

Chapter 33 - JavaFX UI Controls & Multimedia

(Lia10, C.16)

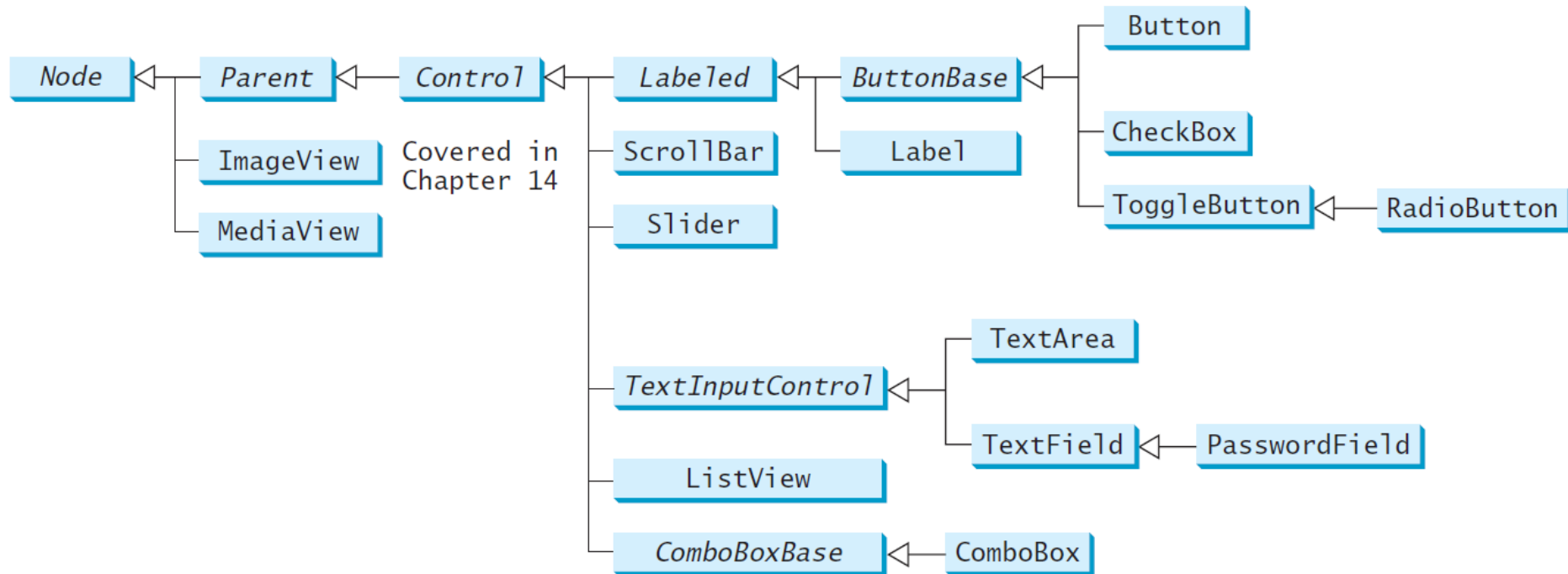


Objectives

- ☞ 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.

javaafx.scene.control.Labeled

- alignment: ObjectProperty<Pos>
- contentDisplay: ObjectProperty<ContentDisplay>
- graphic: ObjectProperty<Node>
- graphicTextGap: DoubleProperty
- textFill: ObjectProperty<Paint>
- text: StringProperty
- underline: BooleanProperty
- wrapText: BooleanProperty

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

Specifies the alignment of the text and node in the labeled.

Specifies the position of the node relative to the text using the constants TOP, BOTTOM, LEFT, and RIGHT defined in ContentDisplay.

A graphic for the labeled.

The gap between the graphic and the text.

The paint used to fill the text.

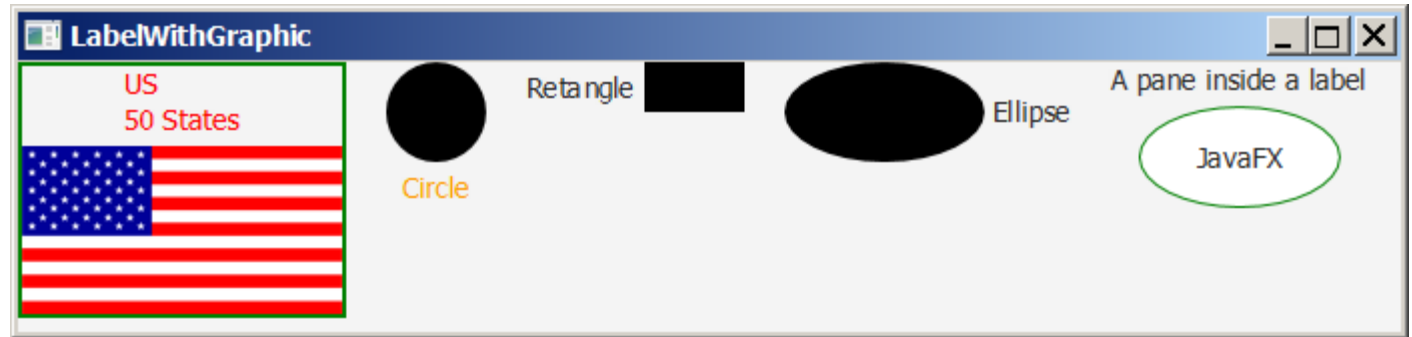
A text for the labeled.

Whether text should be underlined.

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

Label

The Label class defines labels.



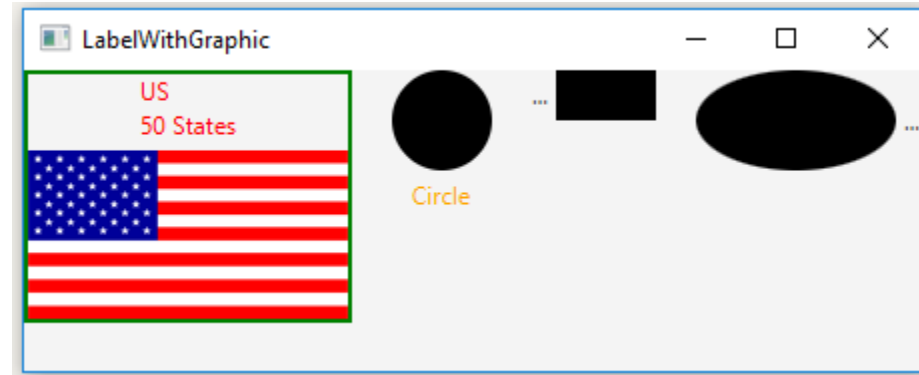
javafx.scene.control.Labeled

javafx.scene.control.Label

+Label()
+Label(text: String)
+Label(text: String, graphic: Node)

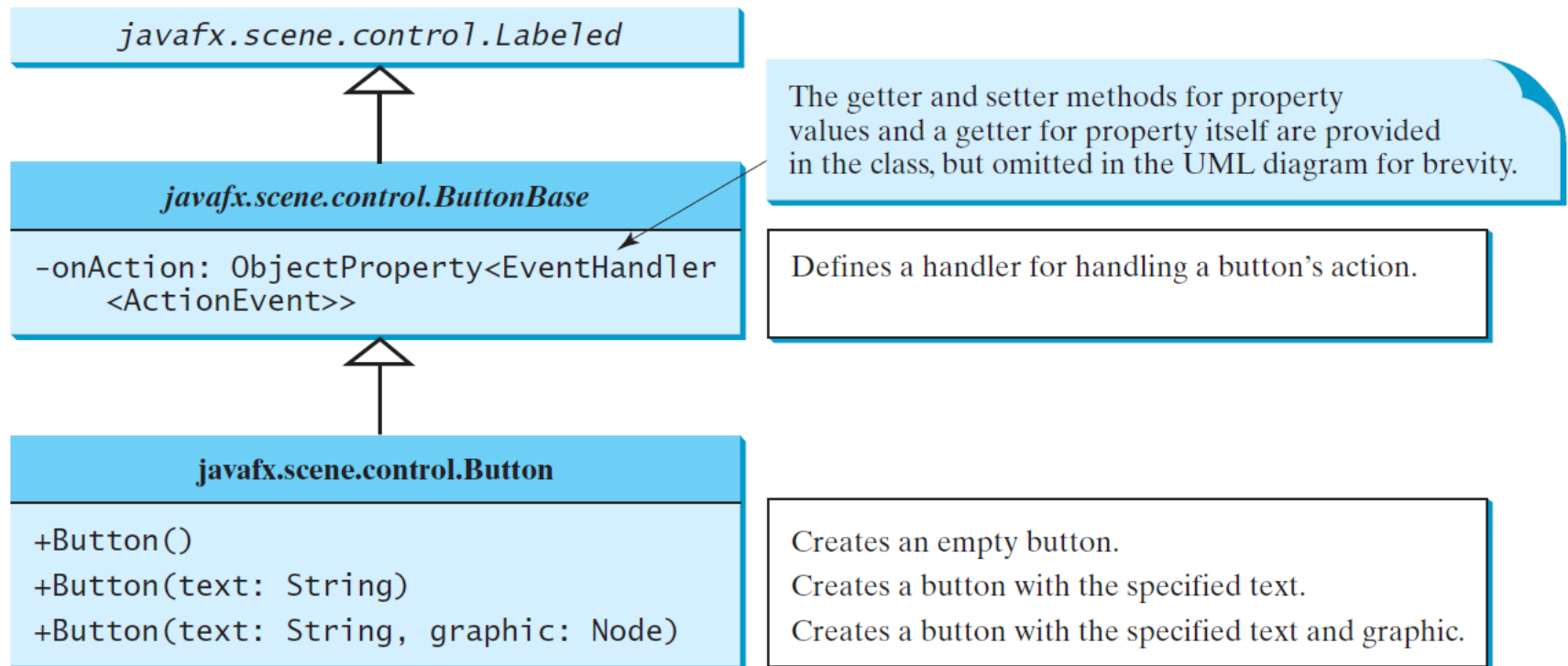
Creates an empty label.
Creates a label with the specified text.
Creates a label with the specified text and graphic.

Example LabelWithGraphic

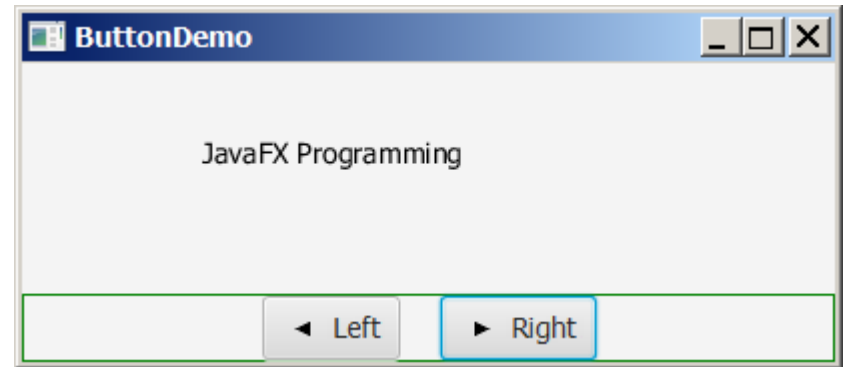


ButtonBase and Button

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.

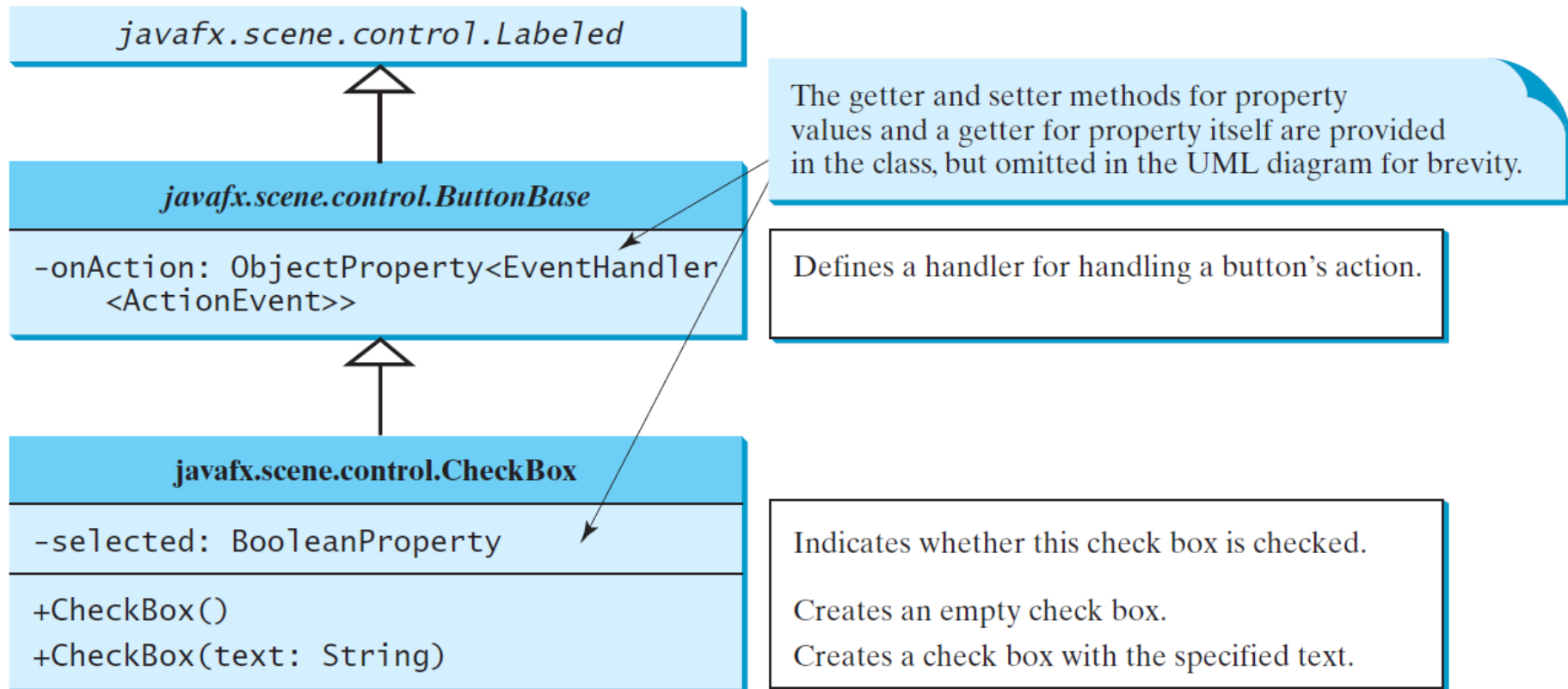


Example ButtonDemo

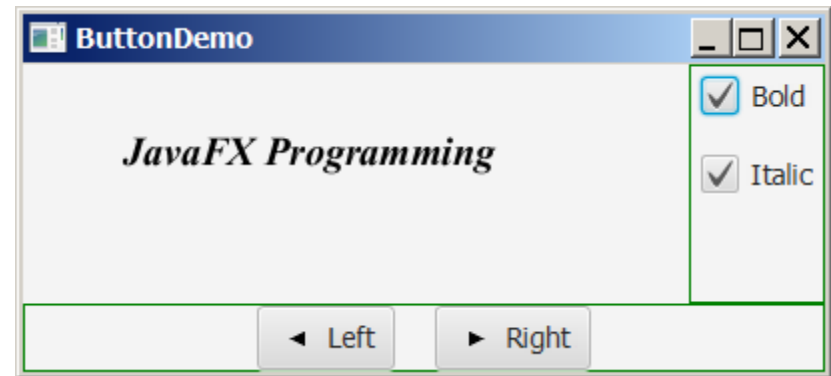


CheckBox

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**.

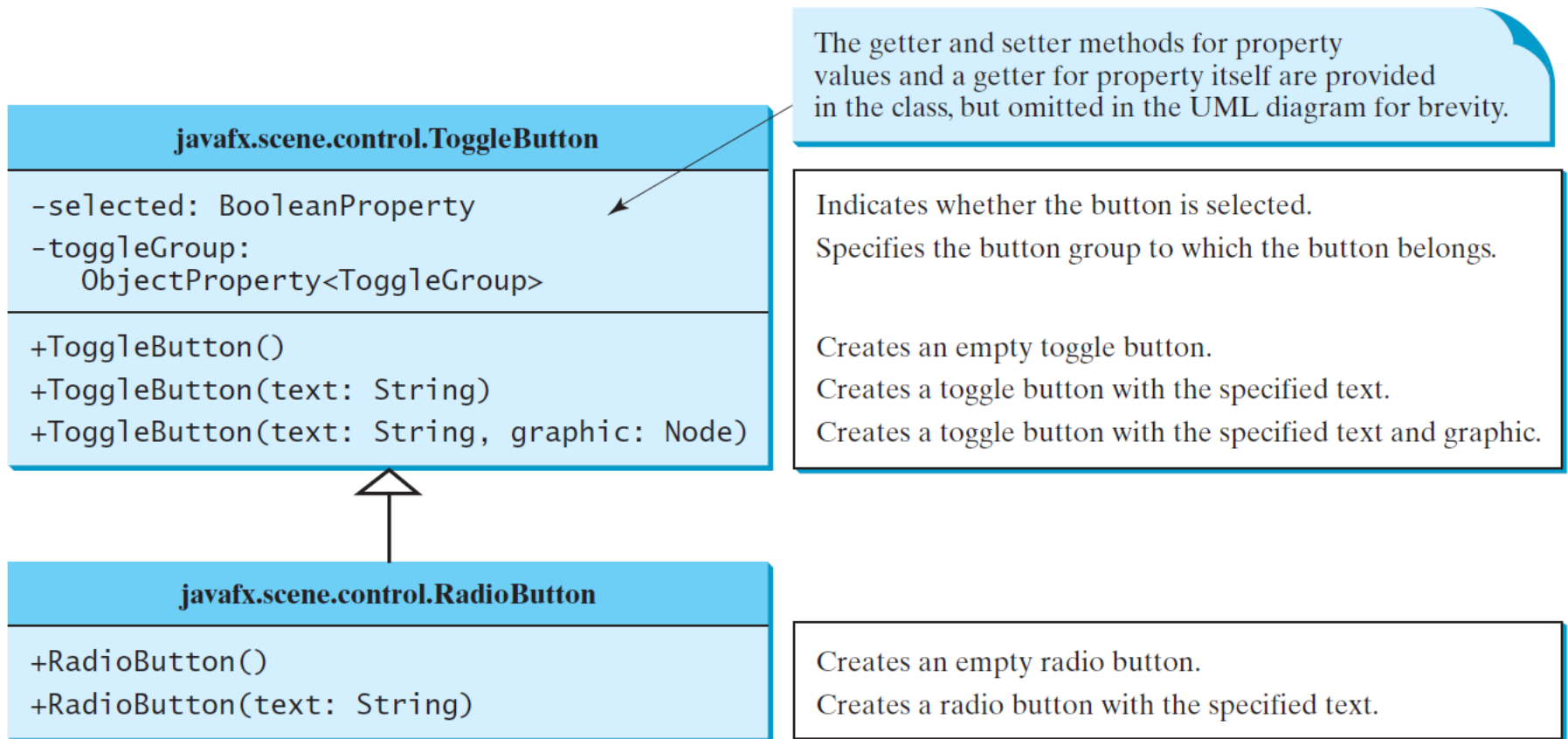


Example CheckBoxDemo

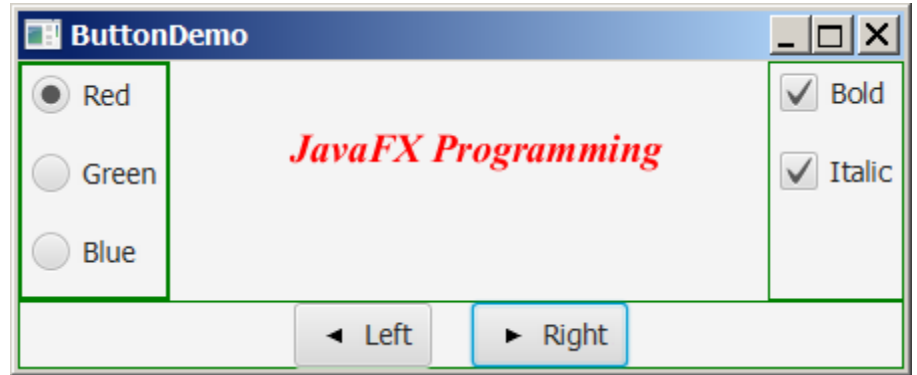


RadioButton

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

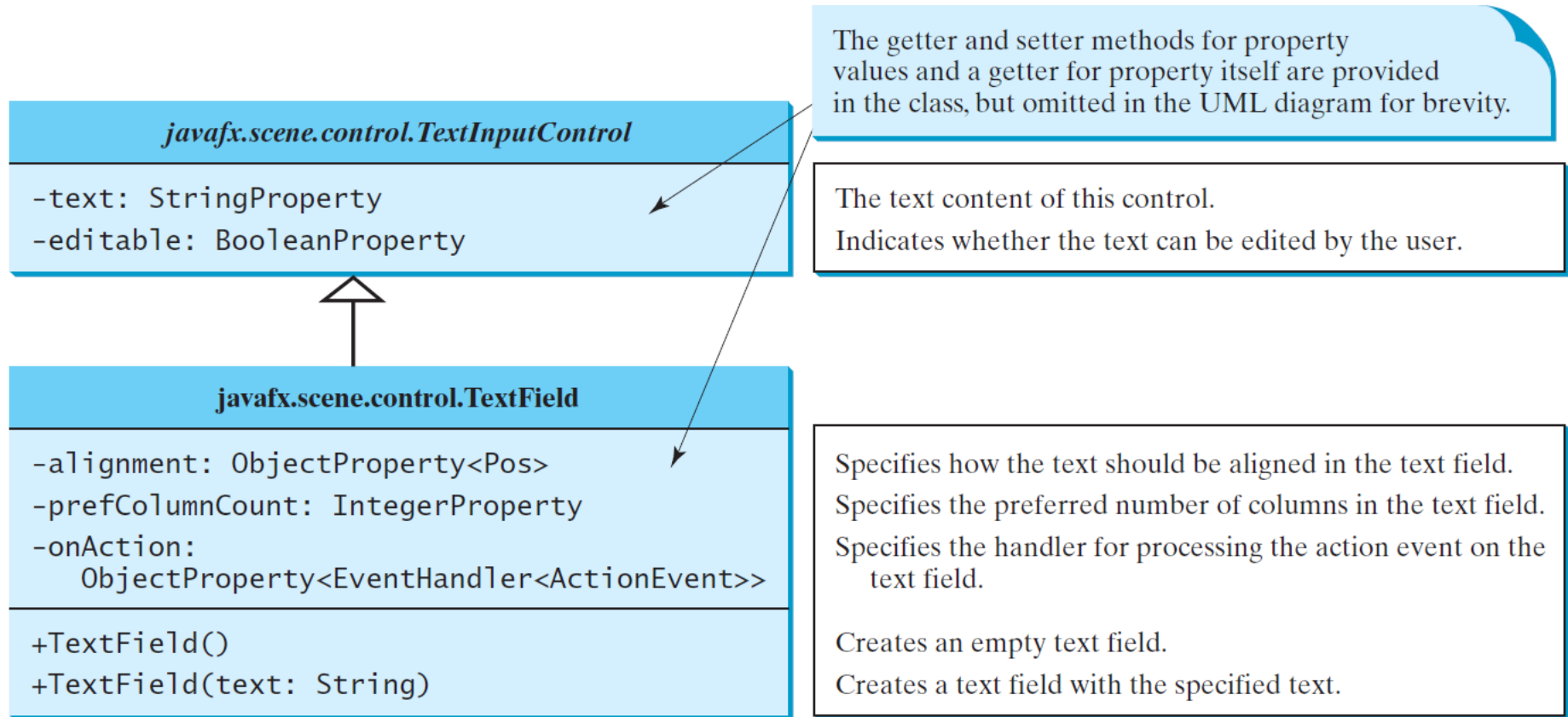


Example RadioButtonDemo



TextField

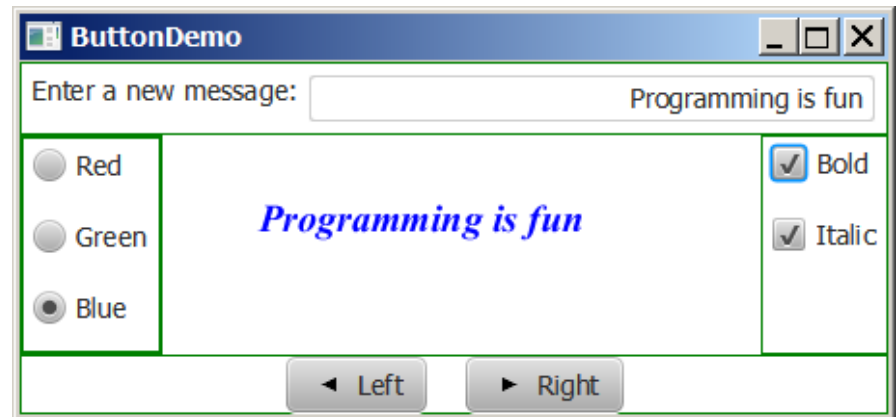
A text field can be used to enter or display a string. **TextField** is a subclass of **TextInputControl**.



TextField Example

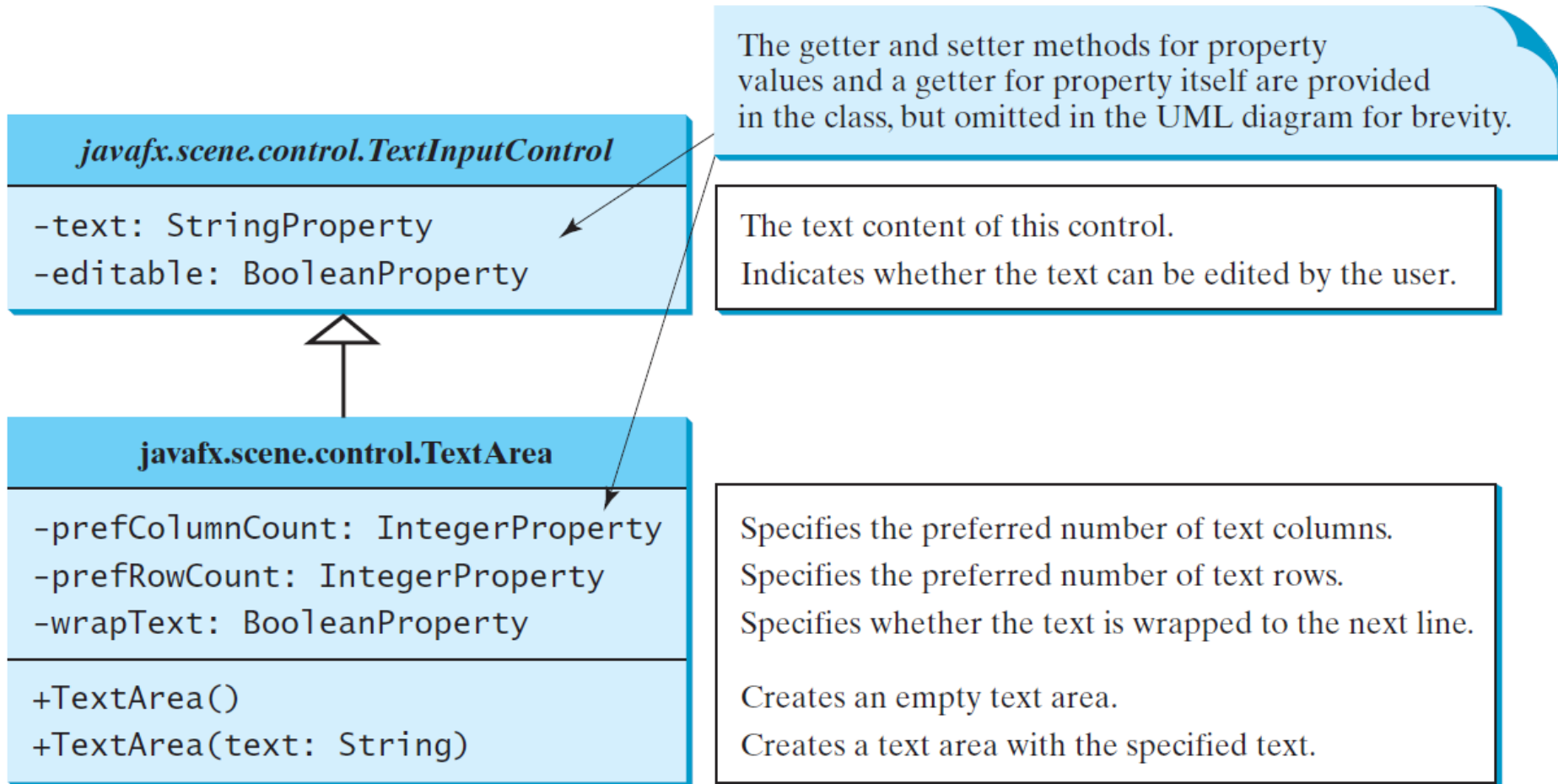


Example `TextFieldDemo`

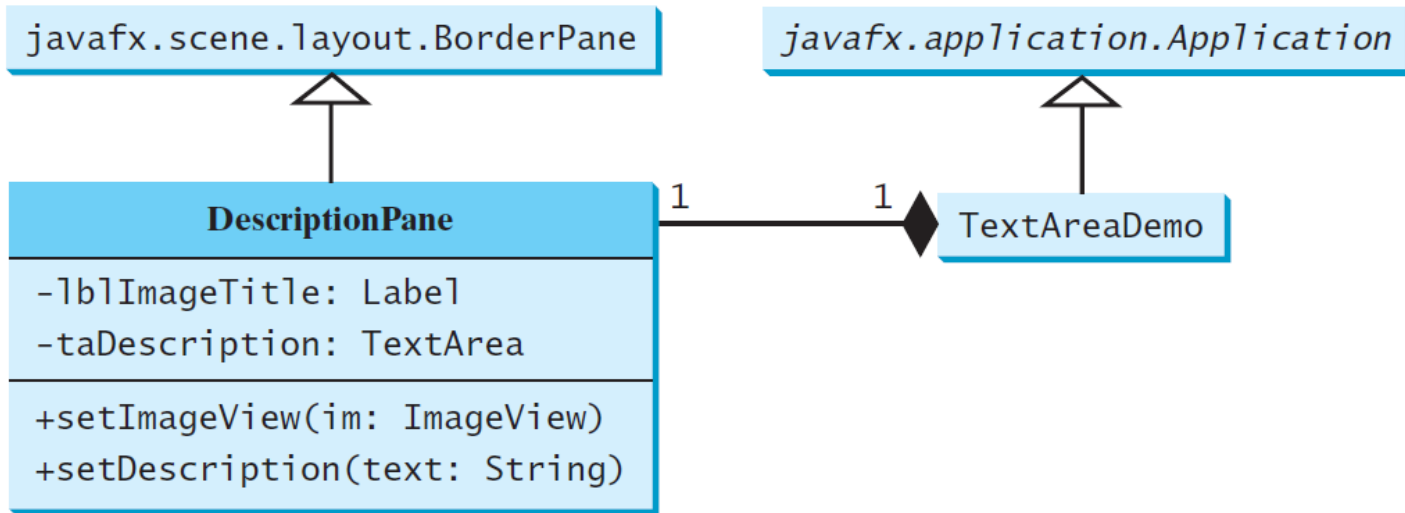


TextArea

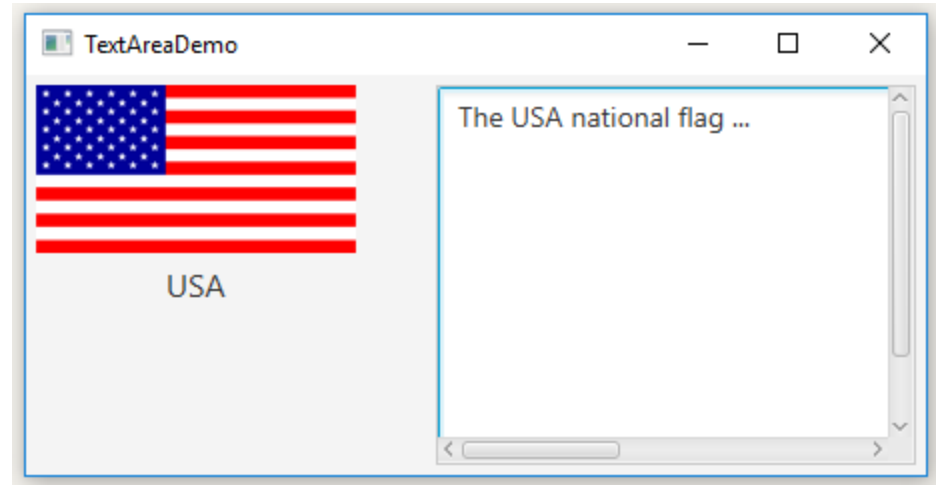
A **TextArea** enables the user to enter multiple lines of text.



TextArea Example



Example **TextAreaDemo**



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.

javafx.scene.media.Media

-duration: ReadOnlyObjectProperty
<Duration>
-width: ReadOnlyIntegerProperty
-height: ReadOnlyIntegerProperty
+Media(source: String)

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

The durations in seconds of the source media.

The width in pixels of the source video.

The height in pixels of the source video.

Creates a **Media** from a URL source.



MediaPlayer

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

javafx.scene.media.MediaPlayer

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

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

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

Specifies whether the playing should start automatically.
The number of completed playback cycles.
Specifies the number of time the media will be played.
Specifies whether the audio is muted.
The volume for the audio.
The amount of time to play the media from start to finish.

Creates a player for a specified media.
Plays the media.
Pauses the media.
Seeks the player to a new playback time.

MediaView

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

javafx.scene.media.MediaView

-x: DoubleProperty
-y: DoubleProperty
-mediaPlayer:
 ObjectProperty<MediaPlayer>
-fitWidth: DoubleProperty
-fitHeight: DoubleProperty

+MediaView()
+MediaView(mediaPlayer: MediaPlayer)

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

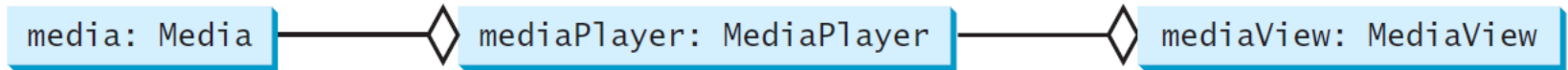
Specifies the current x-coordinate of the media view.
Specifies the current y-coordinate of the media view.
Specifies a media player for the media view.

Specifies the width of the view for the media to fit.
Specifies the height of the view for the media to fit.

Creates an empty media view.
Creates a media view with the specified media player.

Example: MediaDemo

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.



Downloads

- Sample source code:
 - [LabelWithGraphic.java](#)
 - [ButtonDemo](#)
 - [CheckBoxDemo.java](#)
 - [RadioButtonDemo.java](#)
 - [TextFieldDemo.java](#)
 - [TextAreaDemo.java](#), [DescriptionPane.java](#)
 - [MediaDemo.java](#)
- Image file: have a folder called **image** in your BlueJ project folder, and put the image files in it:
 - [us.gif](#)
 - [left.gif](#), [right.gif](#)



THE END

