is only needed for the IDE with limited
  * JavaFX support. Not needed for running from the command line.
  */
 public static void main(String[] args) {
  launch(args);
 }
}
```

ListView

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



Example: Using `ListView`

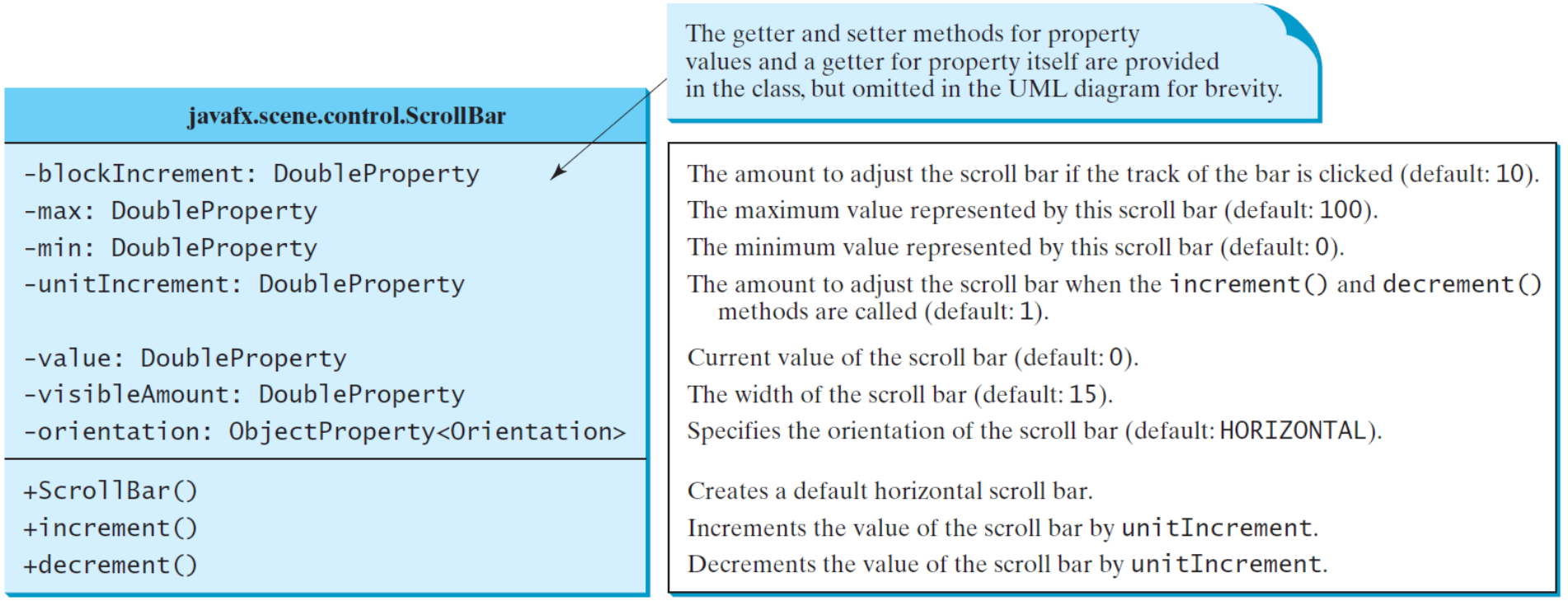
This example gives a program that lets users select countries in a list and display the flags of the selected countries in the labels.

ListViewDemo Run

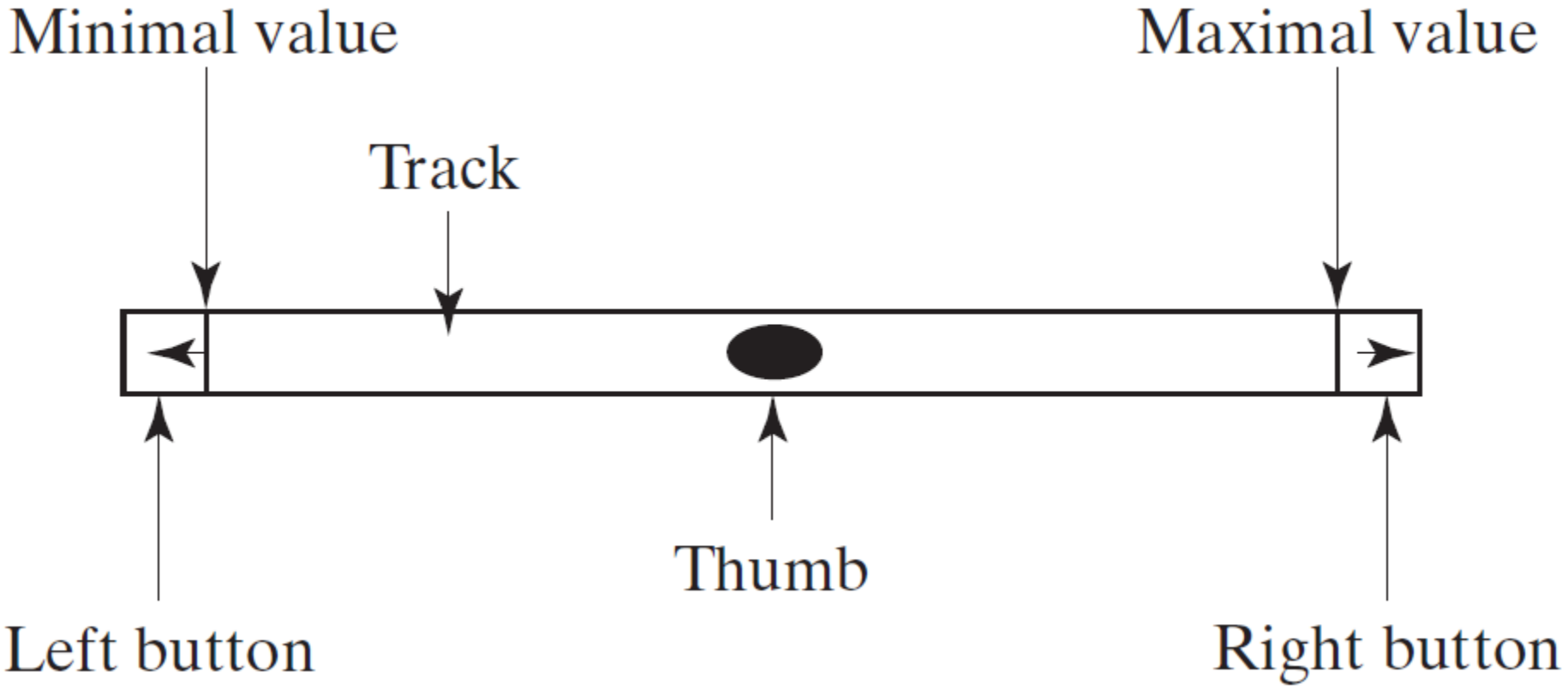
```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.collections.FXCollections;
import javafx.scene.Scene;
import javafx.scene.control.ListView;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.SelectionMode;
import javafx.scene.image.ImageView;
import 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
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

ScrollBar

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

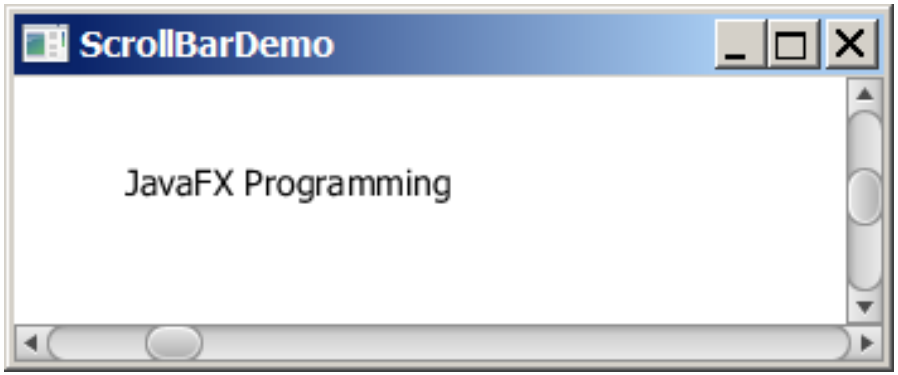


Scroll Bar Properties



Example: Using Scrollbars

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

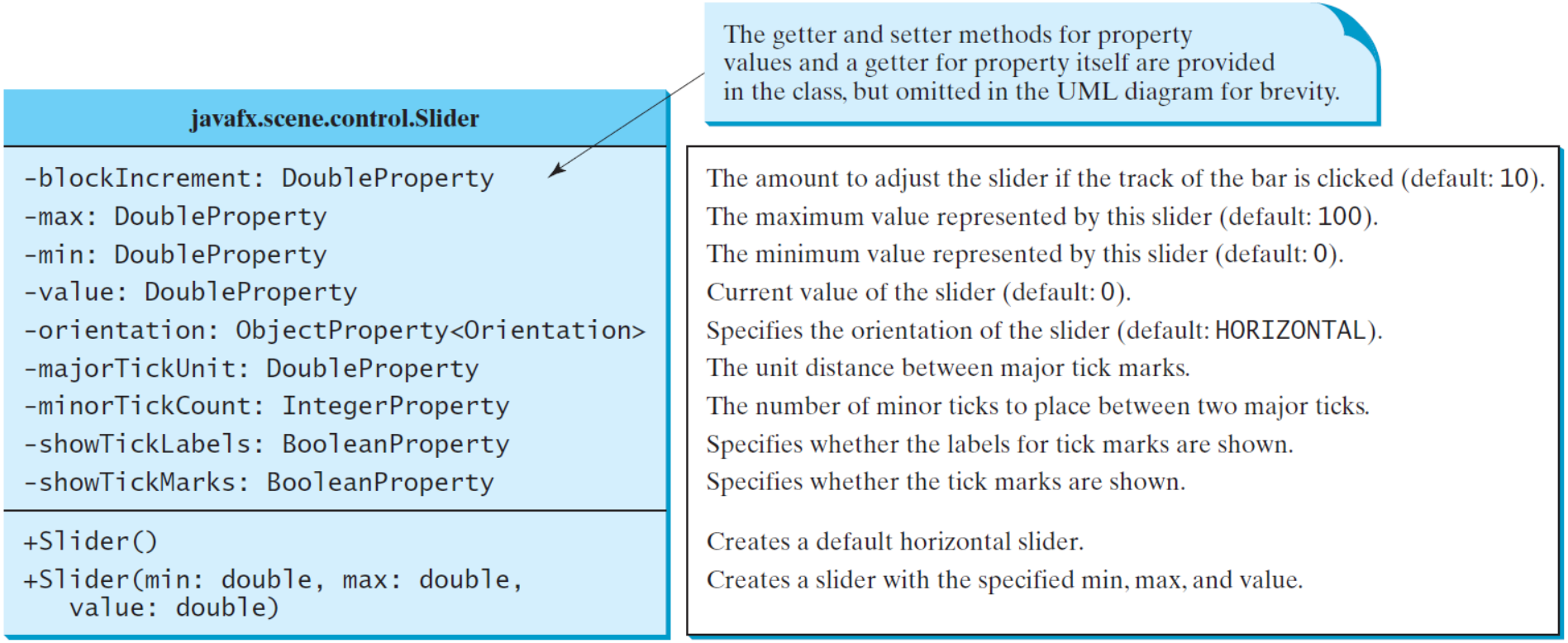


ScrollBarDemo Run

```
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
    Pane paneForText = new Pane();
    paneForText.getChildren().add(text);

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

    // Listener for horizontal scroll bar value change
    sbHorizontal.valueProperty().addListener(ov ->
      text.setX(sbHorizontal.getValue() * paneForText.getWidth() /
        sbHorizontal.getMax()));

    // Listener for vertical scroll bar value change
    sbVertical.valueProperty().addListener(ov ->
      text.setY(sbVertical.getValue() * paneForText.getHeight() /
        sbVertical.getMax()));

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

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

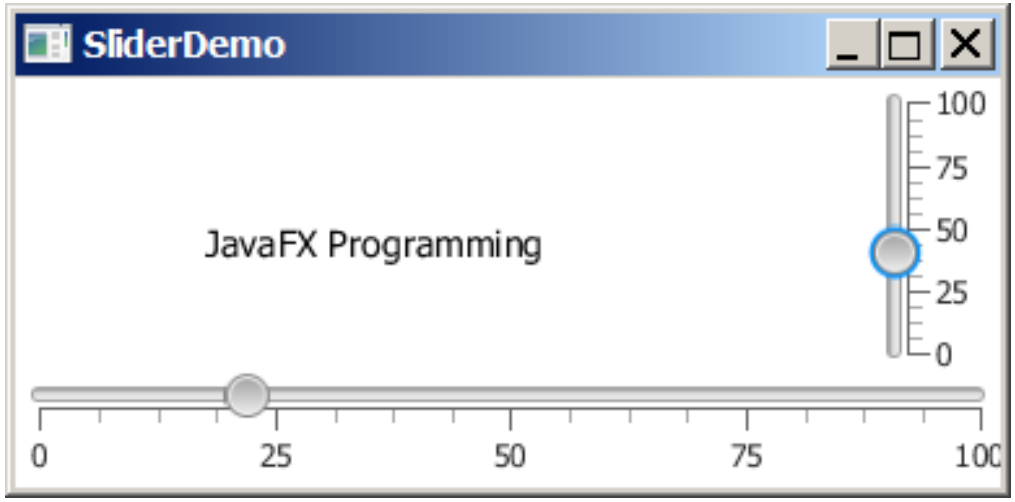
Slider

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



Example: Using Sliders

Rewrite the preceding program using the sliders to control a message displayed on a panel instead of using scroll bars.

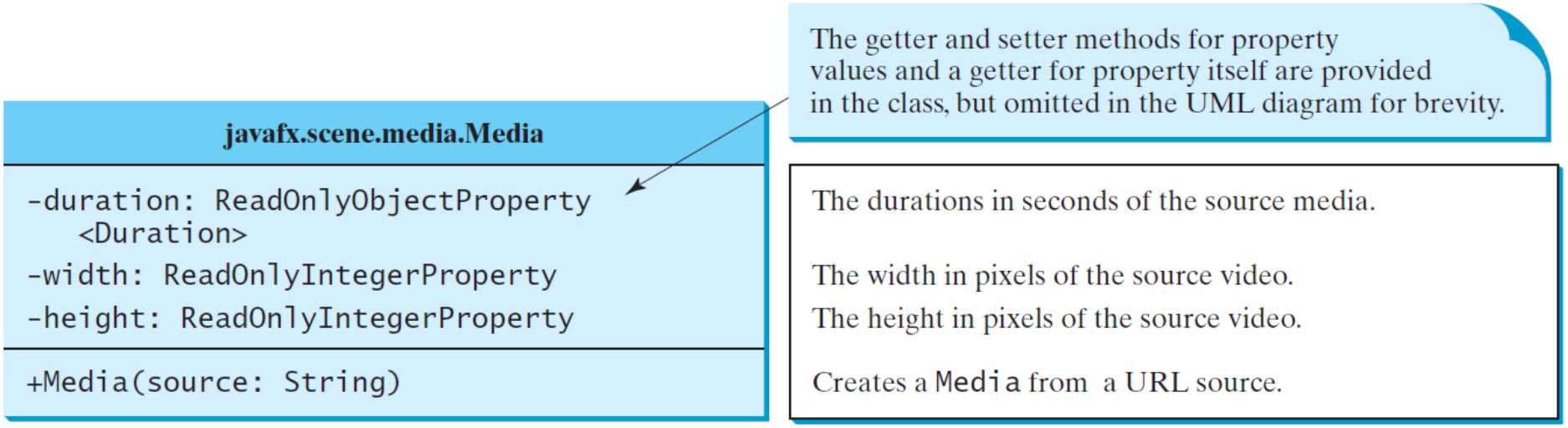


SliderDemo Run

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

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

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

    Slider slVertical = new Slider();
    slVertical.setOrientation(Orientation.VERTICAL);
    slVertical.setShowTickLabels(true);
    slVertical.setShowTickMarks(true);
    slVertical.setValue(100);

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

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

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

    slVertical.valueProperty().addListener(ov ->
      text.setY((slVertical.getMax() - slVertical.getValue())
        * paneForText.getHeight() / slVertical.getMax()));

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

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

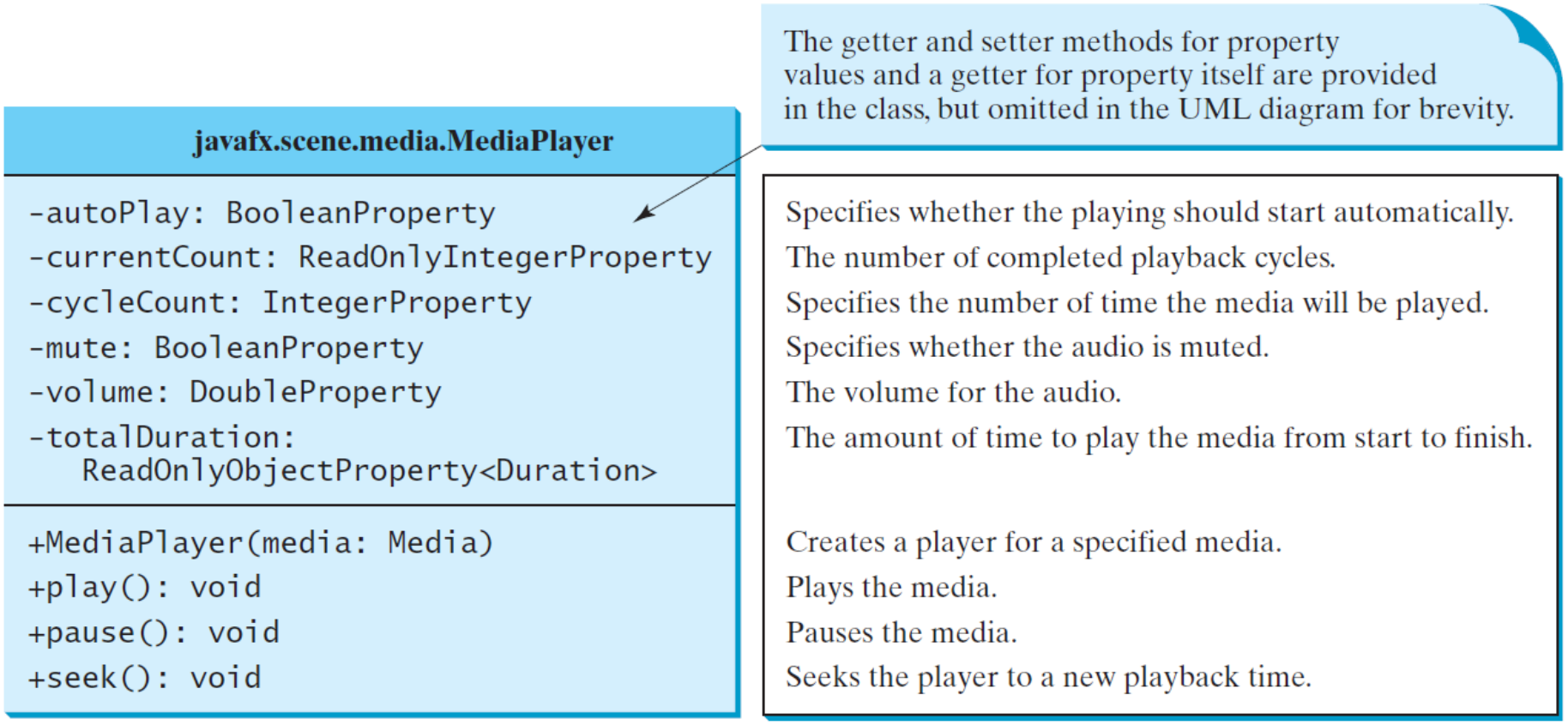
Media

You can use the **Media** class to obtain the source of the media, the **MediaPlayer** class to play and control the media, and the **MediaView** class to display the video.



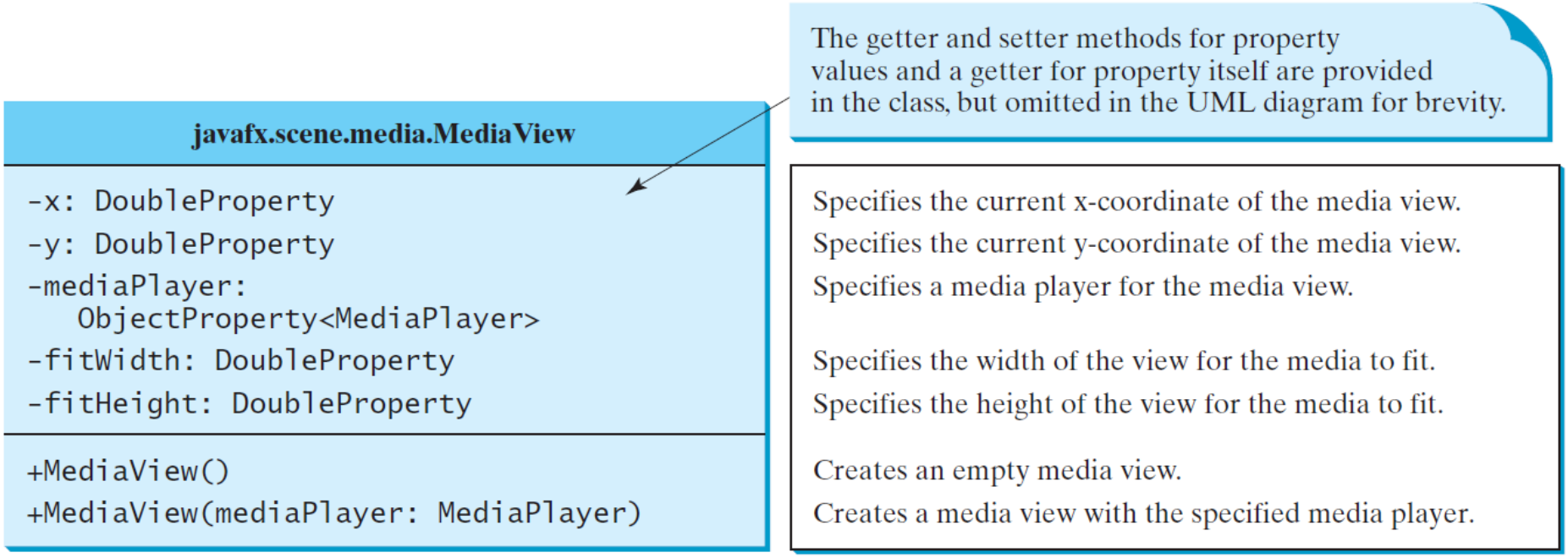
MediaPlayer

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



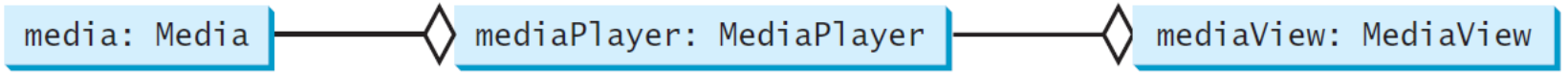
MediaView

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



Example: Using Media

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

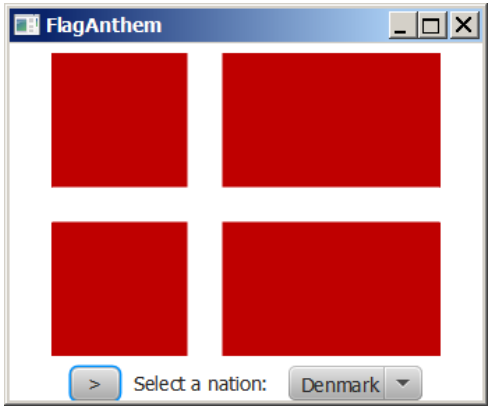


MediaDemo Run

```
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;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import javafx.util.Duration;

public class MediaDemo extends Application {
  private static final String MEDIA_URL =
    "https://liveexample.pearsoncmg.com/common/sample.mp4";

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

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

    Slider slVolume = new Slider();
    slVolume.setPrefWidth(150);
    slVolume.setMaxWidth(Region.USE_PREF_SIZE);
    slVolume.setMinWidth(30);
    slVolume.setValue(50);
    mediaPlayer.volumeProperty().bind(
      slVolume.valueProperty().divide(100));

    HBox hBox = new HBox(10);
    hBox.setAlignment(Pos.CENTER);
    hBox.getChildren().addAll(playButton, rewindButton,
      new Label("Volume"), slVolume);

    BorderPane pane = new BorderPane();
    pane.setCenter(mediaView);
    pane.setBottom(hBox);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 650, 500);
    primaryStage.setTitle("MediaDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

Case Study: National Flags and Anthems

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



```
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].pause();
      }
      else {
        btPlayPause.setText(">");
        mp[currentIndex].play();
      }
    });

    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();
    });

    HBox hBox = new HBox(10);
    hBox.getChildren().addAll(btPlayPause,
      new Label("Select a nation: "), cboNation);
    hBox.setAlignment(Pos.CENTER);
```

```
    // Create a pane to hold nodes
    BorderPane pane = new BorderPane();
    pane.setCenter(imageView);
    pane.setBottom(hBox);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 350, 270);
    primaryStage.setTitle("FlagAnthem"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

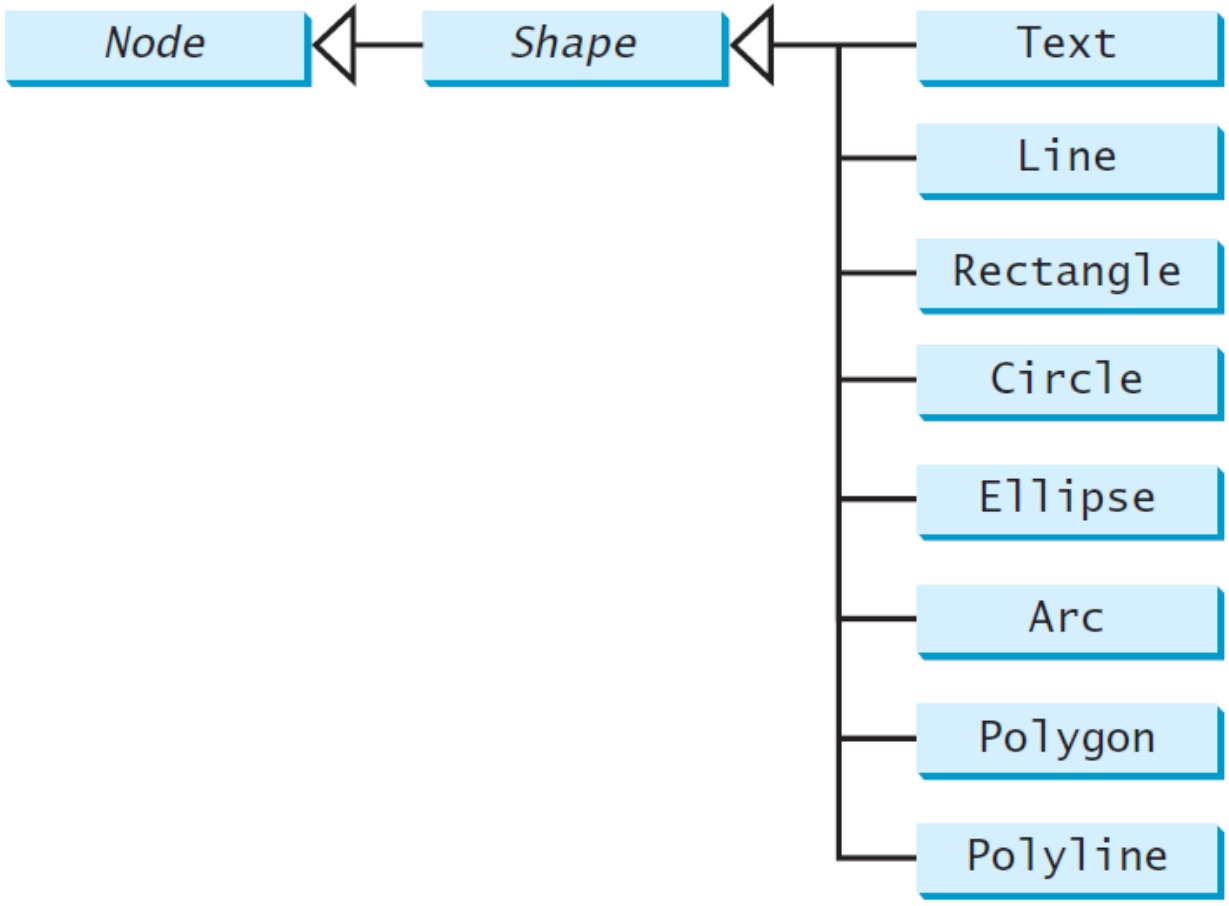
Drawing with Shapes and Text
Property Binding
Animation

Objectives

- To use binding properties to synchronize property values (§14.5).
- To display text using the **Text** class and create shapes using **Line**, **Circle**, **Rectangle**, **Ellipse**, **Arc**, **Polygon**, and **Polyline** (§14.11).
- To develop the reusable GUI components **ClockPane** for displaying an analog clock (§14.12).

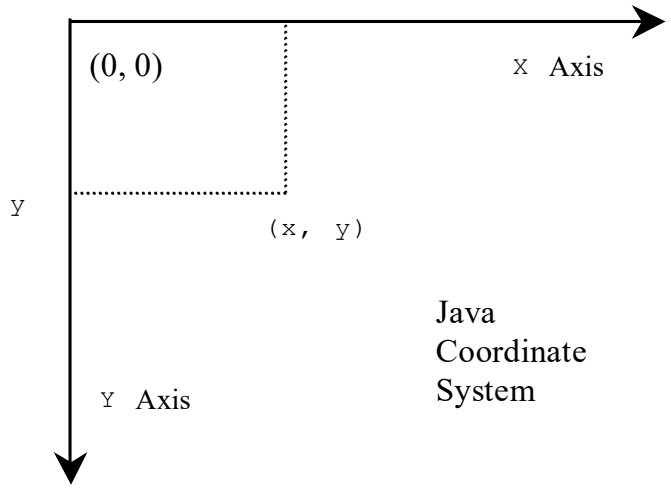
Shapes

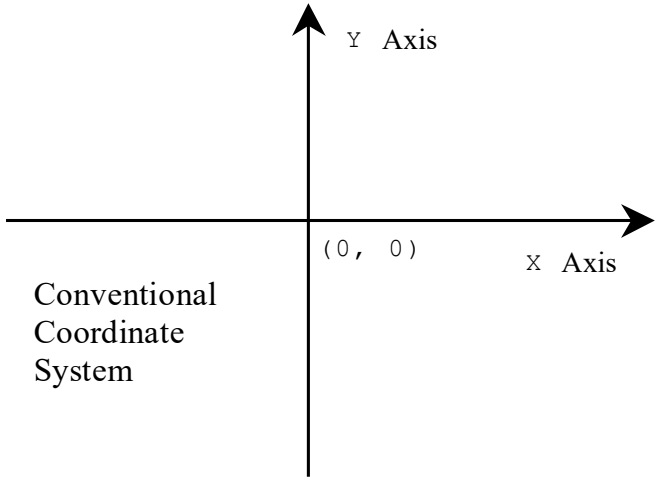
JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.



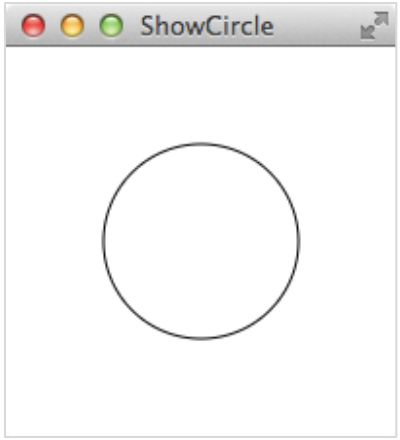
Display a Shape

This example displays a circle in the center of the pane.





```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

public class ShowCircle extends Application {
  @Override // Override the start method in the Application
class
  public void start(Stage primaryStage) {
    // Create a circle and set its properties
    Circle circle = new Circle();
    circle.setCenterX(100);
    circle.setCenterY(100);
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(null);
```




```
    // Create a pane to hold the circle
    Pane pane = new Pane();
    pane.getChildren().add(circle);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 200, 200);
    primaryStage.setTitle("ShowCircle"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

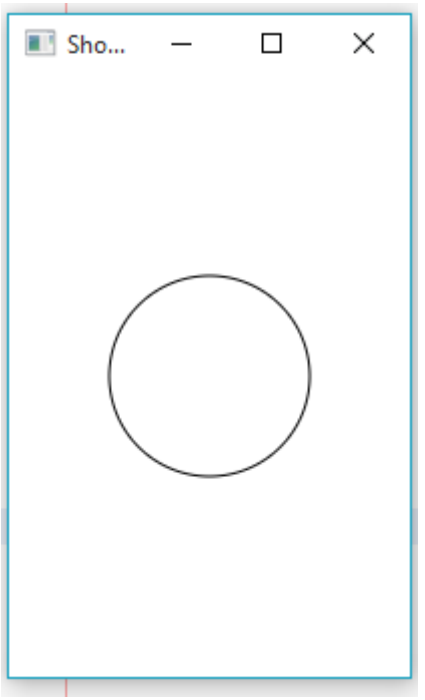
Binding Properties

- JavaFX introduces a new concept called *binding property* that enables a *target object* to be bound to a *source object*.
- If the value in the source object changes, the target property is also changed automatically.
- The target object is simply called a *binding object* or a *binding property*.

ShowCircleCentered Run

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

public class ShowCircleCentered extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane to hold the circle
    Pane pane = new Pane();

    // Create a circle and set its properties
    Circle circle = new Circle();
    circle.centerXProperty().bind(pane.widthProperty().divide(2));
    circle.centerYProperty().bind(pane.heightProperty().divide(2));
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(Color.WHITE);
    pane.getChildren().add(circle); // Add circle to the pane
```

```
// Create a scene and place it in the stage
  Scene scene = new Scene(pane, 200, 200);
  primaryStage.setTitle("ShowCircleCentered"); // Set the stage title
  primaryStage.setScene(scene); // Place the scene in the stage
  primaryStage.show(); // Display the stage
 }

 /**
  * The main method is only needed for the IDE with limited
  * JavaFX support. Not needed for running from the command line.
  */
 public static void main(String[] args) {
  launch(args);
 }
}
```



Binding Property: getter, setter, and property getter

```
public class SomeClassName {

  private PropertyType x;

  /** Value getter method */
  public propertyValueType getX() { ... }

  /** Value setter method */
  public void setX(propertyValueType value) { ... }

  /** Property getter method */
  public PropertyType
    xProperty() { ... }
}
```

(a) x is a binding property

```
public class Circle {

  private DoubleProperty centerX;

  /** Value getter method */
  public double getCenterX() { ... }

  /** Value setter method */
  public void setCenterX(double value) { ... }

  /** Property getter method */
  public DoubleProperty centerXProperty() { ... }
}
```

(b) centerX is binding property

Uni/Bidirectional Binding

```
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;

public class BidirectionalBindingDemo {
  public static void main(String[] args) {
    DoubleProperty d1 = new SimpleDoubleProperty(1);
    DoubleProperty d2 = new SimpleDoubleProperty(2);
    d1.bindBidirectional(d2);
    System.out.println("d1 is " + d1.getValue()
      + " and d2 is " + d2.getValue());
    d1.setValue(50.1);
    System.out.println("d1 is " + d1.getValue()
      + " and d2 is " + d2.getValue());
    d2.setValue(70.2);
    System.out.println("d1 is " + d1.getValue()
      + " and d2 is " + d2.getValue());
  }
}
```

```
d1 is 2.0 and d2 is 2.0
d1 is 50.1 and d2 is 50.1
d1 is 70.2 and d2 is 70.2
```

BidirectionalBindingDemo

Run

Uni/Bidirectional Binding, cont'

```
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;

public class BindingDemo {
  public static void main(String[] args) {
    DoubleProperty d1 = new SimpleDoubleProperty(1);
    DoubleProperty d2 = new SimpleDoubleProperty(2);
    d1.bind(d2); // Bind d1 with d2
    System.out.println("d1 is " + d1.getValue()
      + " and d2 is " + d2.getValue());
    d2.setValue(70.2);
    System.out.println("d1 is " + d1.getValue()
      + " and d2 is " + d2.getValue());
  }
}
```

```
d1 is 2.0 and d2 is 2.0
d1 is 70.2 and d2 is 70.2
```

BindingDemo

Run

Listeners for Observable Objects

You can add a listener to process a value change in an observable object.

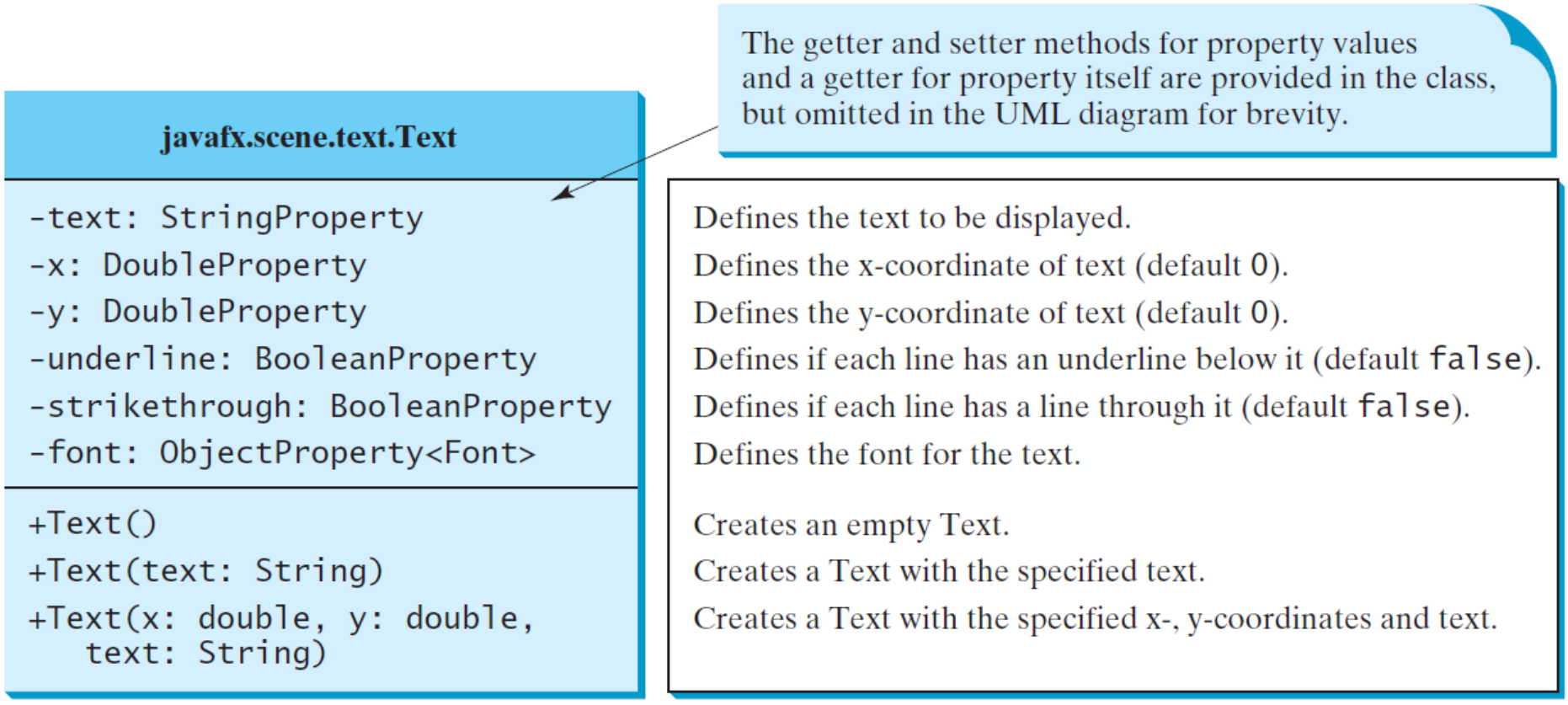
An instance of **Observable** is known as an *observable object*, which contains the **addListener(InvalidationListener listener)** method for adding a listener. Once the value is changed in the property, a listener is notified. The listener class should implement the **InvalidationListener** interface, which uses the **invalidated(Observable o)** method to handle the property value change. Every binding property is an instance of **Observable**.

ObservablePropertyDemo Run

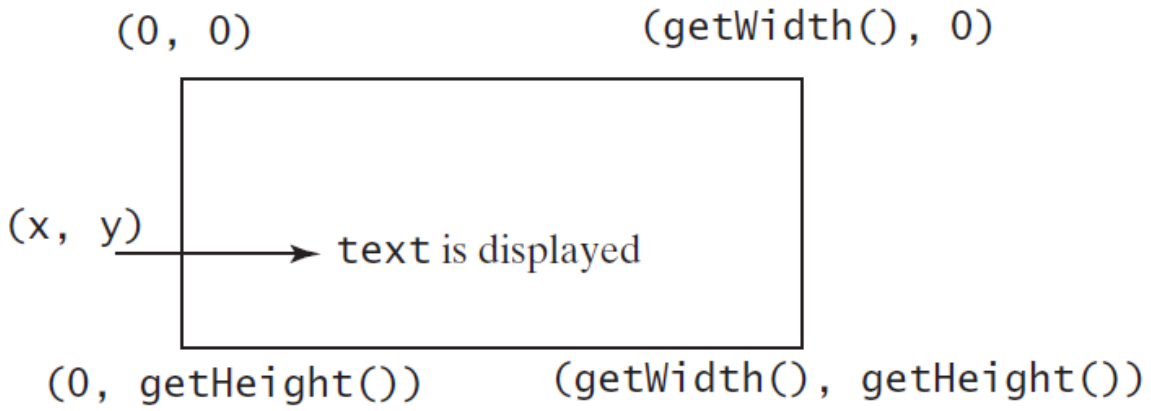
```
import javafx.beans.InvalidationListener;
import javafx.beans.Observable;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;

public class ObservablePropertyDemo {
  public static void main(String[] args) {
    DoubleProperty balance = new SimpleDoubleProperty();
    balance.addListener(new InvalidationListener() {
      public void invalidated(Observable ov) {
        System.out.println("The new value is " +
          balance.doubleValue());
      }
    });

    balance.set(4.5);
  }
}
```

```
The new value is 4.5
```

Text


javafx.scene.text.Text
-text: StringProperty
-x: DoubleProperty
-y: DoubleProperty
-underline: BooleanProperty
-strikethrough: BooleanProperty
-font: ObjectProperty<Font>
+Text()
+Text(text: String)
+Text(x: double, y: double, 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 the text to be displayed.
Defines the x-coordinate of text (default 0).
Defines the y-coordinate of text (default 0).
Defines if each line has an underline below it (default false).
Defines if each line has a line through it (default false).
Defines the font for the text.
Creates an empty Text.
Creates a Text with the specified text.
Creates a Text with the specified x-, y-coordinates and text.

Text Example



(a) Text(x, y, text)

(b) *Three* Text *objects are displayed*

ShowText Run

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.geometry.Insets;
import javafx.stage.Stage;
import javafx.scene.text.Text;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.FontPosture;

public class ShowText extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane to hold the texts
    Pane pane = new Pane();
    pane.setPadding(new Insets(5, 5, 5, 5));
    Text text1 = new Text(20, 20, "Programming is fun");
    text1.setFont(Font.font("Courier", FontWeight.BOLD,
      FontPosture.ITALIC, 15));
    pane.getChildren().add(text1);

    Text text2 = new Text(60, 60, "Programming is fun\nDisplay text");
    pane.getChildren().add(text2);

    Text text3 = new Text(10, 100, "Programming is fun\nDisplay text");
    text3.setFill(Color.RED);
    text3.setUnderline(true);
    text3.setStrikethrough(true);
    pane.getChildren().add(text3);

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

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

Line

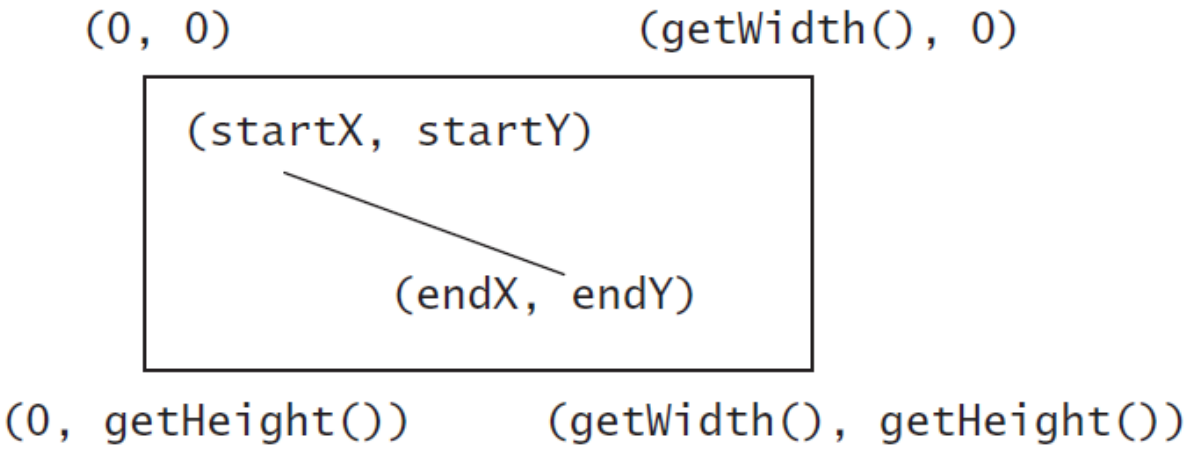
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.shape.Line

```
-startX: DoubleProperty
-startY: DoubleProperty
-endX: DoubleProperty
-endY: DoubleProperty
+Line()
+Line(startX: double, startY:
   double, endX: double, endY:
   double)
```

The x-coordinate of the start point.
The y-coordinate of the start point.
The x-coordinate of the end point.
The y-coordinate of the end point.

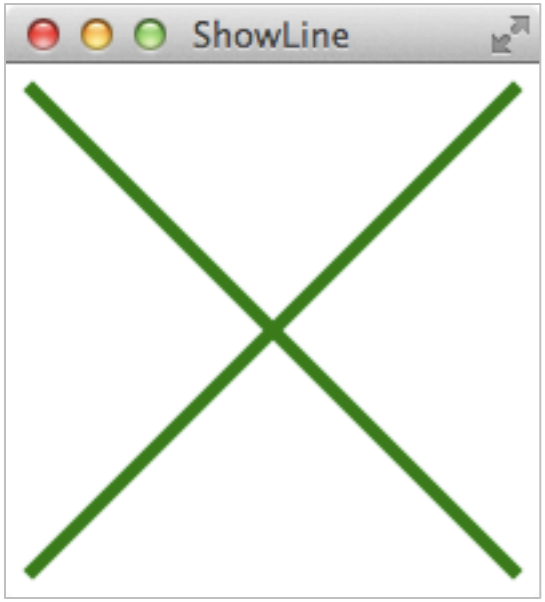
Creates an empty `Line`.
Creates a `Line` with the specified starting and ending points.



ShowLine Run

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.shape.Line;

public class ShowLine extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a scene and place it in the stage
    Scene scene = new Scene(new LinePane(), 200, 200);
    primaryStage.setTitle("ShowLine"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
  public static void main(String[] args) {
    launch(args);
  }
}
```

```
class LinePane extends Pane {
  public LinePane() {
    Line line1 = new Line(10, 10, 10, 10);
    line1.endXProperty().bind(widthProperty().subtract(10));
    line1.endYProperty().bind(heightProperty().subtract(10));
    line1.setStrokeWidth(5);
    line1.setStroke(Color.GREEN);
    getChildren().add(line1);

    Line line2 = new Line(10, 10, 10, 10);
    line2.startXProperty().bind(widthProperty().subtract(10));
    line2.endYProperty().bind(heightProperty().subtract(10));
    line2.setStrokeWidth(5);
    line2.setStroke(Color.GREEN);
    getChildren().add(line2);
  }
}
```



Rectangle

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.shape.Rectangle

```
-x: DoubleProperty
-y:DoubleProperty
-width: DoubleProperty
-height: DoubleProperty
-arcWidth: DoubleProperty

-arcHeight: DoubleProperty
```

```
+Rectangle()
+Rectanlge(x: double, y:
   double, width: double,
   height: double)
```

The x-coordinate of the upper-left corner of the rectangle (default 0).

The y-coordinate of the upper-left corner of the rectangle (default 0).

The width of the rectangle (default: 0).

The height of the rectangle (default: 0).

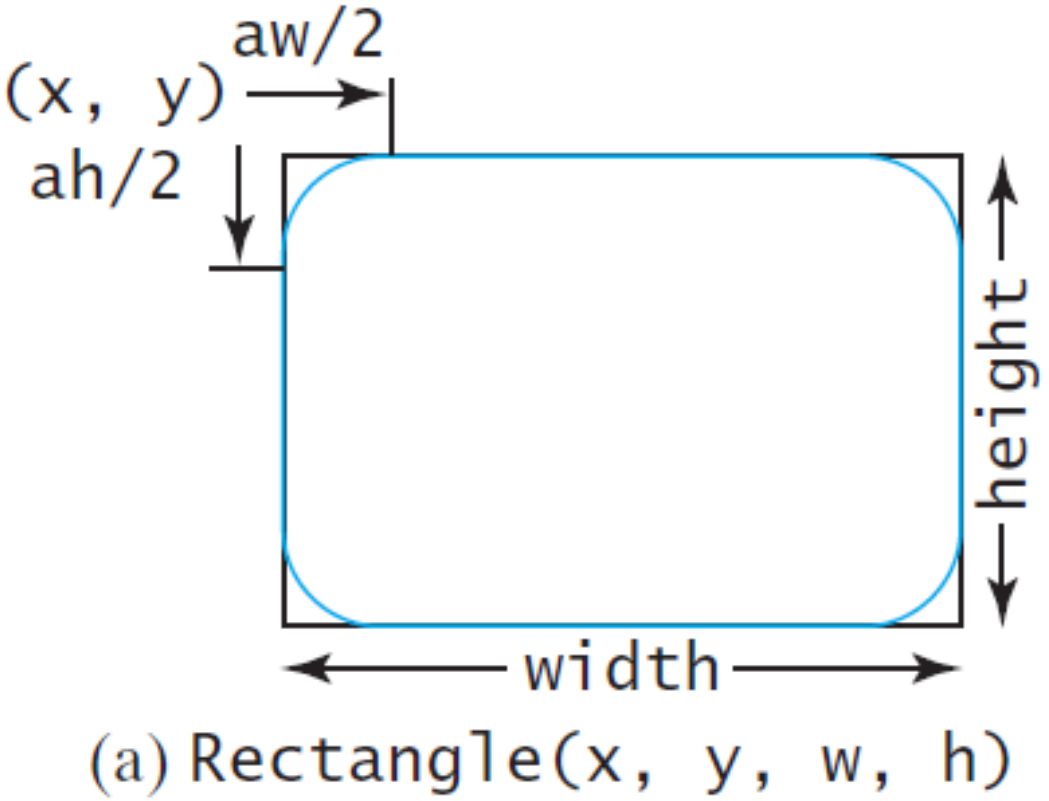
The `arcWidth` of the rectangle (default: 0). `arcWidth` is the horizontal diameter of the arcs at the corner (see Figure 14.31a).

The `arcHeight` of the rectangle (default: 0). `arcHeight` is the vertical diameter of the arcs at the corner (see Figure 14.31a).

Creates an empty `Rectangle`.

Creates a `Rectangle` with the specified upper-left corner point, width, and height.

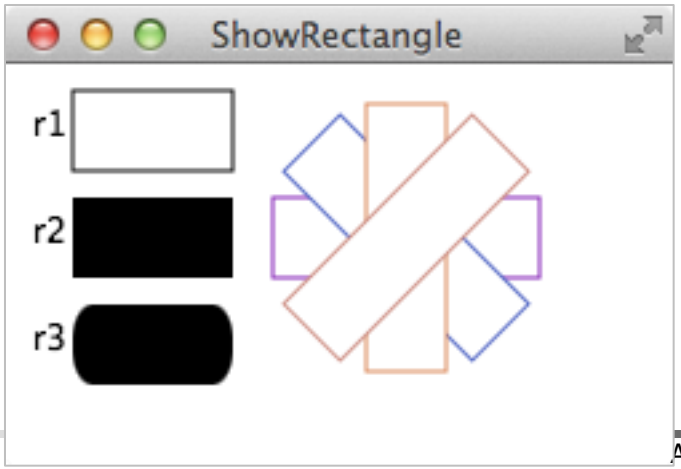
Rectangle Example



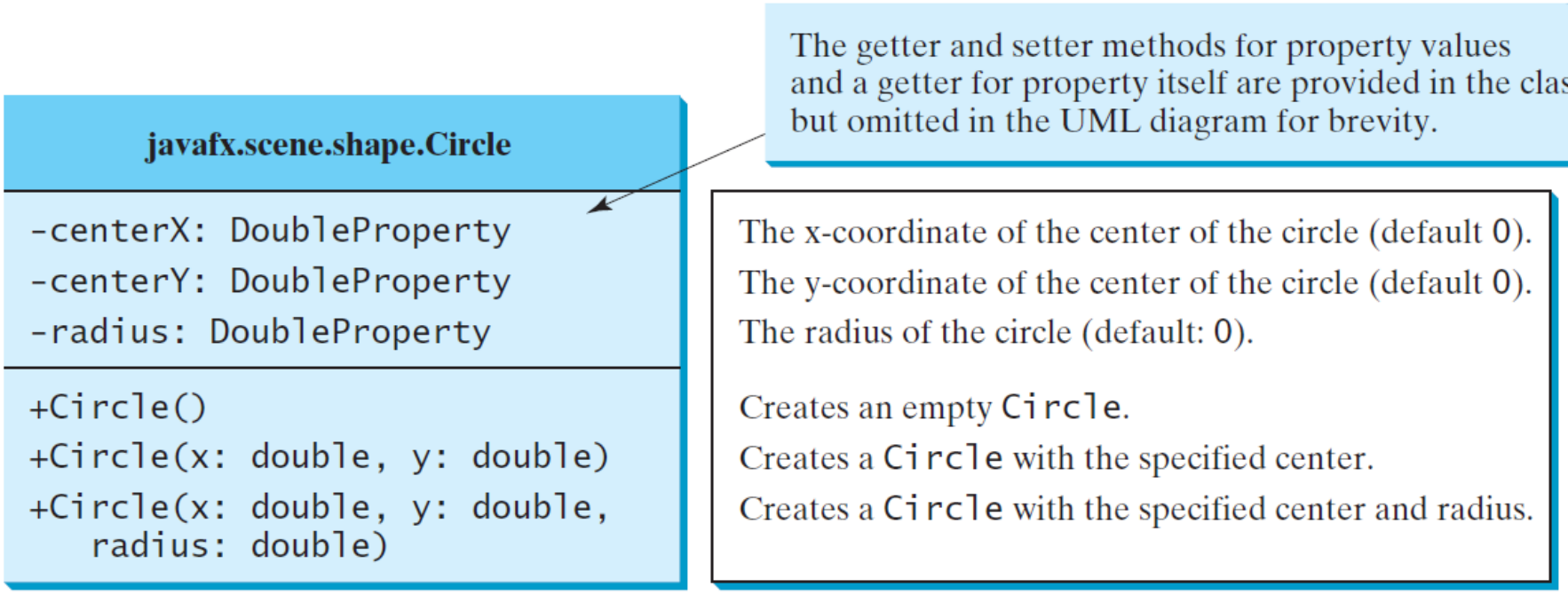
(a) Rectangle(x, y, w, h)

ShowRectangle Run

```
import java.util.Collections;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.text.Text;
import javafx.scene.shape.Rectangle;

public class ShowRectangle extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane
    Pane pane = new Pane();

    // Create rectangles and add to pane
    Rectangle r1 = new Rectangle(25, 10, 60, 30);
    r1.setStroke(Color.BLACK);
    r1.setFill(Color.WHITE);
    pane.getChildren().add(new Text(10, 27, "r1"));
    pane.getChildren().add(r1);

    Rectangle r2 = new Rectangle(25, 50, 60, 30);
    pane.getChildren().add(new Text(10, 67, "r2"));
    pane.getChildren().add(r2);

    Rectangle r3 = new Rectangle(25, 90, 60, 30);
    r3.setArcWidth(15);
    r3.setArcHeight(25);
    pane.getChildren().add(new Text(10, 107, "r3"));
    pane.getChildren().add(r3);

    for (int i = 0; i < 4; i++) {
      Rectangle r = new Rectangle(100, 50, 100, 30);
      r.setRotate(i * 360 / 8);
      r.setStroke(Color.color(Math.random(), Math.random(),
        Math.random()));
      r.setFill(Color.WHITE);
      pane.getChildren().add(r);
    }

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 250, 150);
    primaryStage.setTitle("ShowRectangle"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```



Circle


javafx.scene.shape.Circle
-centerX: DoubleProperty
-centerY: DoubleProperty
-radius: DoubleProperty
+Circle()
+Circle(x: double, y: double)
+Circle(x: double, y: double, radius: double)
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 x-coordinate of the center of the circle (default 0).
The y-coordinate of the center of the circle (default 0).
The radius of the circle (default: 0).
Creates an empty Circle.
Creates a Circle with the specified center.
Creates a Circle with the specified center and radius.

Ellipse

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.

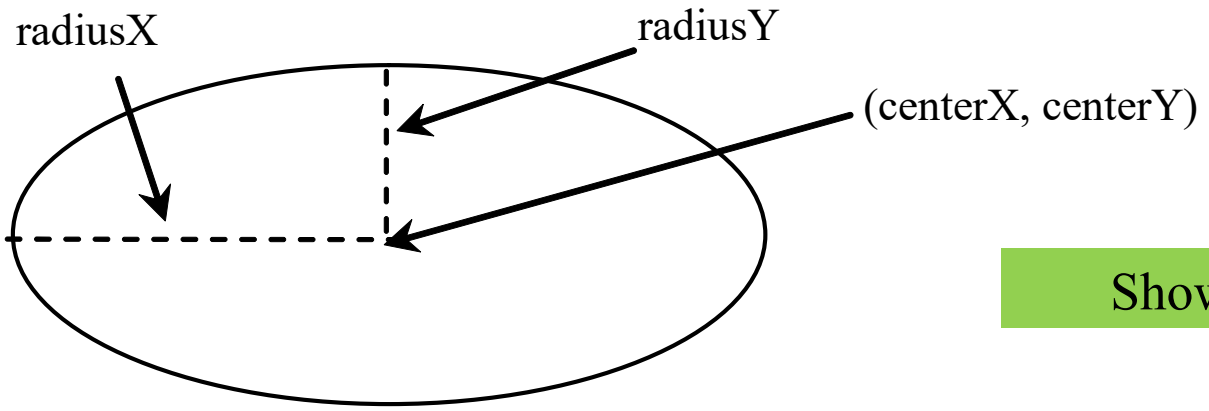
javafx.scene.shape.Ellipse

```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty
```

```
+Ellipse()
+Ellipse(x: double, y: double)
+Ellipse(x: double, y: double,
   radiusX: double, radiusY:
   double)
```

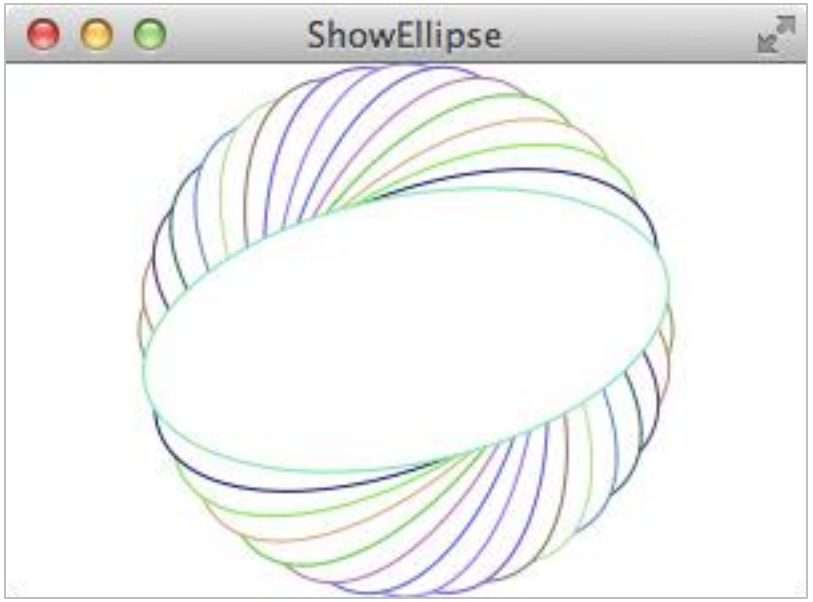
The x-coordinate of the center of the ellipse (default 0).
The y-coordinate of the center of the ellipse (default 0).
The horizontal radius of the ellipse (default: 0).
The vertical radius of the ellipse (default: 0).

Creates an empty `Ellipse`.
Creates an `Ellipse` with the specified center.
Creates an `Ellipse` with the specified center and radiuses.

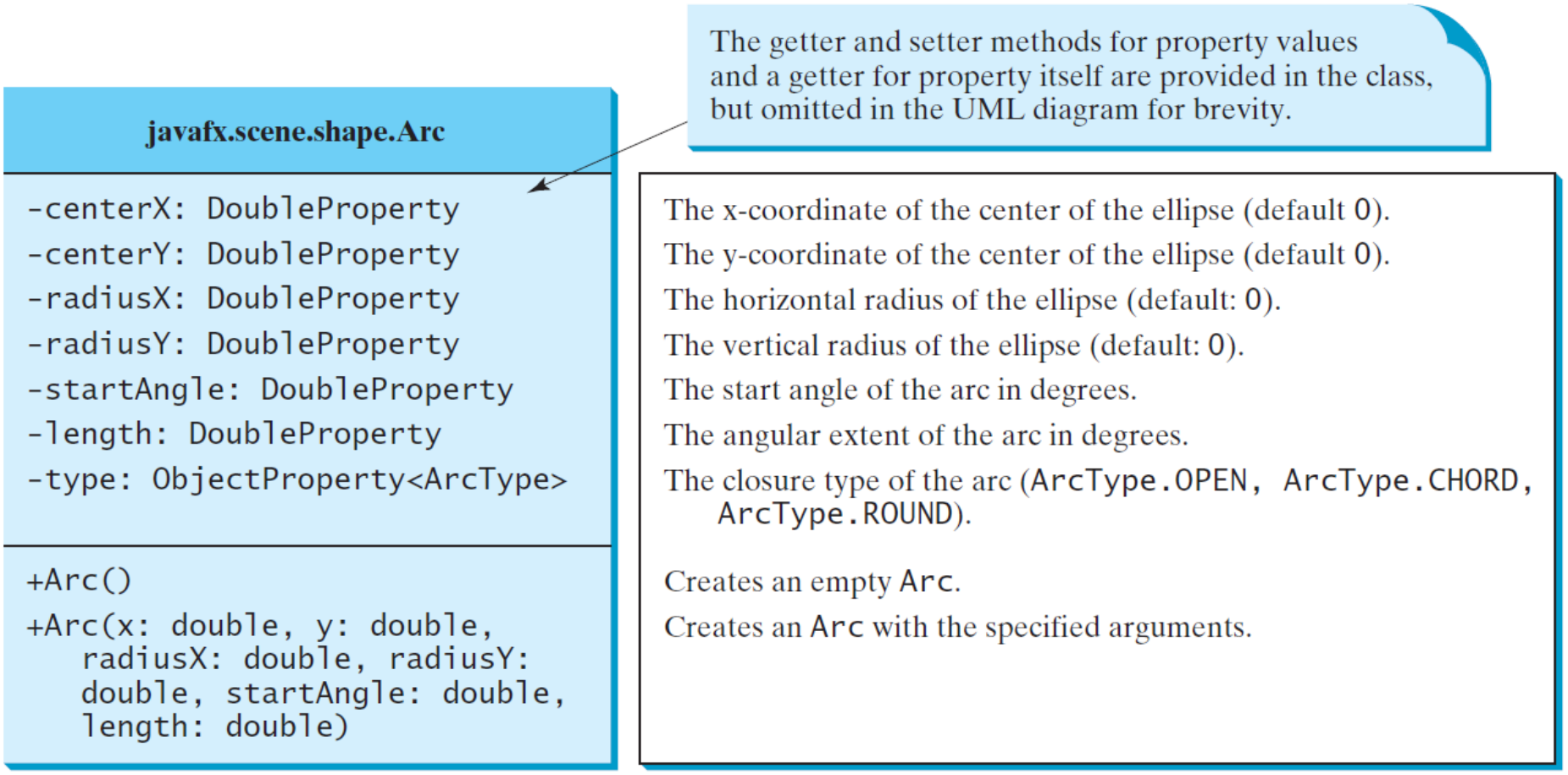


ShowEllipse Run

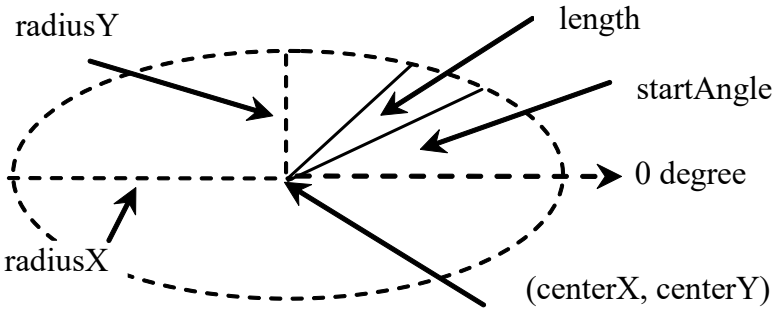
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.shape.Ellipse;

public class ShowEllipse extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane
    Pane pane = new Pane();

    for (int i = 0; i < 16; i++) {
      // Create an ellipse and add it to pane
      Ellipse e1 = new Ellipse(150, 100, 100, 50);
      e1.setStroke(Color.color(Math.random(), Math.random(),
        Math.random()));
      e1.setFill(Color.WHITE);
      e1.setRotate(i * 180 / 16);
      pane.getChildren().add(e1);
    }

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 300, 200);
    primaryStage.setTitle("ShowEllipse"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

public static void main(String[] args) {
    launch(args);
  }
}
```

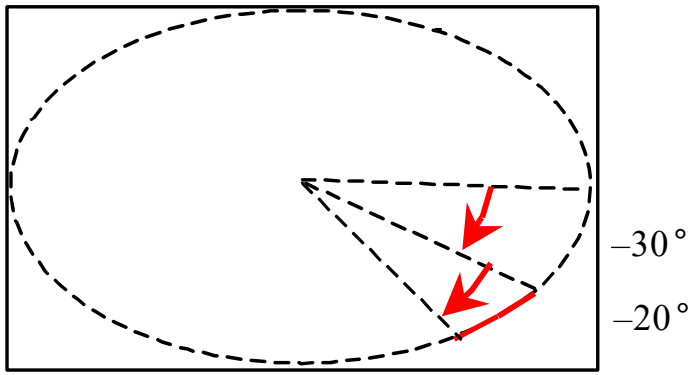


Arc

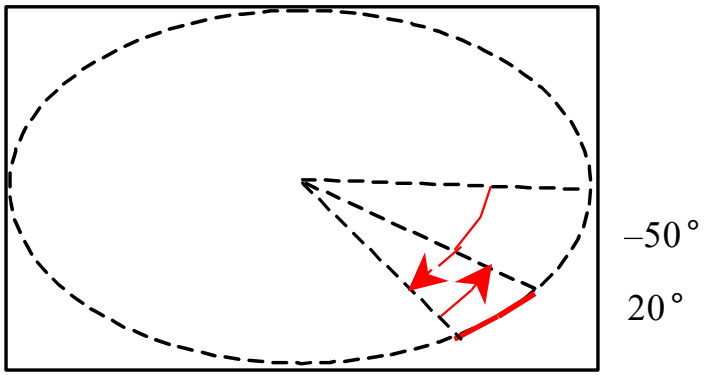

javafx.scene.shape.Arc
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty
-startAngle: DoubleProperty
-length: DoubleProperty
-type: ObjectProperty<ArcType>
+Arc()
+Arc(x: double, y: double, radiusX: double, radiusY: double, startAngle: double, length: double)
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 x-coordinate of the center of the ellipse (default 0).
The y-coordinate of the center of the ellipse (default 0).
The horizontal radius of the ellipse (default: 0).
The vertical radius of the ellipse (default: 0).
The start angle of the arc in degrees.
The angular extent of the arc in degrees.
The closure type of the arc (ArcType.OPEN, ArcType.CHORD, ArcType.ROUND).
Creates an empty Arc.
Creates an Arc with the specified arguments.

Arc Examples





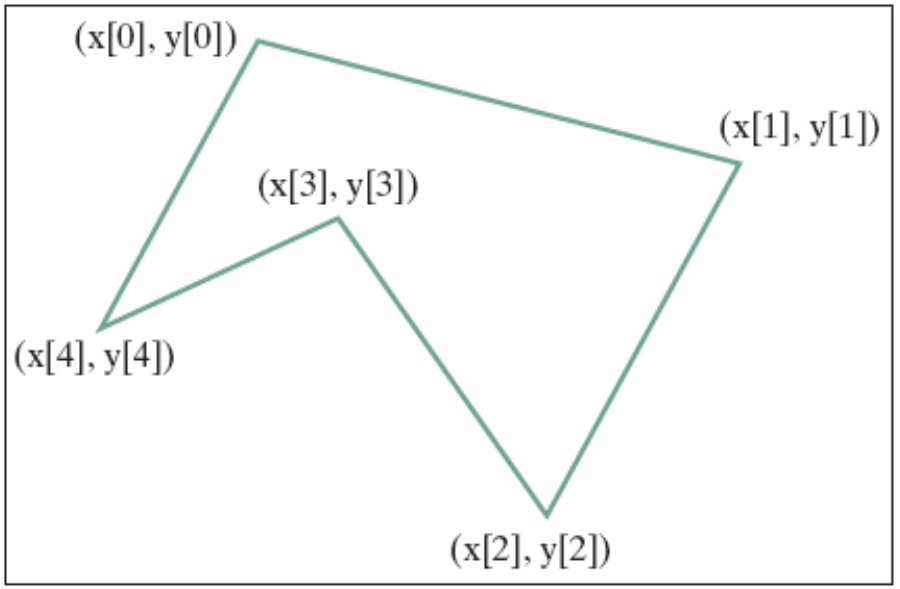
(a) Negative starting angle –30° and negative spanning angle –20°



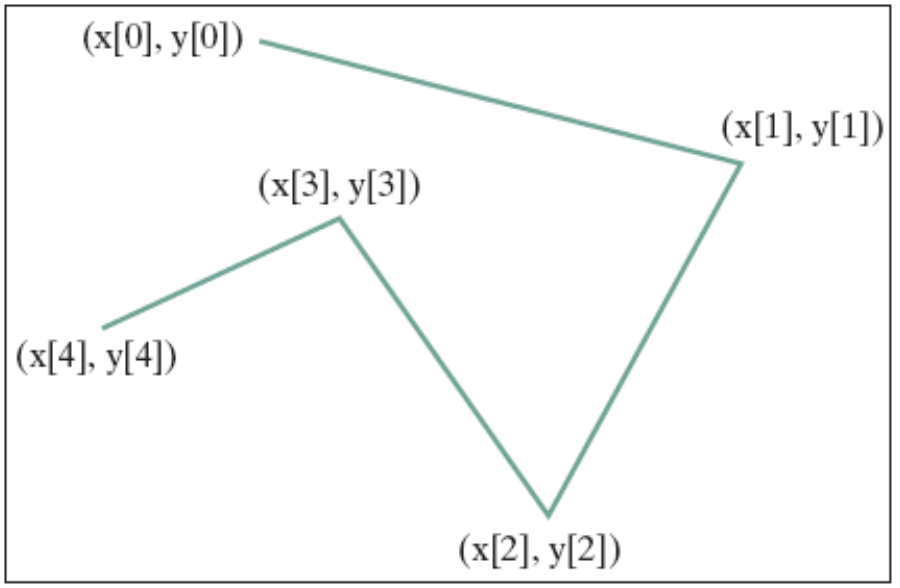
(b) Negative starting angle –50° and positive spanning angle 20°

ShowArc Run

Polygon and Polyline

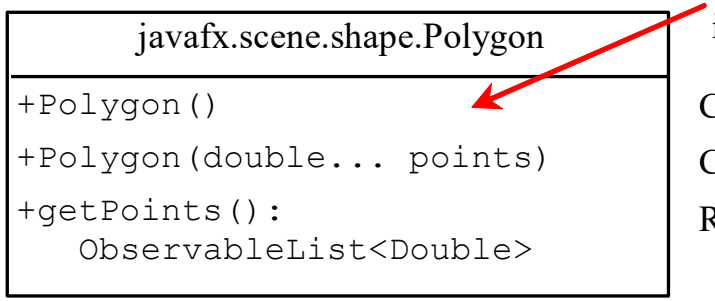


(a) Polygon



(b) Polyline

Polygon



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

Creates an empty polygon.

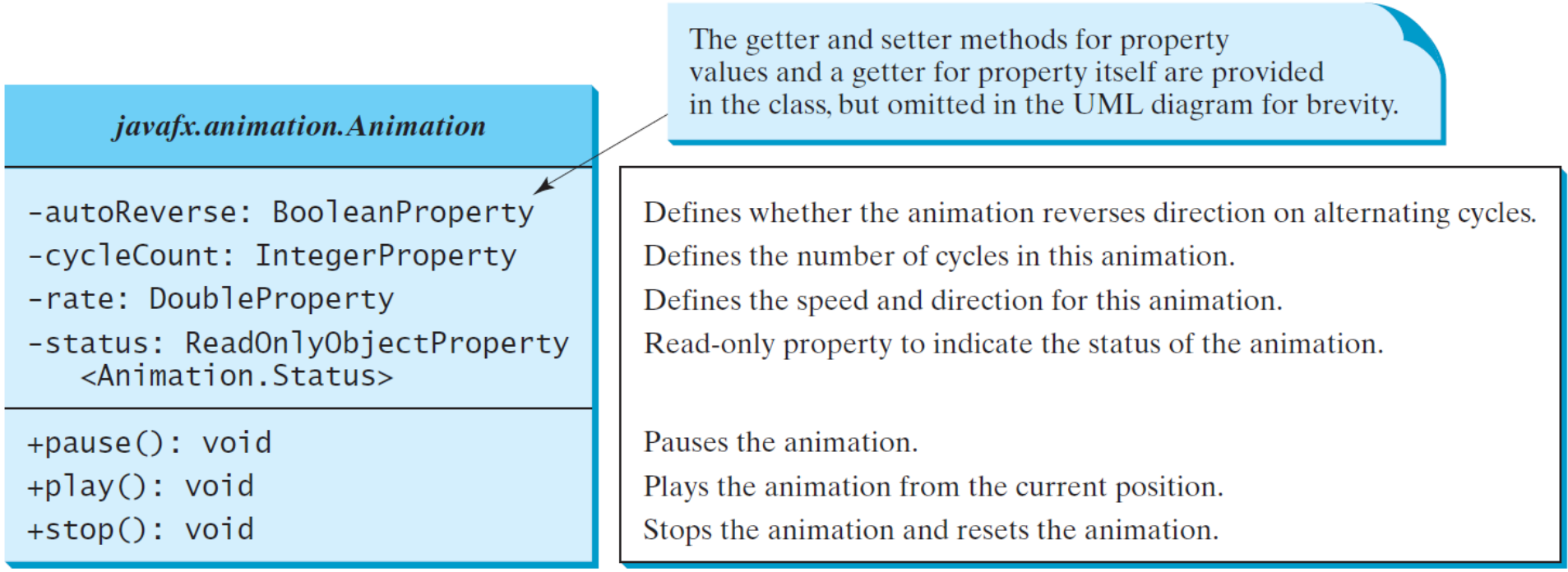
Creates a polygon with the given points.

Returns a list of double values as x- and y-coordinates of the points.

ShowPolygon Run

Animation

JavaFX provides the **Animation** class with the core functionality for all animations.



PathTransition

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.animation.PathTransition

```
-duration: ObjectProperty<Duration>
-node: ObjectProperty<Node>
-orientation: ObjectProperty
   <PathTransition.OrientationType>
-path: ObjectType<Shape>
```

```
+PathTransition()
+PathTransition(duration: Duration,
   path: Shape)
+PathTransition(duration: Duration,
   path: Shape, node: Node)
```

The duration of this transition.

The target node of this transition.

The orientation of the node along the path.

The shape whose outline is used as a path to animate the node move.

Creates an empty `PathTransition`.

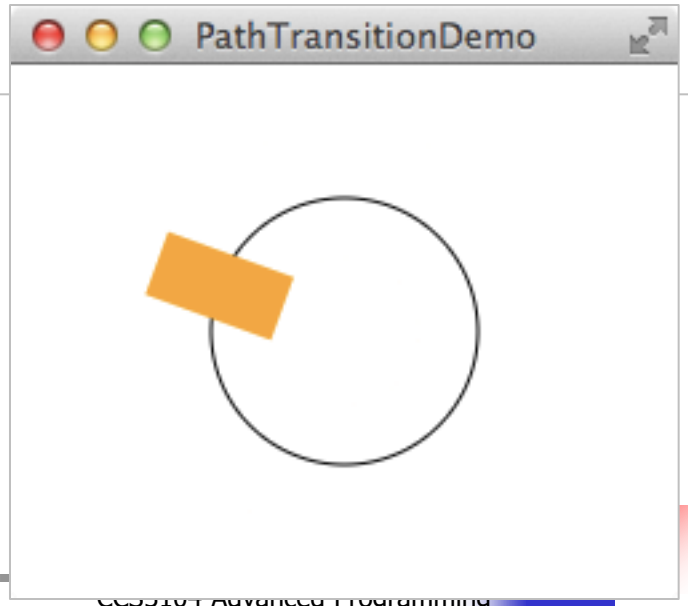
Creates a `PathTransition` with the specified duration and path.

Creates a `PathTransition` with the specified duration, path, and node.

PathTransitionDemo Run

FlagRisingAnimation Run

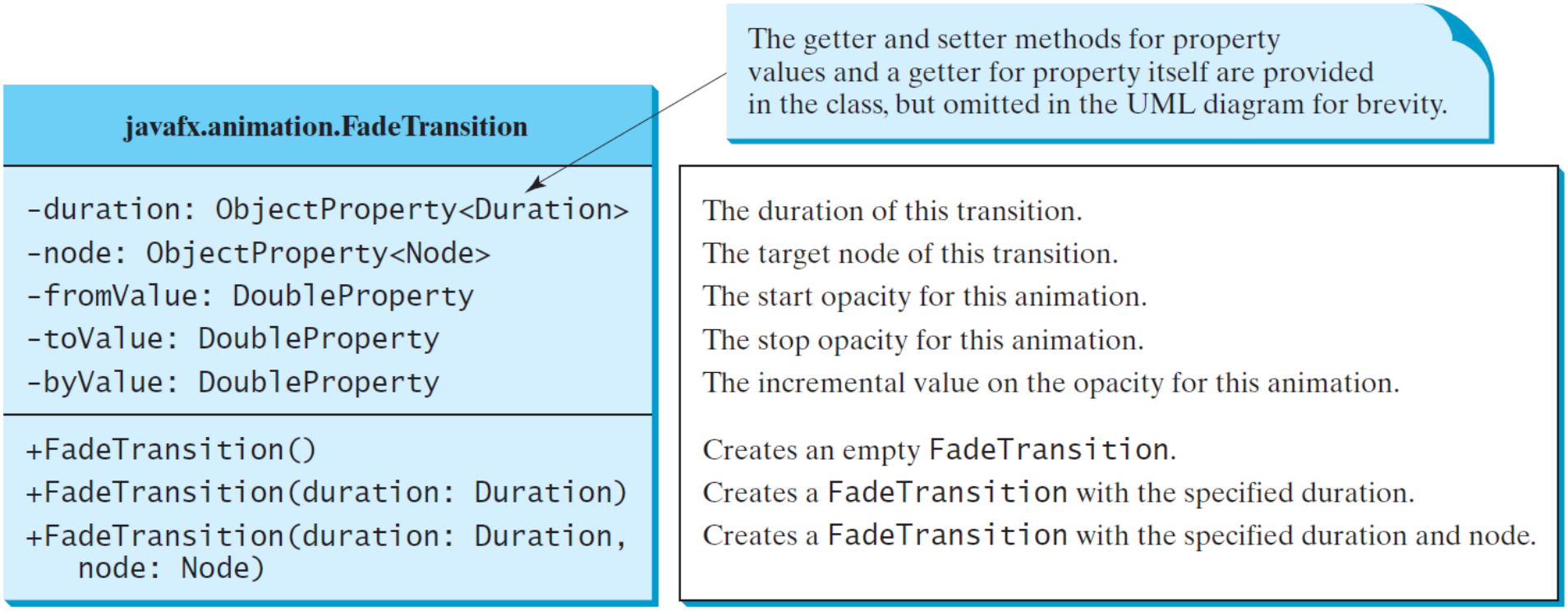
```
import javafx.animation.PathTransition;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
import javafx.util.Duration;

public class PathTransitionDemo extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane
    Pane pane = new Pane();

    // Create a rectangle
    Rectangle rectangle = new Rectangle (0, 0, 25, 50);
    rectangle.setFill(Color.ORANGE);

    // Create a circle
    Circle circle = new Circle(125, 100, 50);
    circle.setFill(Color.WHITE);
    circle.setStroke(Color.BLACK);

    // Add circle and rectangle to the pane
    pane.getChildren().add(circle);
    pane.getChildren().add(rectangle);

    // Create a path transition
    PathTransition pt = new PathTransition();
    pt.setDuration(Duration.millis(4000));
    pt.setPath(circle);
    pt.setNode(rectangle);
    pt.setOrientation(
      PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT);
    pt.setCycleCount(Timeline.INDEFINITE);
    pt.setAutoReverse(true);
    pt.play(); // Start animation

    circle.setOnMousePressed(e -> pt.pause());
    circle.setOnMouseReleased(e -> pt.play());

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 250, 200);
    primaryStage.setTitle("PathTransitionDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```



```
import javafx.animation.PathTransition;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.image.ImageView;
import javafx.scene.layout.Pane;
import javafx.scene.shape.Line;
import javafx.stage.Stage;
import javafx.util.Duration;

public class FlagRisingAnimation extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane
    Pane pane = new Pane();

    // Add an image view and add it to pane
    ImageView imageView = new ImageView("image/us.gif");
    pane.getChildren().add(imageView);

    // Create a path transition
    PathTransition pt = new PathTransition(Duration.millis(10000),
      new Line(100, 200, 100, 0), imageView);
    pt.setCycleCount(5);
    pt.play(); // Start animation

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 250, 200);
    primaryStage.setTitle("FlagRisingAnimation"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

public static void main(String[] args) {
    launch(args);
  }
}
```

FadeTransition

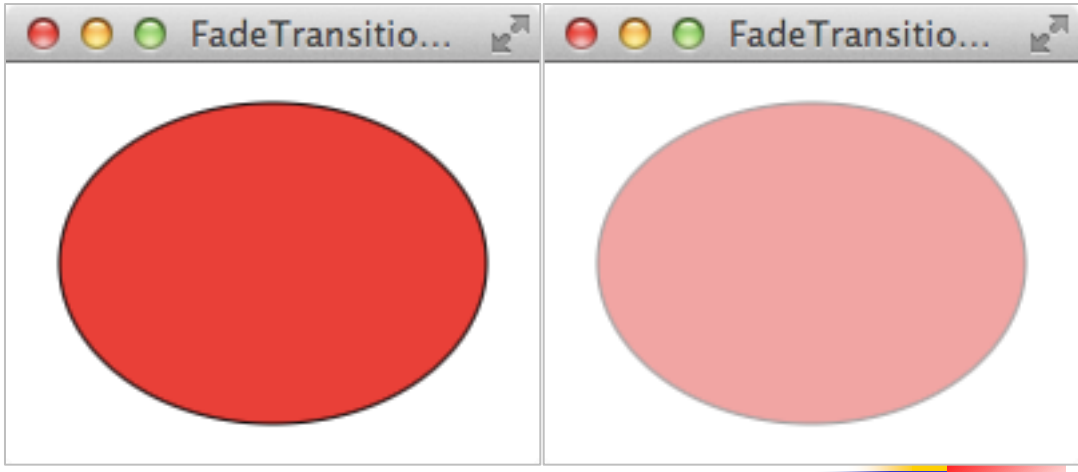
The **FadeTransition** class animates the change of the opacity in a node over a given time.



FadeTransitionDemo Run

```
import javafx.animation.FadeTransition;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Ellipse;
import javafx.stage.Stage;
import javafx.util.Duration;

public class FadeTransitionDemo extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Place an ellipse to the pane
    Pane pane = new Pane();
    Ellipse ellipse = new Ellipse(10, 10, 100, 50);
    ellipse.setFill(Color.RED);
    ellipse.setStroke(Color.BLACK);
    ellipse.centerXProperty().bind(pane.widthProperty().divide(2));
    ellipse.centerYProperty().bind(pane.heightProperty().divide(2));
    ellipse.radiusXProperty().bind(
      pane.widthProperty().multiply(0.4));
    ellipse.radiusYProperty().bind(
      pane.heightProperty().multiply(0.4));
    pane.getChildren().add(ellipse);

    // Apply a fade transition to ellipse
    FadeTransition ft =
      new FadeTransition(Duration.millis(3000), ellipse);
    ft.setFromValue(1.0);
    ft.setToValue(0.1);
    ft.setCycleCount(Timeline.INDEFINITE);
    ft.setAutoReverse(true);
    ft.play(); // Start animation
```

```
    // Control animation
    ellipse.setOnMousePressed(e -> ft.pause());
    ellipse.setOnMouseReleased(e -> ft.play());

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 200, 150);
    primaryStage.setTitle("FadeTransitionDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

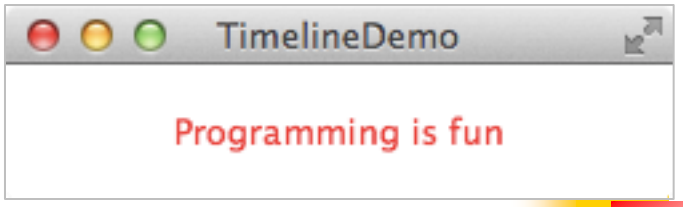


Timeline

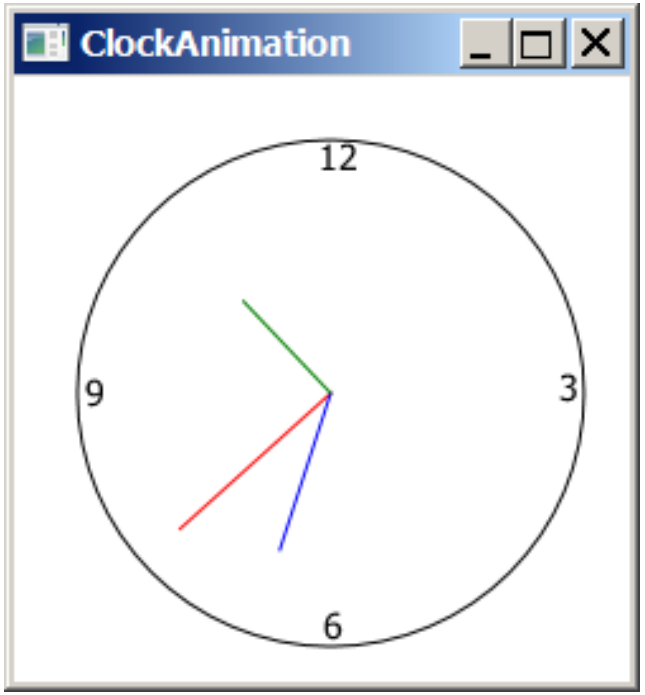
PathTransition and **FadeTransition** define specialized animations. The **Timeline** class can be used to program any animation using one or more **KeyFrame**s. Each **KeyFrame** is executed sequentially at a specified time interval. **Timeline** inherits from **Animation**.

TimelineDemo Run

```
import javafx.animation.Animation;
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.util.Duration;

public class TimelineDemo extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    StackPane pane = new StackPane();
    Text text = new Text(20, 50, "Programming if fun");
    text.setFill(Color.RED);
    pane.getChildren().add(text); // Place text into the stack pane

    // Create a handler for changing text
    EventHandler<ActionEvent> eventHandler = e -> {
      if (text.getText().length() != 0) {
        text.setText("");
      }
      else {
        text.setText("Programming is fun");
      }
    };

    // Create an animation for alternating text
    Timeline animation = new Timeline(
      new KeyFrame(Duration.millis(500), eventHandler));
    animation.setCycleCount(Timeline.INDEFINITE);
    animation.play(); // Start animation

    // Pause and resume animation
    text.setOnMouseClicked(e -> {
      if (animation.getStatus() == Animation.Status.PAUSED) {
        animation.play();
      }
      else {
        animation.pause();
      }
    });

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 250, 50);
    primaryStage.setTitle("TimelineDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  public static void main(String[] args) {
    launch(args);
  }
}
```

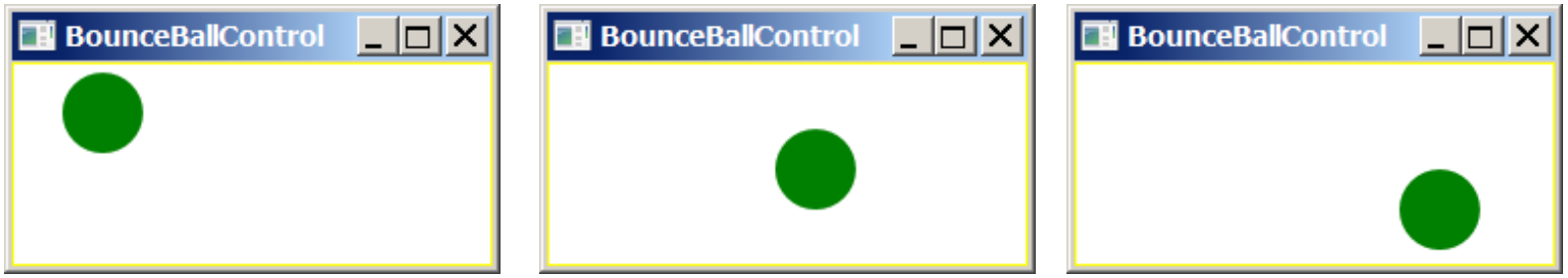


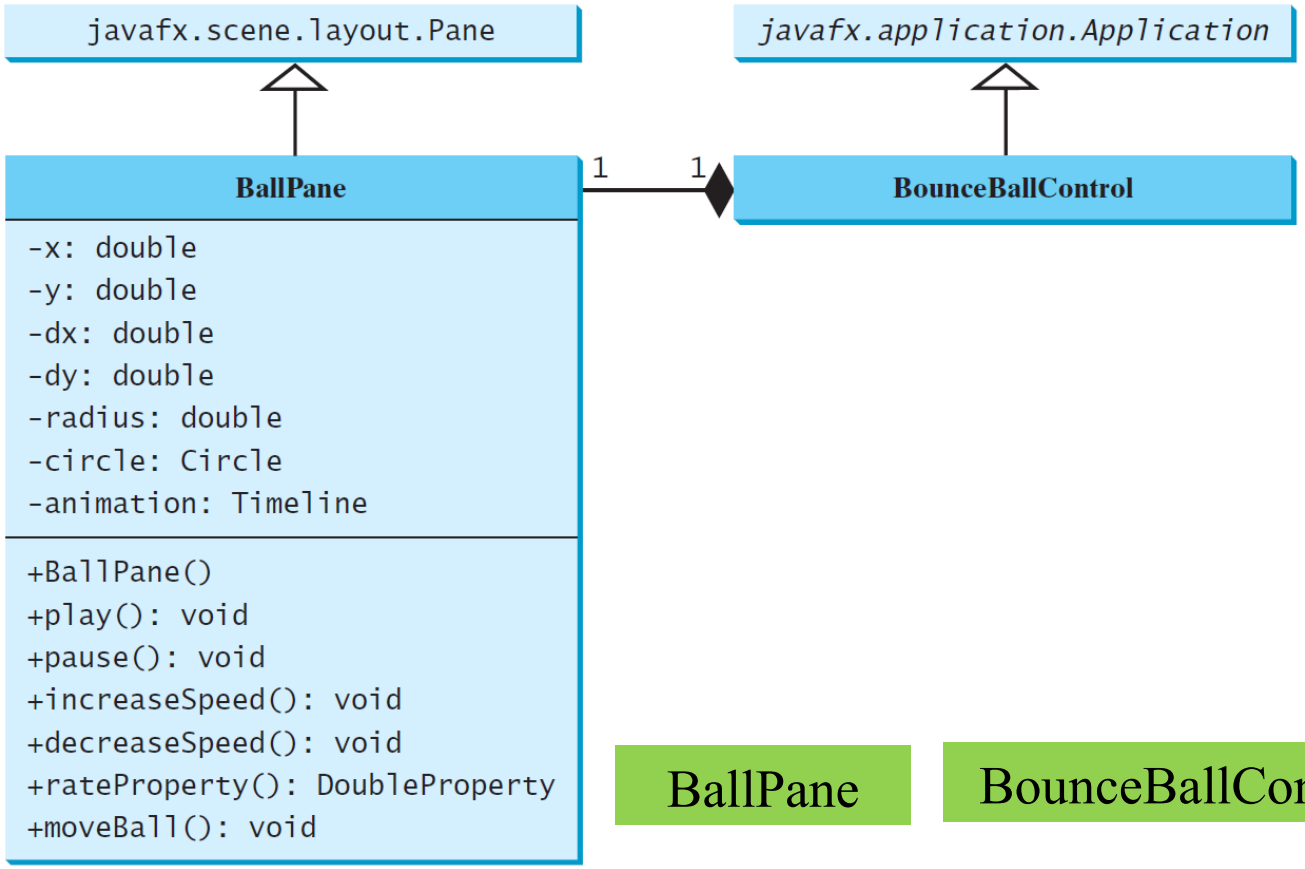
Clock Animation



ClockAnimation Run

Case Study: Bouncing Ball

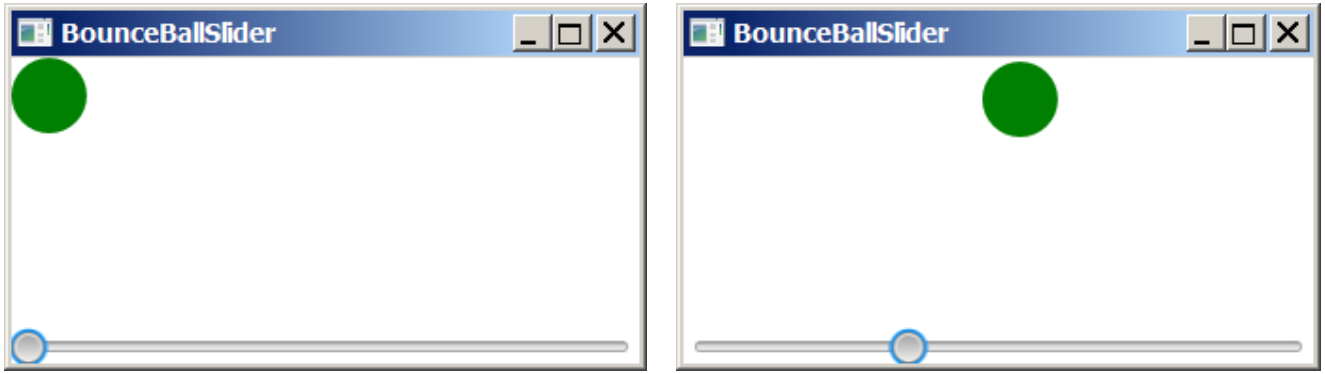




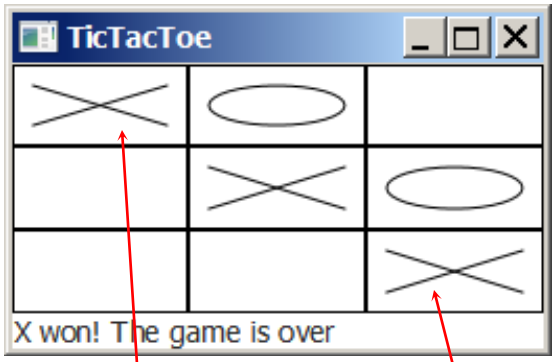
BallPane BounceBallControl Run

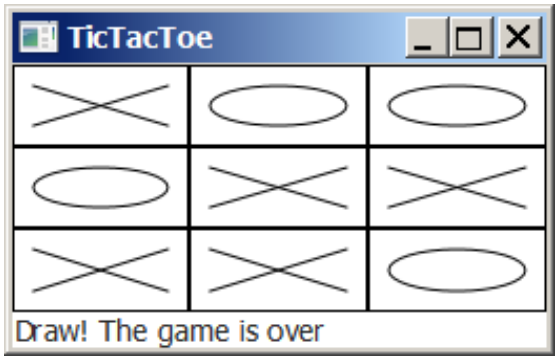
Case Study: Bounce Ball Slider

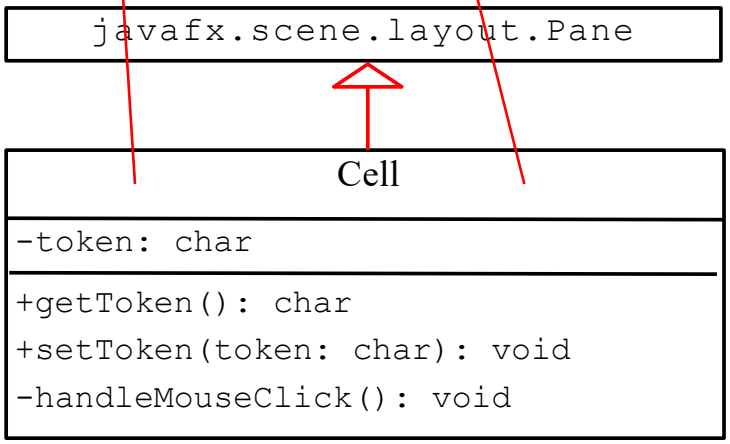
Listing 15.17 gives a program that displays a bouncing ball. You can add a slider to control the speed of the ball movement.



Case Study: TicTacToe







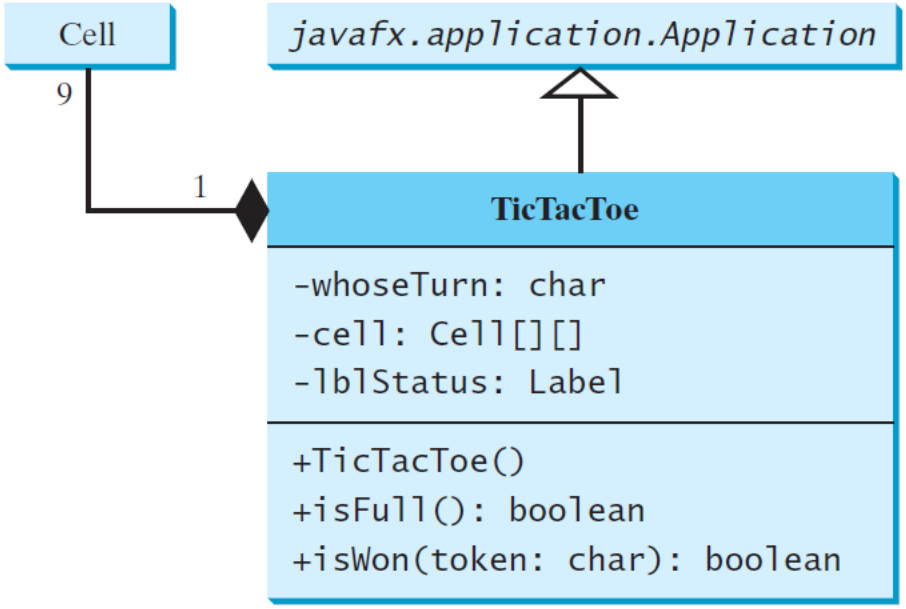
Token used in the cell (default: ' ').

Returns the token in the cell.

Sets a new token in the cell.

Handles a mouse click event.

Case Study: TicTacToe, cont.



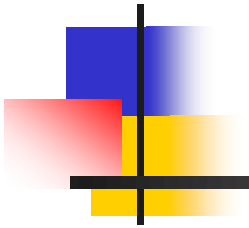
Indicates which player has the turn, initially X.
A 3 × 3, two-dimensional array for cells.
A label to display game status.

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

TicTacToe Run

Summary

- Controls – Label, Button, CheckBox, RadioButton, TextField, …
  - Esp. slides #8, #11, #14, #17 and #20.



End of Chapter 3