Chapter 5
JavaFX UI Controls

Motivations

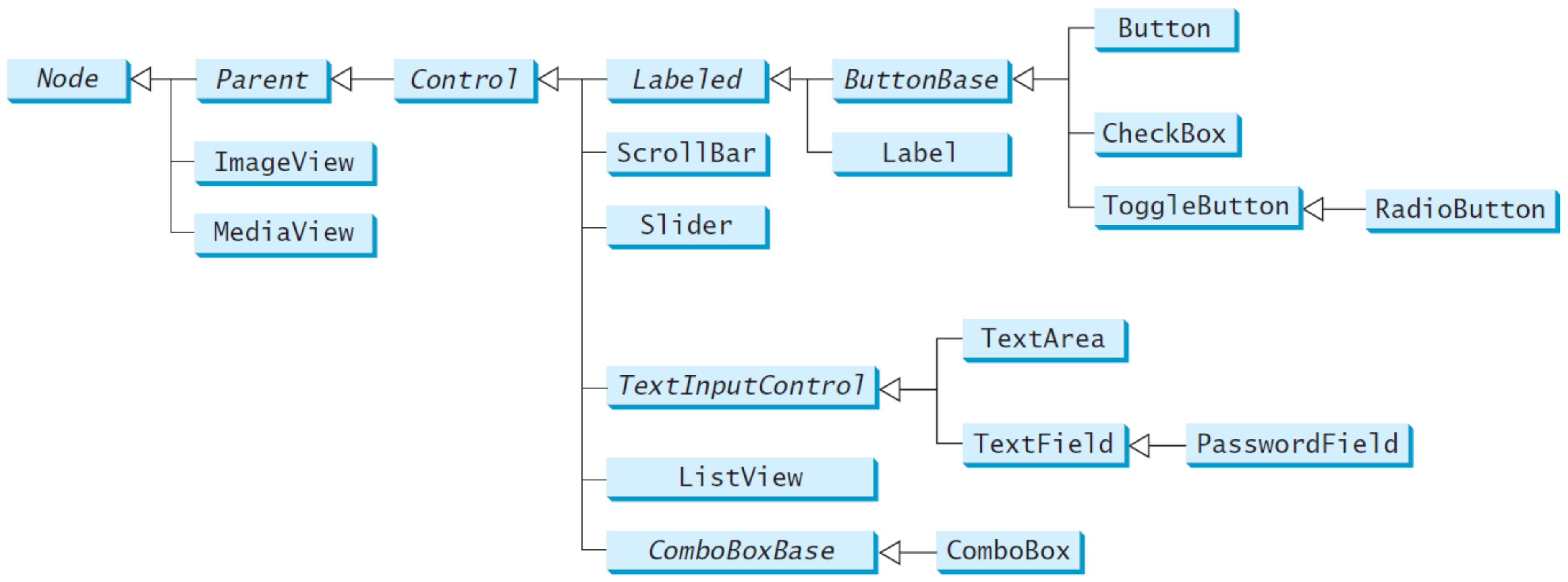
A graphical user interface (GUI) makes a system user-friendly and easy to use. Creating a GUI requires creativity and knowledge of how GUI components work. Since the GUI components in Java are very flexible and versatile, you can create a wide assortment of useful user interfaces.

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

Objectives

- ☞ To create a label with text and graphic using the **Label** class.
- ☞ 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.
- ☞ To create a check box using the **CheckBox** class and to create a radio button using the **RadioButton** class and group radio buttons using a **ToggleGroup**.
- ☞ To enter data using the **TextField** class and password using the **PasswordField** class.
- ☞ To enter data in multiple lines using the **TextArea** class.
- ☞ To select a single item using **ComboBox**.
- ☞ To select a single or multiple items using **ListView**.
- ☞ To select a range of values using **ScrollBar**.
- ☞ To select a range of values using **Slider** and explore differences between **ScrollBar** and **Slider**.

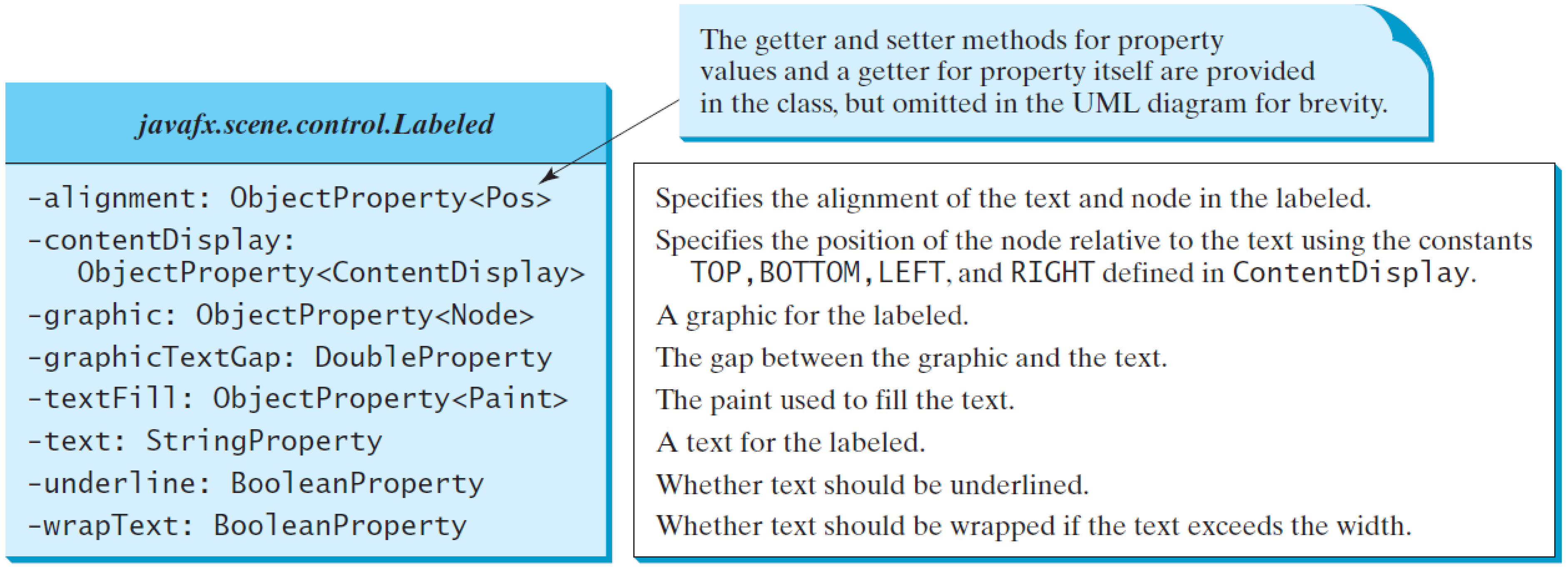
Frequently Used UI Controls



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

Labeled

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



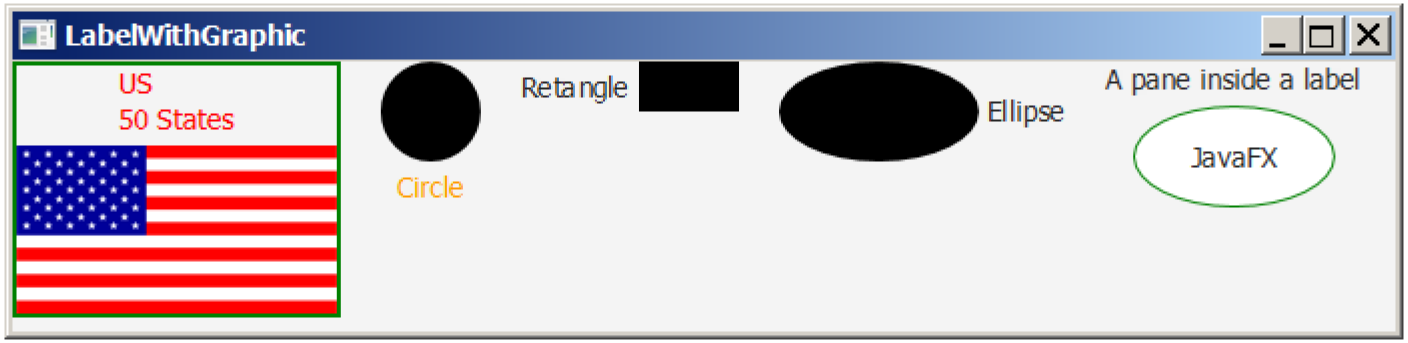
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

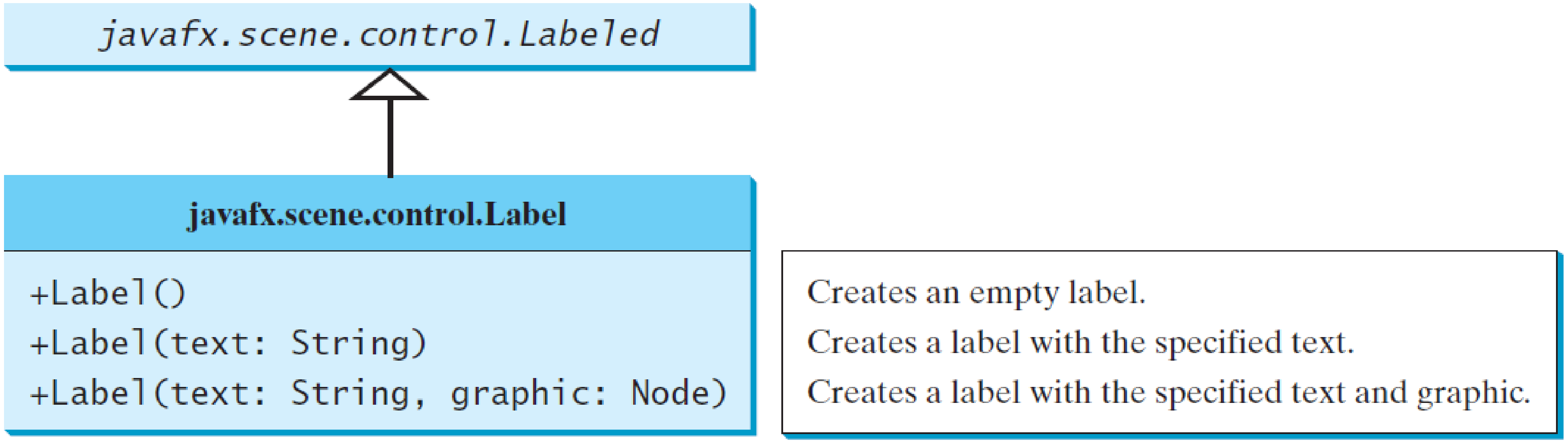
javafx.scene.control.Labeled

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

Label

The Label class defines labels.



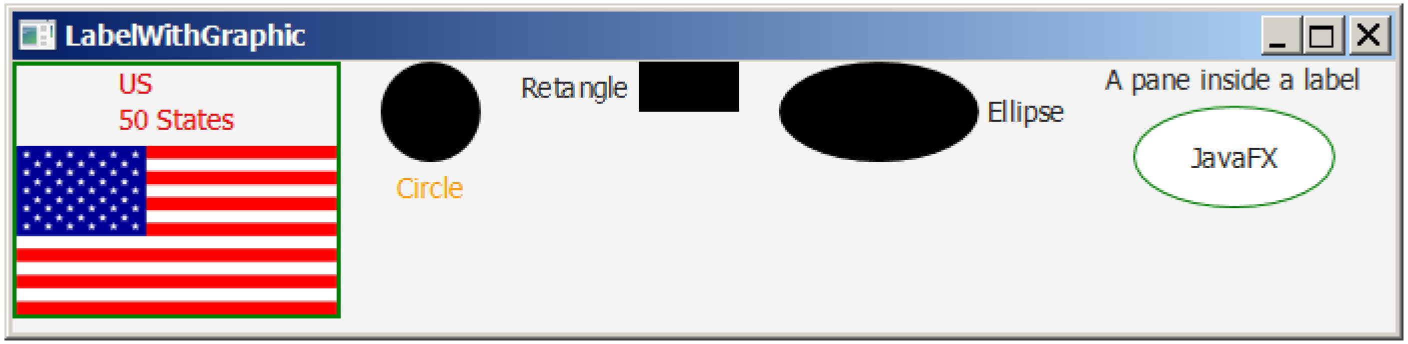

javafx.scene.control.Labeled
javafx.scene.control.Label
+Label()
+Label(text: String)
+Label(text: String, graphic: Node)
Creates an empty label.
Creates a label with the specified text.
Creates a label with the specified text and graphic.

Label

```
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);

```

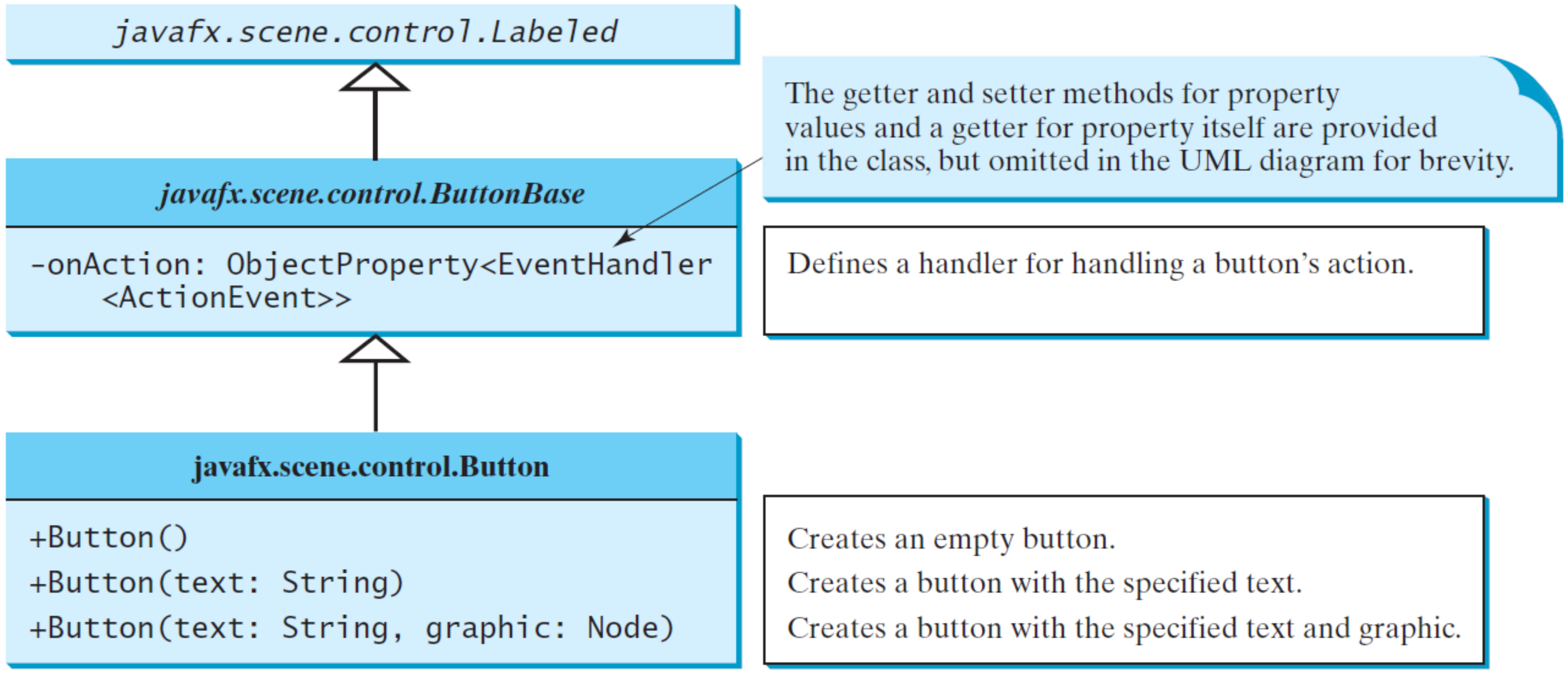
Label

```
        // 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
    }
}
```



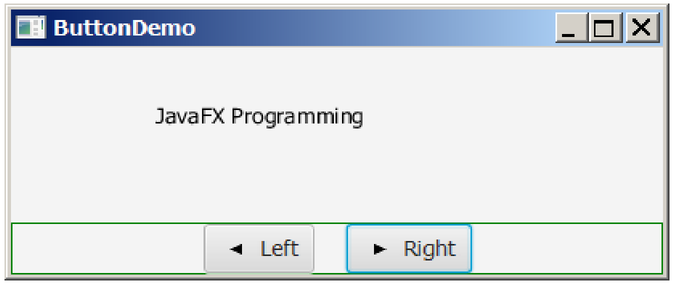
ButtonBase and Button

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


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

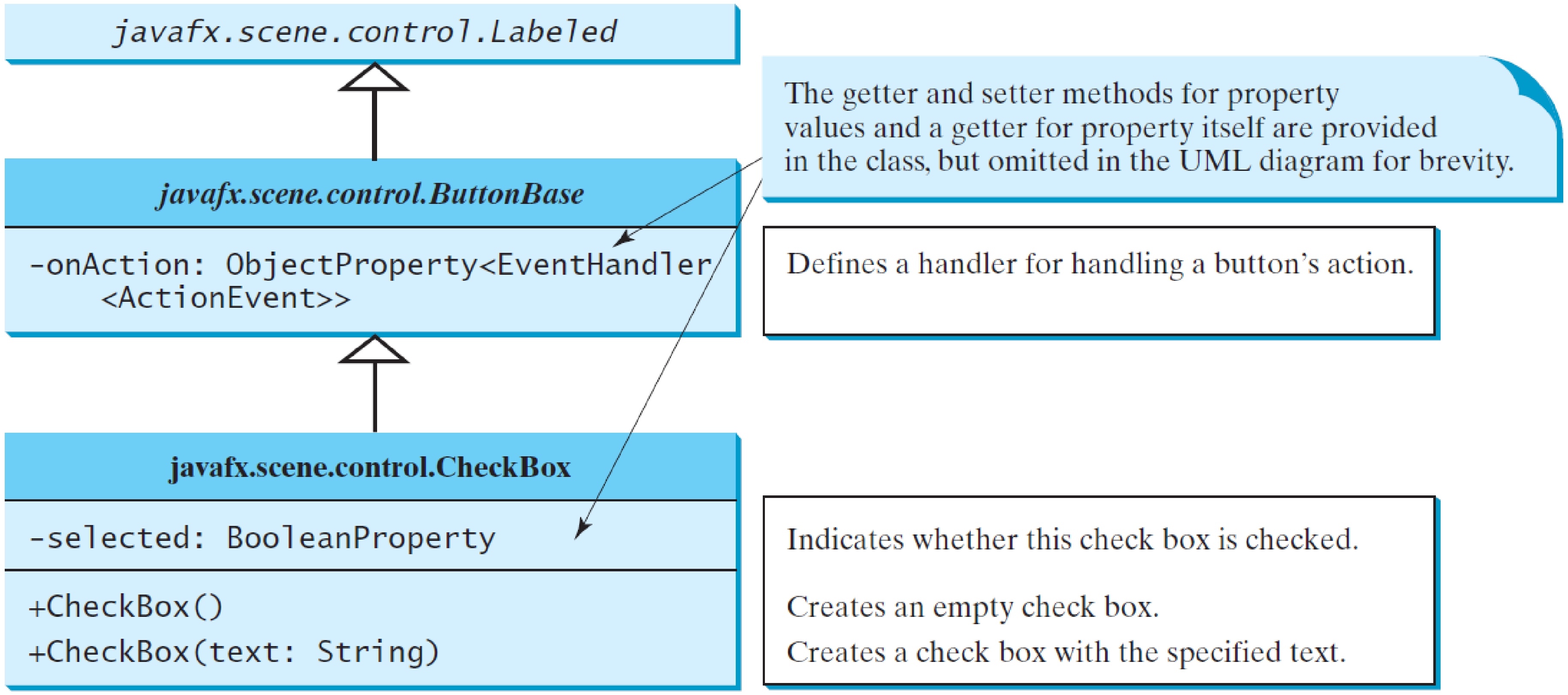
Button Example

```java
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
  }
}
```



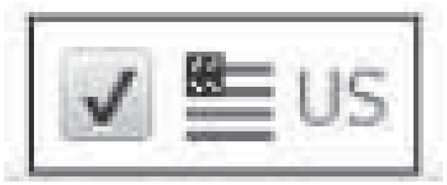
CheckBox

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


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

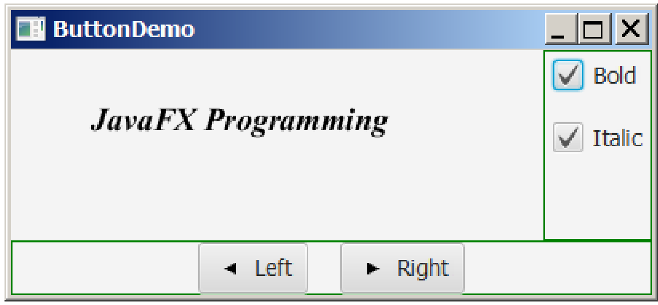
CheckBox

```
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));
```



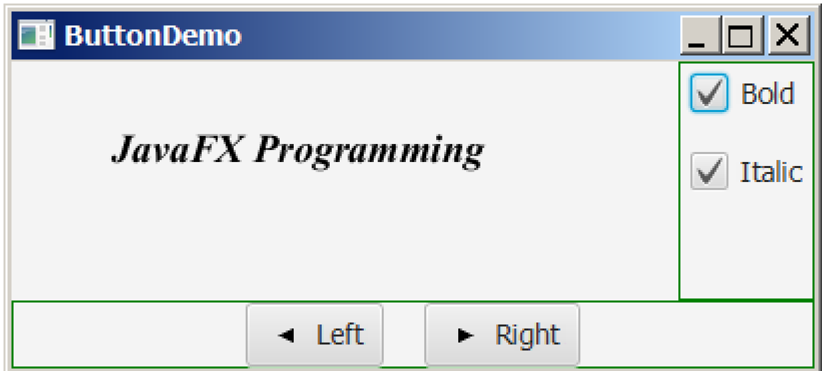
CheckBox Example

```
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);
```



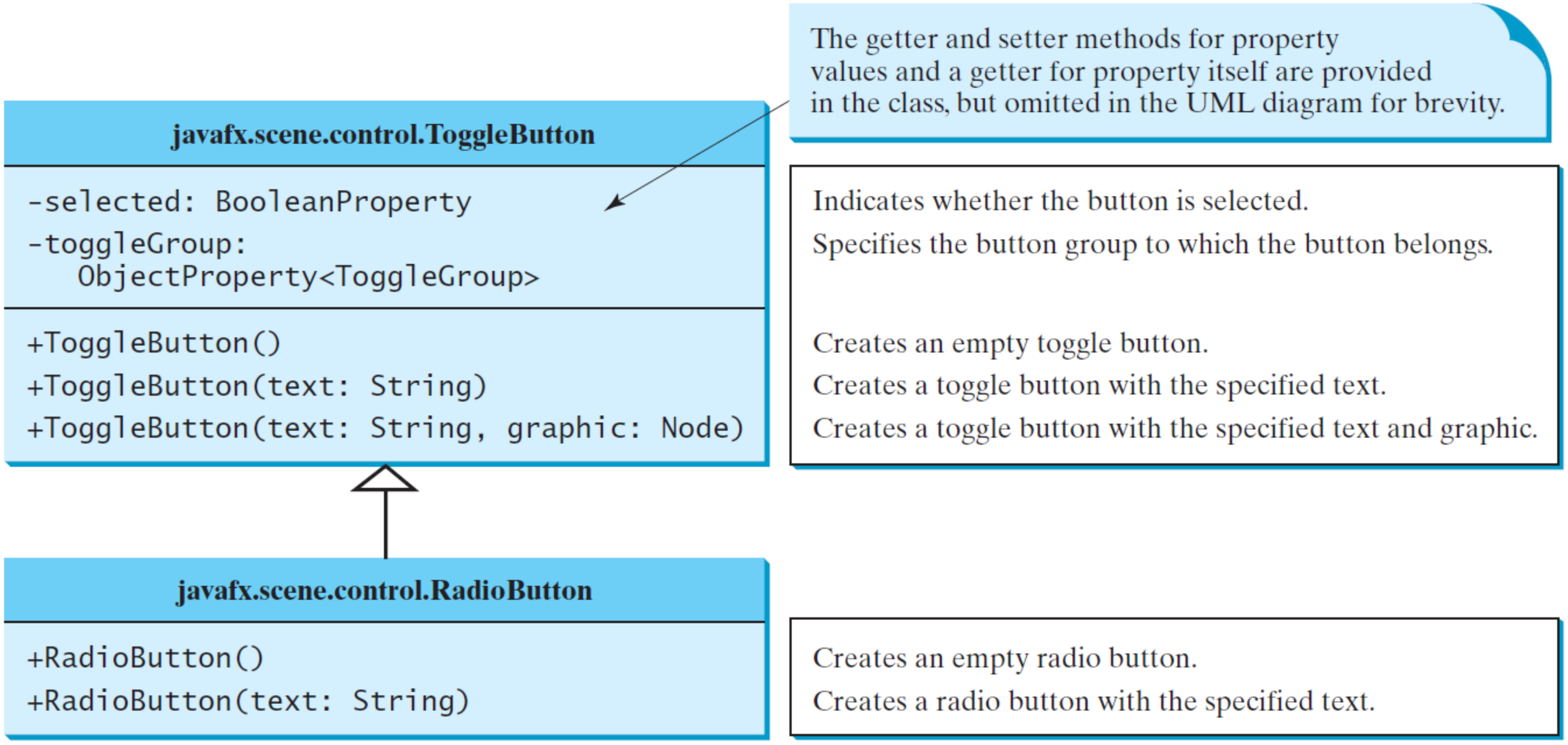
CheckBox Example

```

    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
  }
}
```



RadioButton

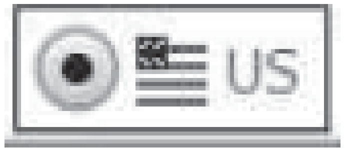
Radio buttons, also known as *option buttons*, enable you to choose a single item from a group of choices. In appearance radio buttons resemble check boxes, but check boxes display a square that is either checked or blank, whereas radio buttons display a circle that is either filled (if selected) or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



The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.
javafx.scene.control.ToggleButton
-selected: BooleanProperty
-toggleGroup: ObjectProperty<ToggleGroup>
+ToggleButton()
+ToggleButton(text: String)
+ToggleButton(text: String, graphic: Node)
Indicates whether the button is selected.
Specifies the button group to which the button belongs.
Creates an empty toggle button.
Creates a toggle button with the specified text.
Creates a toggle button with the specified text and graphic.
javafx.scene.control.RadioButton
+RadioButton()
+RadioButton(text: String)
Creates an empty radio button.
Creates a radio button with the specified text.

RadioButton

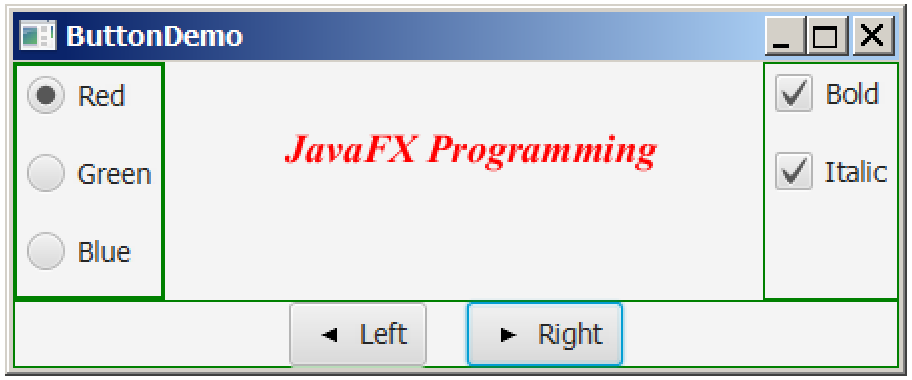
```
RadioButton rbUS = new RadioButton("US");
rbUS.setGraphic(new ImageView("image/usIcon.gif"));
rbUS.setTextFill(Color.GREEN);
rbUS.setContentDisplay(ContentDisplay.LEFT);
rbUS.setStyle("-fx-border-color: black");
rbUS.setSelected(true);
rbUS.setPadding(new Insets(5, 5, 5,));
```



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

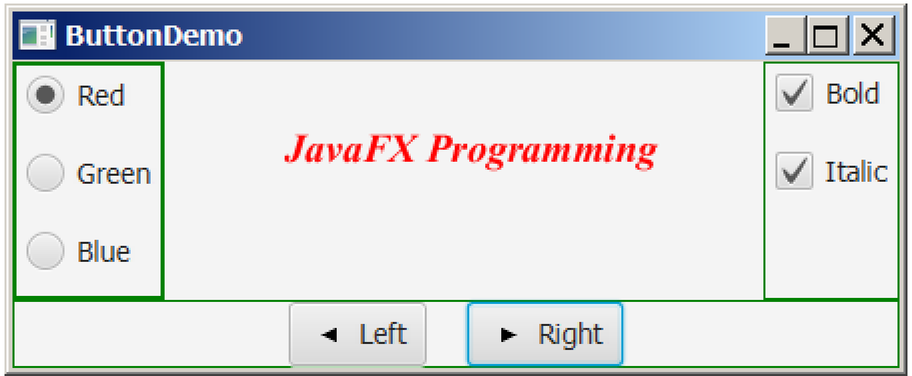
RadioButton Example

```java
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);
```



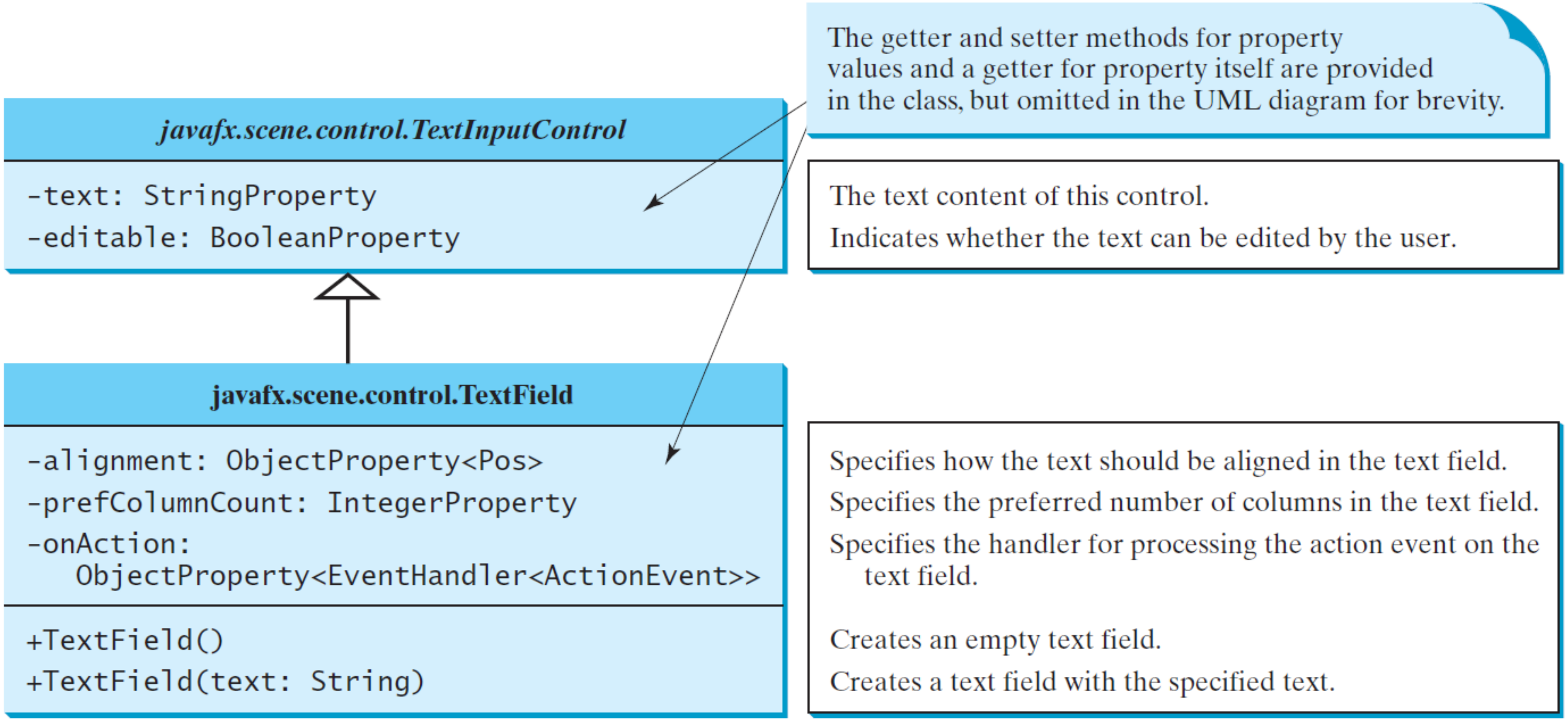
RadioButton Example

```
    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;
  }
}
```



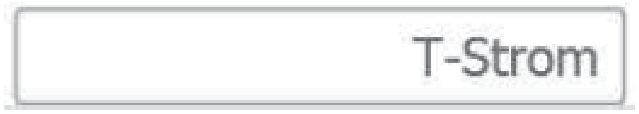
TextField

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


The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.
javafx.scene.control.TextInputControl
-text: StringProperty
-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.

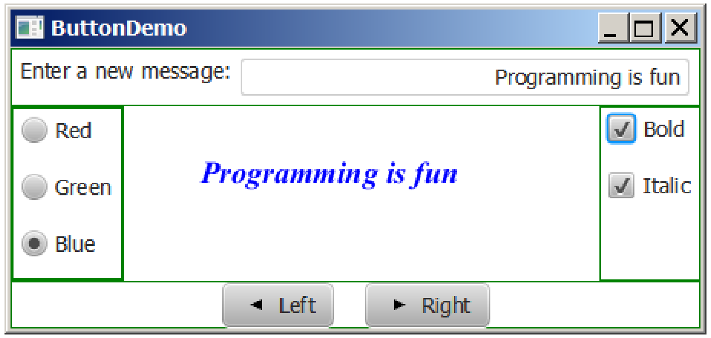
TextField

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



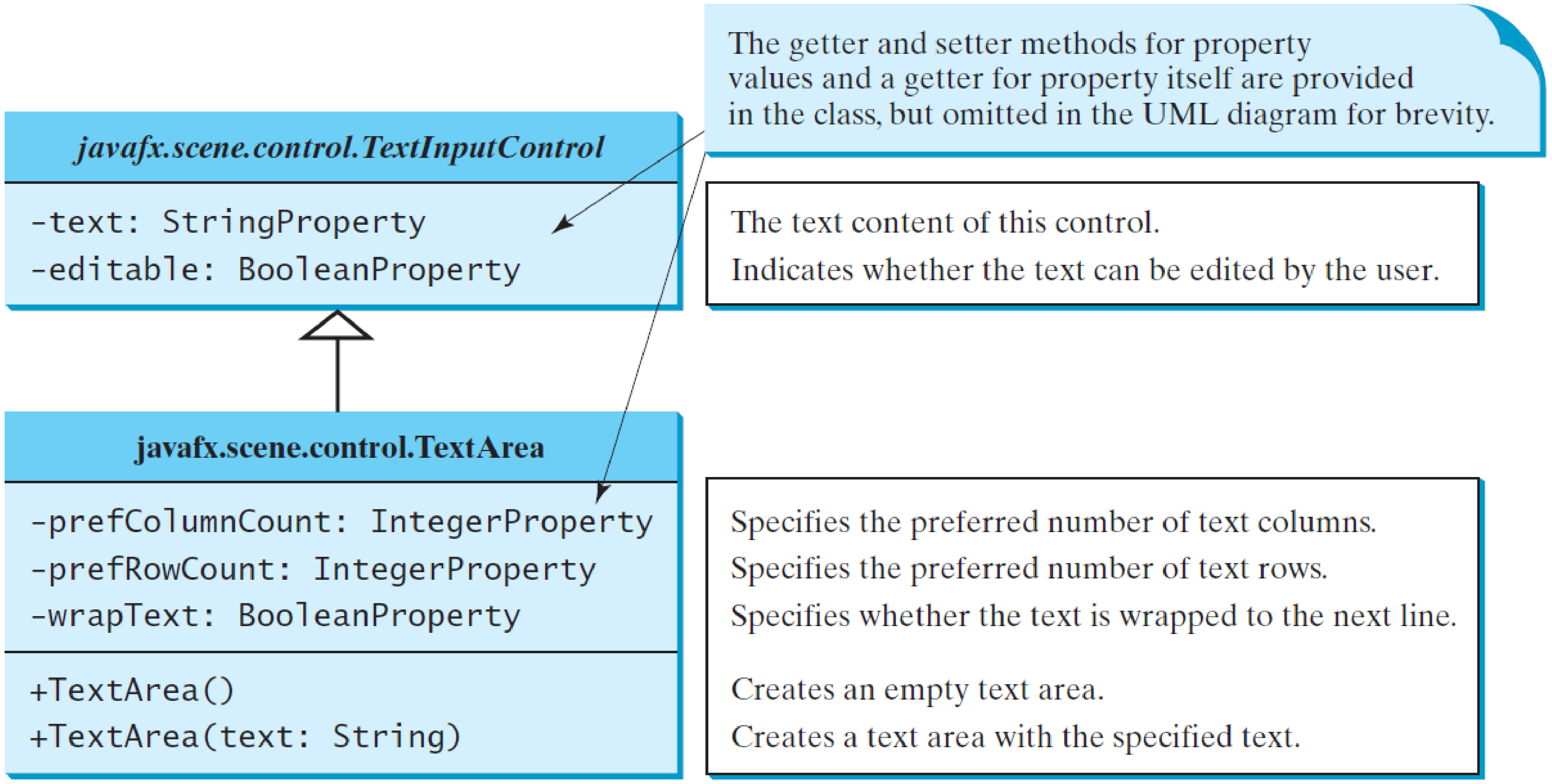
TextField Example

```
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;
  }
}
```

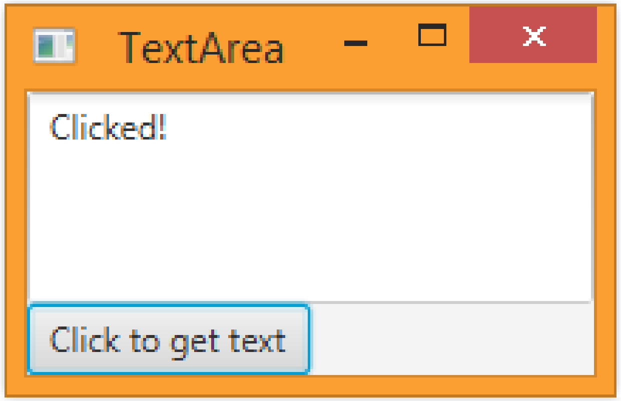


TextArea

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


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.TextArea
-prefColumnCount: IntegerProperty
-prefRowCount: IntegerProperty
-wrapText: BooleanProperty
+TextArea()
+TextArea(text: String)
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.

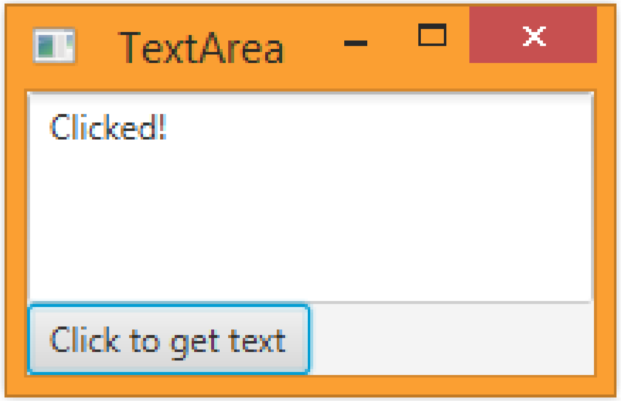
TextArea Example



```
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));
```

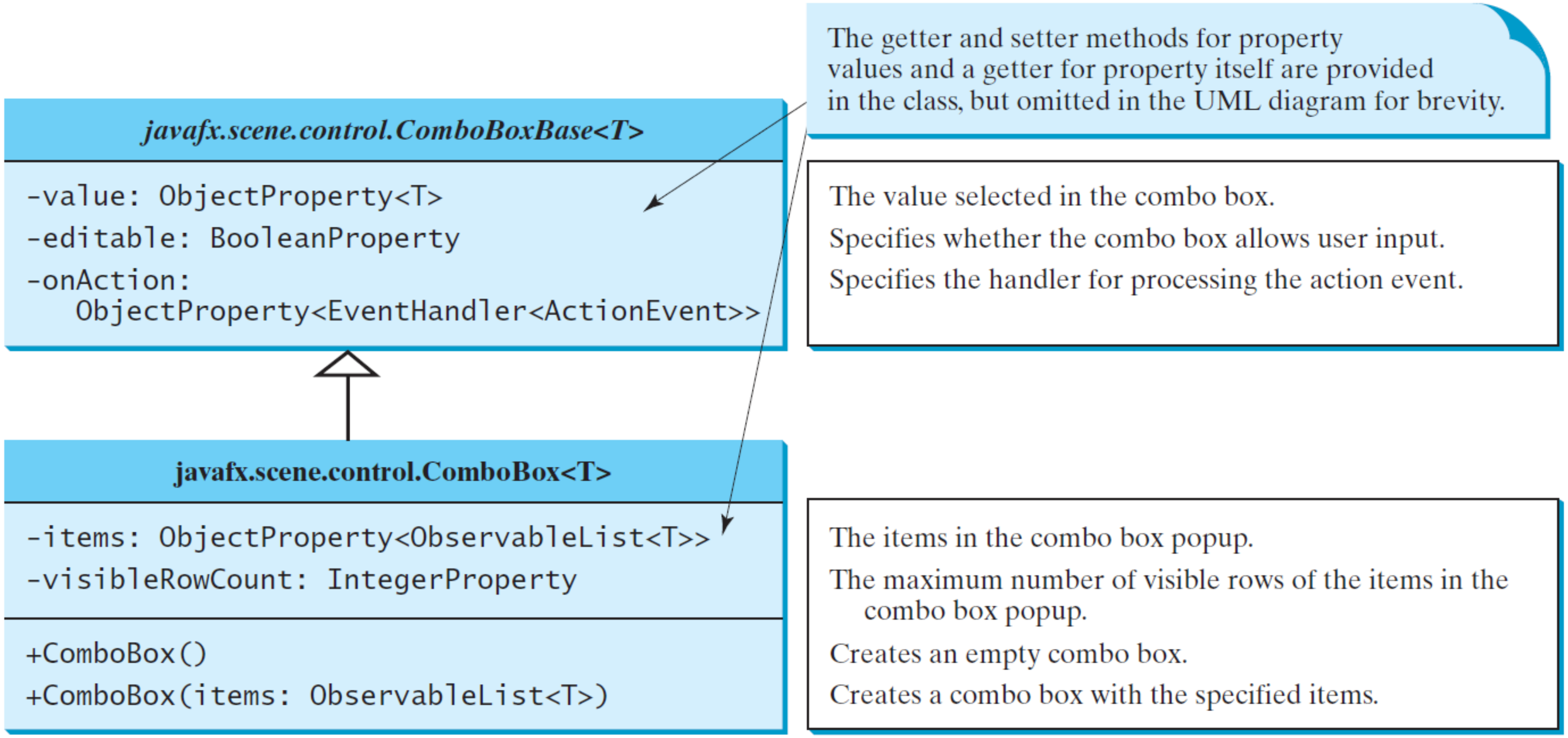
TextArea Example

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class TextAreaDemo extends Application  {
    @Override
    public void start(Stage primaryStage) throws Exception {

        TextArea textArea = new TextArea();
        Button button = new Button("Click to get text");
        button.setMinWidth(50);

        button.setOnAction(action -> {
            System.out.println(textArea.getText());
            textArea.setText("Clicked!");
        });

        VBox vbox = new VBox(textArea, button);

        Scene scene = new Scene(vbox, 200, 100);
        primaryStage.setTitle("TextArea");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



ComboBox

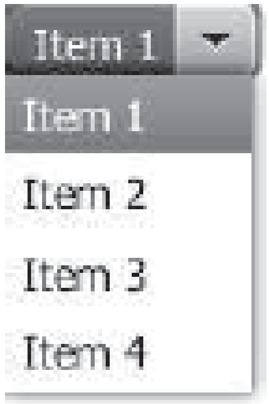
A combo box, also known as a choice list or drop-down list, contains a list of items from which the user can choose.

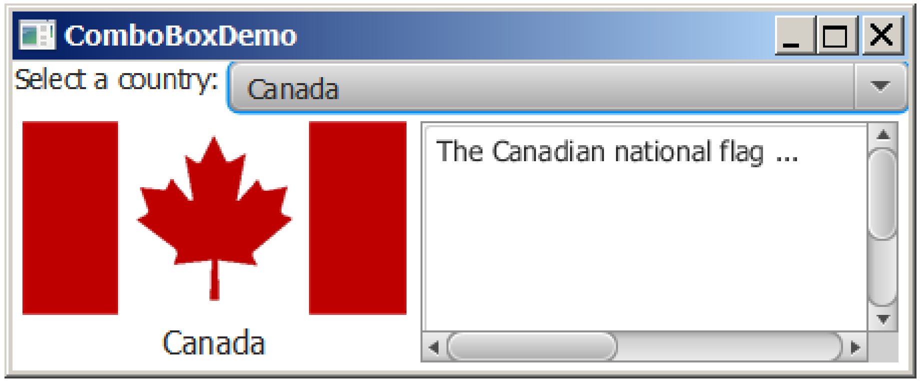

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.
javafx.scene.control.ComboBoxBase<T>
-value: ObjectProperty<T>
-editable: BooleanProperty
-onAction: ObjectProperty<EventHandler<ActionEvent>>
The value selected in the combo box.
Specifies whether the combo box allows user input.
Specifies the handler for processing the action event.
javafx.scene.control.ComboBox<T>
-items: ObjectProperty<ObservableList<T>>
-visibleRowCount: IntegerProperty
+ComboBox()
+ComboBox(items: ObservableList<T>)
The items in the combo box popup.
The maximum number of visible rows of the items in the combo box popup.
Creates an empty combo box.
Creates a combo box with the specified items.

ComboBox Example

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

```
ComboBox<String> cbo = new ComboBox<>();
cbo.getItems().addAll("Item 1", "Item 2",
  "Item 3", "Item 4");
cbo.setStyle("-fx-color: red");
cbo.setValue("Item 1");
```

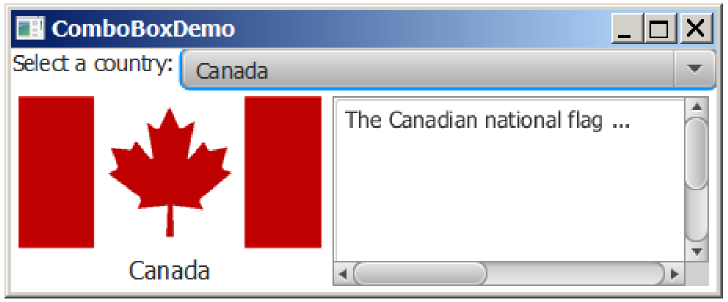




The Canadian national flag ...

ComboBox Example

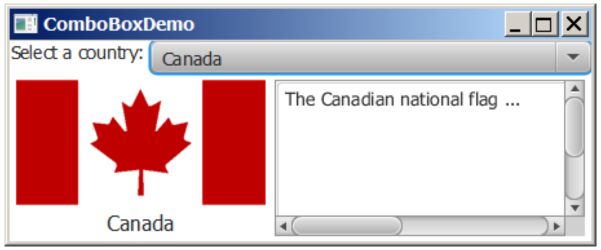
```java
public class ComboBoxDemo extends Application {
  // Declare an array of Strings for flag titles
  private String[] flagTitles = {"Canada", "China", "Denmark",
      "France", "Germany", "India", "Norway", "United Kingdom",
      "United States of America"};

  // Declare an ImageView array for the national flags of 9 countrie
  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();

```



The Canadian national flag ...

ComboBox Example

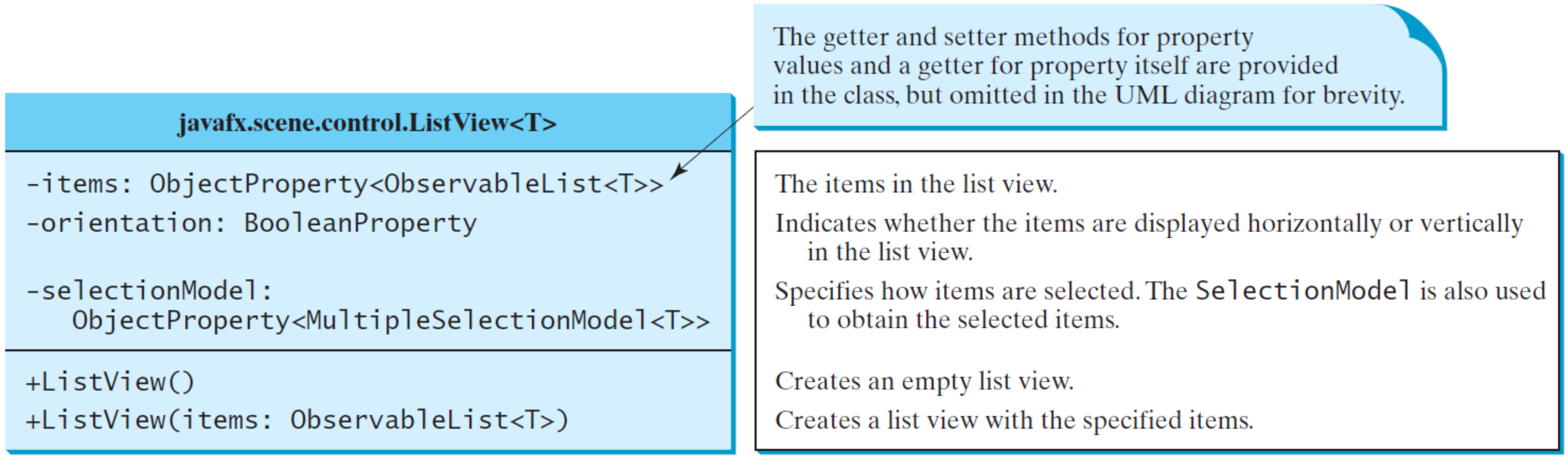
```
    BorderPane paneForComboBox = new BorderPane();
    paneForComboBox.setLeft(new Label("Select a country: "));
    paneForComboBox.setCenter(cbo);
    pane.setTop(paneForComboBox);
    cbo.setPrefWidth(400);
    cbo.setValue("Canada");

    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 Canadian national flag ...

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.



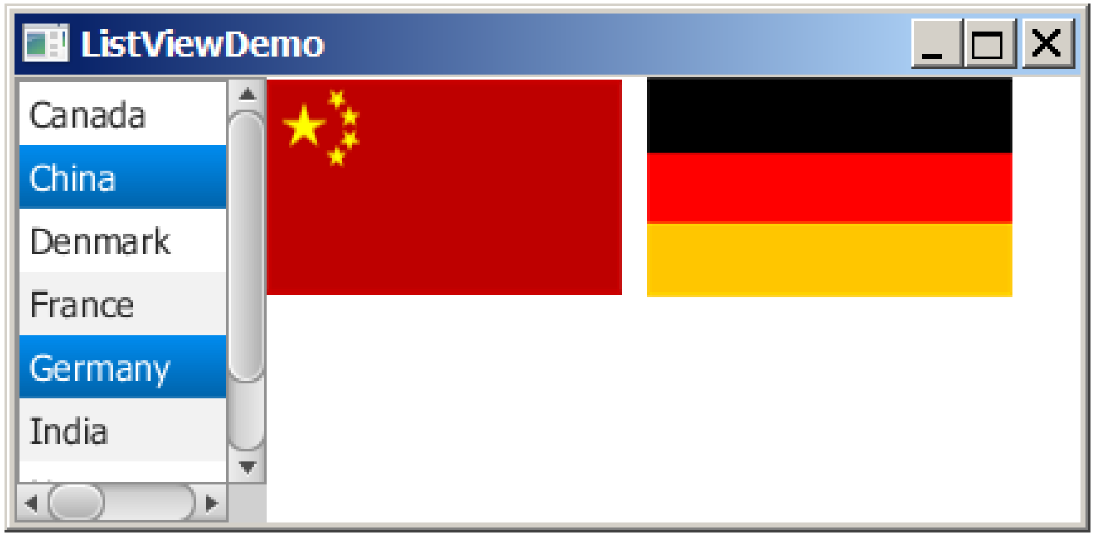
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

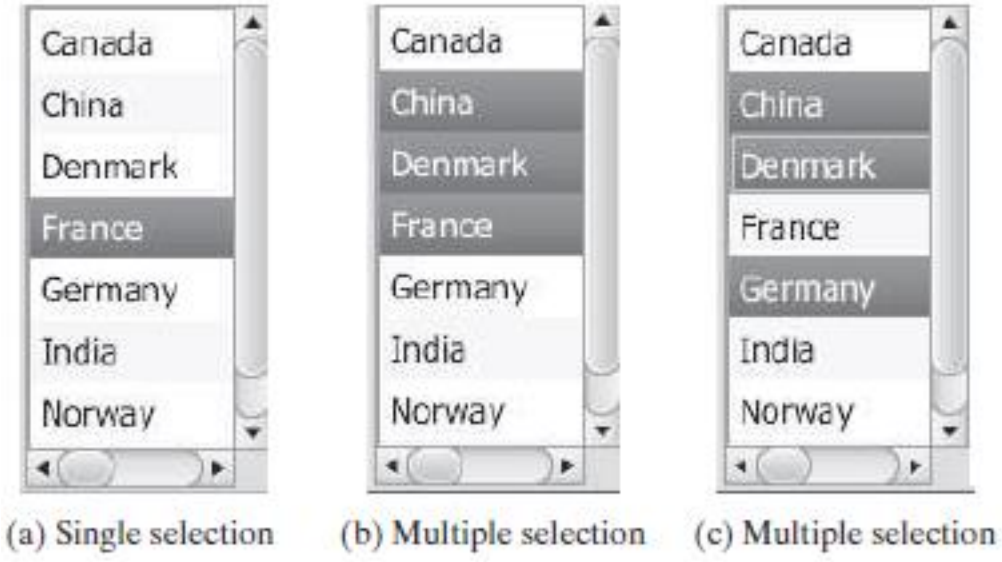
javafx.scene.control.ListView<T>

`-items: ObjectProperty<ObservableList<T>>`	The items in the list view.
`-orientation: BooleanProperty`	Indicates whether the items are displayed horizontally or vertically in the list view.
`-selectionModel: ObjectProperty<MultipleSelectionModel<T>>`	Specifies how items are selected. The `SelectionModel` is also used to obtain the selected items.
`+ListView()`	Creates an empty list view.
`+ListView(items: ObservableList<T>)`	Creates a list view with the specified items.

Example: Using ListView

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





Example: Using ListView

Create a list view of 6 items with multiple selections

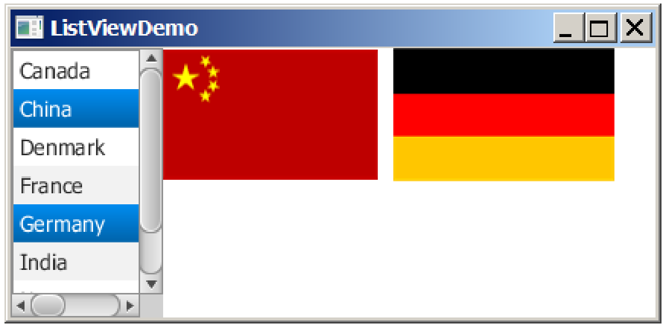
```
ObservableList<String> items =
  FXCollections.observableArrayList("Item 1", "Item 2",
    "Item 3", "Item 4", "Item 5", "Item 6");
ListView<String> lv = new ListView<>(items);
lv.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
```

A list view has **selectedItemProperty** property, and instance of **Observable**

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

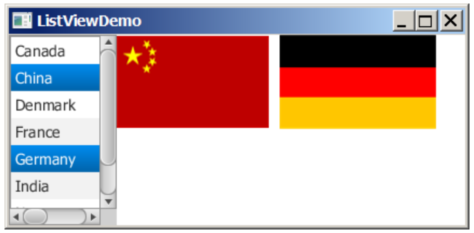
Example: Using ListView

```
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")
  };
```



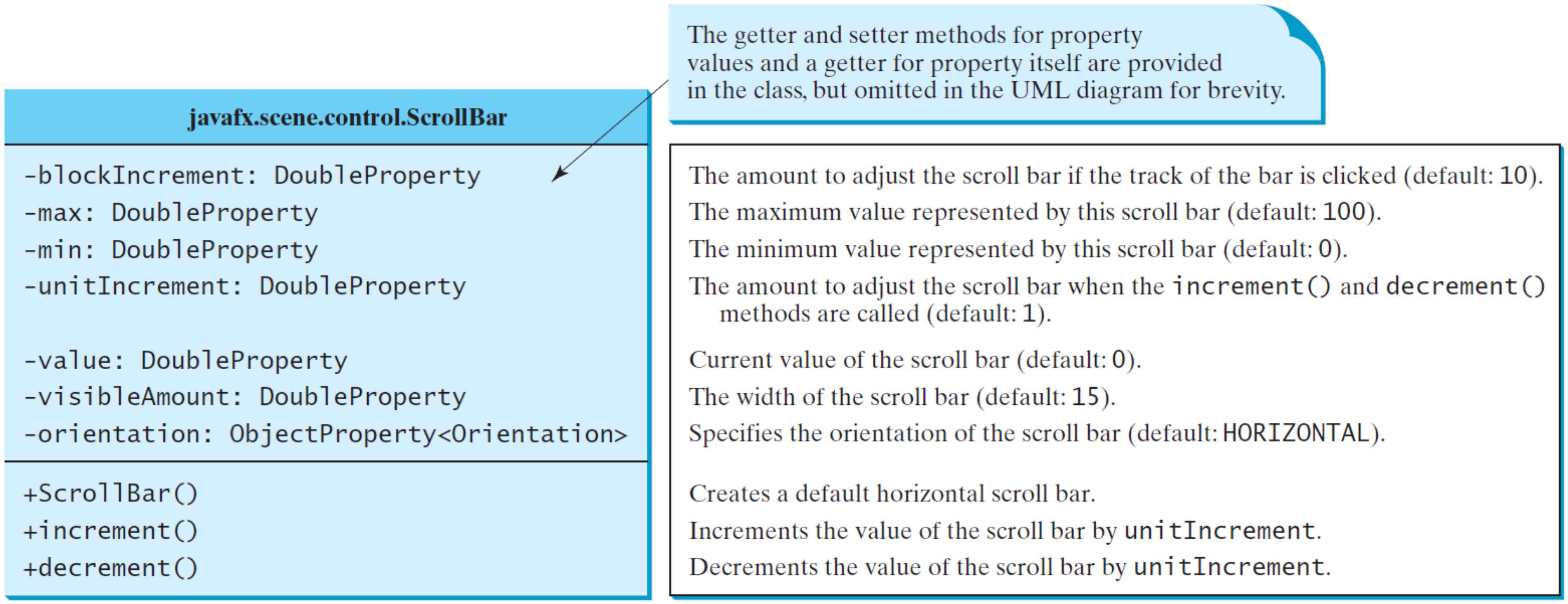
Example: Using ListView

```java

  @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
  }
}
```



ScrollBar

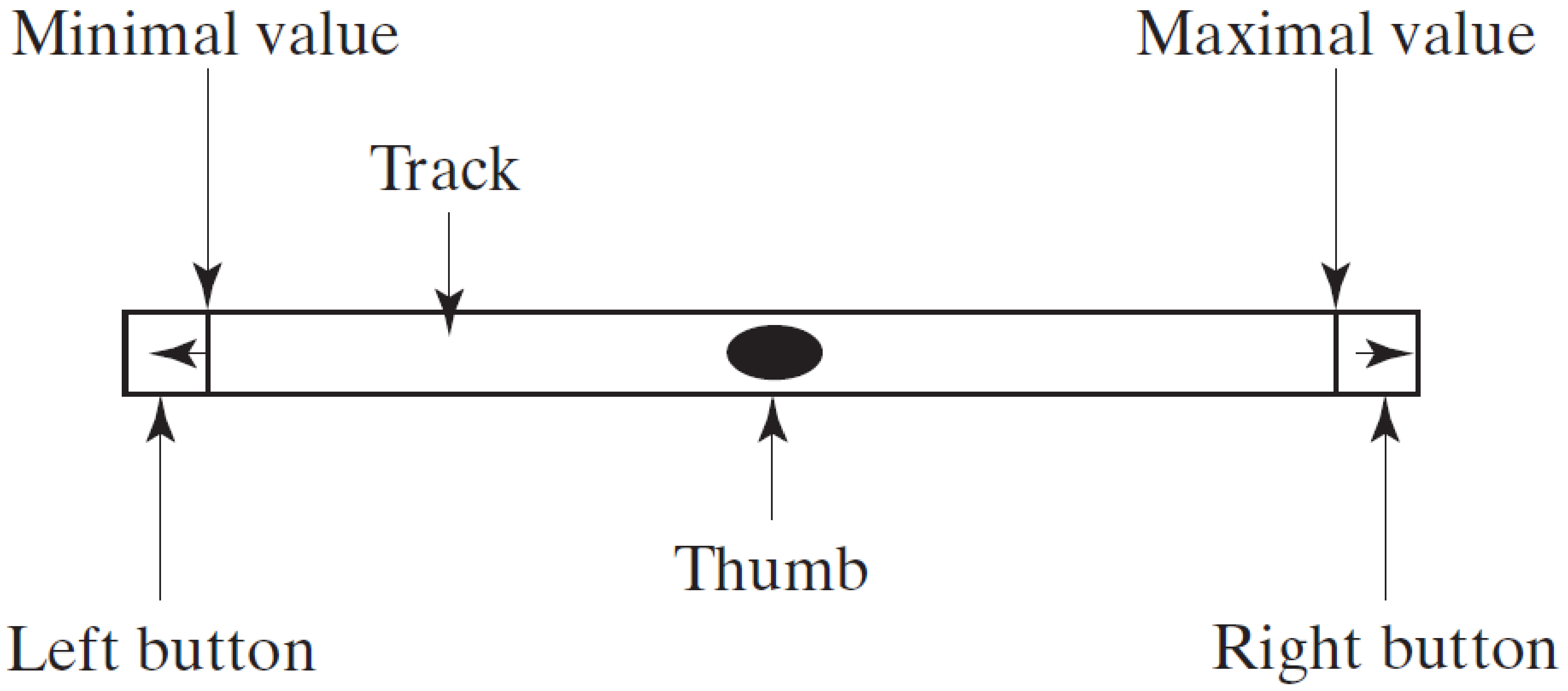
A *scroll bar* is a control that enables the user to select from a range of values. The scrollbar appears in two styles: *horizontal* and *vertical*.



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

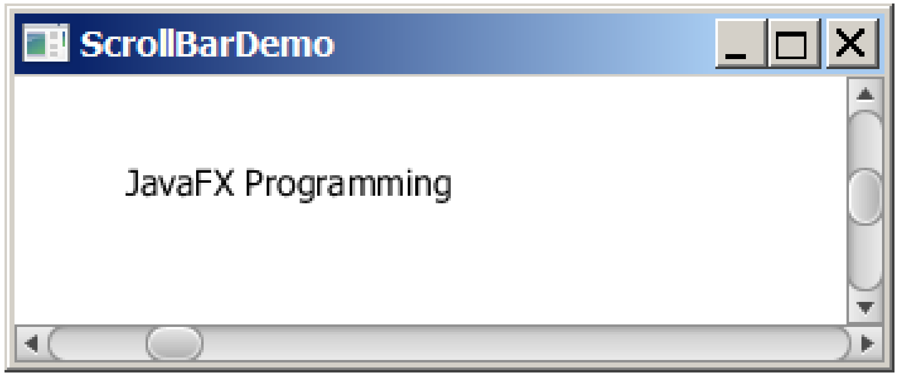
javafx.scene.control.ScrollBar	
`-blockIncrement: DoubleProperty`	The amount to adjust the scroll bar if the track of the bar is clicked (default: `10`).
`-max: DoubleProperty`	The maximum value represented by this scroll bar (default: `100`).
`-min: DoubleProperty`	The minimum value represented by this scroll bar (default: `0`).
`-unitIncrement: DoubleProperty`	The amount to adjust the scroll bar when the `increment()` and `decrement()` methods are called (default: `1`).
`-value: DoubleProperty`	Current value of the scroll bar (default: `0`).
`-visibleAmount: DoubleProperty`	The width of the scroll bar (default: `15`).
`-orientation: ObjectProperty<Orientation>`	Specifies the orientation of the scroll bar (default: `HORIZONTAL`).
`+ScrollBar()`	Creates a default horizontal scroll bar.
`+increment()`	Increments the value of the scroll bar by `unitIncrement`.
`+decrement()`	Decrements the value of the scroll bar by `unitIncrement`.

Scroll Bar Properties



Example: Using Scrollbars

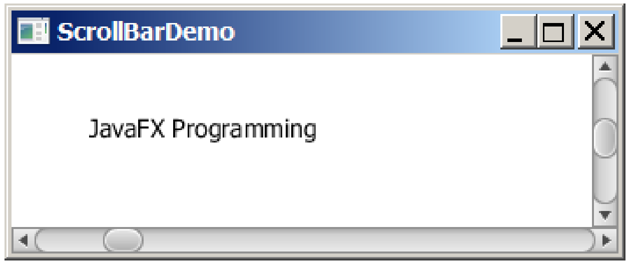
This example uses horizontal and vertical scrollbars to control a message displayed on a panel. The horizontal scrollbar is used to move the message to the left or the right, and the vertical scrollbar to move it up and down.



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

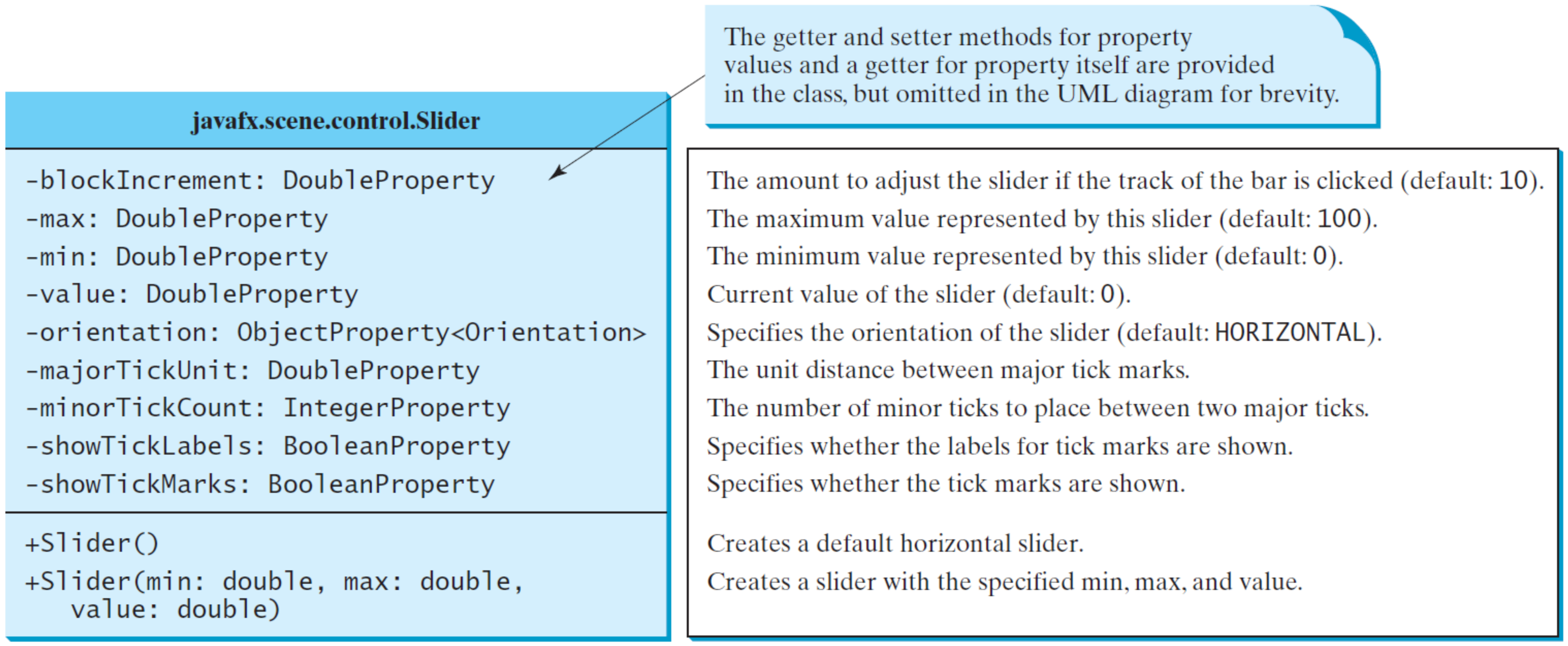
Example: Using Scrollbars

```
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
  }
}
```



Slider

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



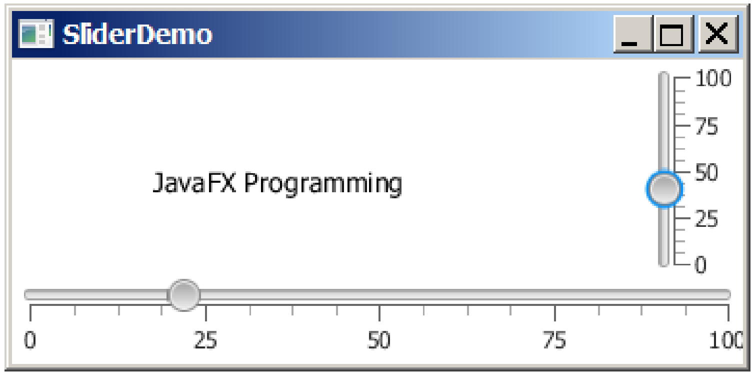
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.control.Slider

`-blockIncrement: DoubleProperty`	The amount to adjust the slider if the track of the bar is clicked (default: 10).
`-max: DoubleProperty`	The maximum value represented by this slider (default: 100).
`-min: DoubleProperty`	The minimum value represented by this slider (default: 0).
`-value: DoubleProperty`	Current value of the slider (default: 0).
`-orientation: ObjectProperty<Orientation>`	Specifies the orientation of the slider (default: HORIZONTAL).
`-majorTickUnit: DoubleProperty`	The unit distance between major tick marks.
`-minorTickCount: IntegerProperty`	The number of minor ticks to place between two major ticks.
`-showTickLabels: BooleanProperty`	Specifies whether the labels for tick marks are shown.
`-showTickMarks: BooleanProperty`	Specifies whether the tick marks are shown.
`+Slider()`	Creates a default horizontal slider.
`+Slider(min: double, max: double, value: double)`	Creates a slider with the specified min, max, and value.

Example: Using Sliders

```
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
  }
}
```



JavaFX Programming

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