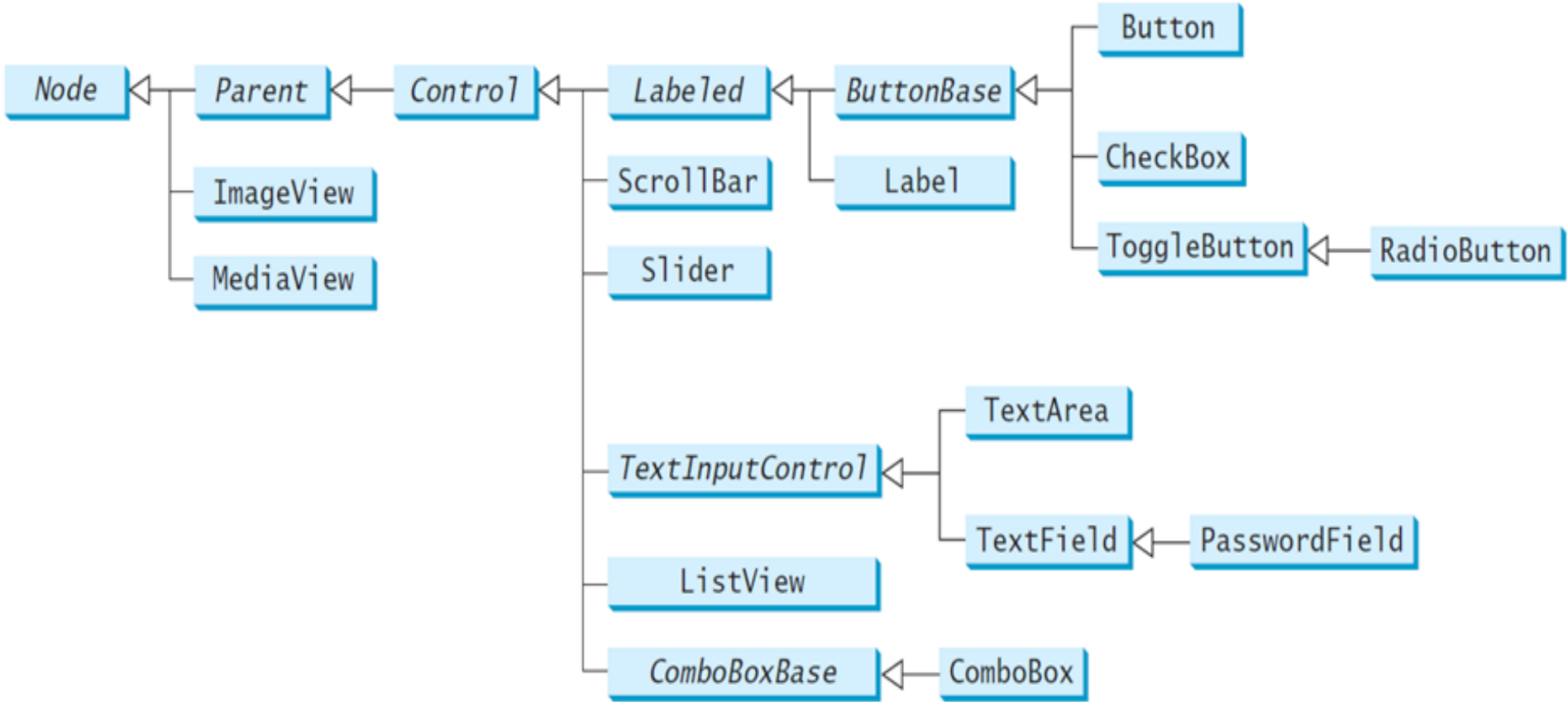
جامعة بيرزيت
BIRZEIT UNIVERSITY

JavaFX UI Controls

Frequently Used UI Controls



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

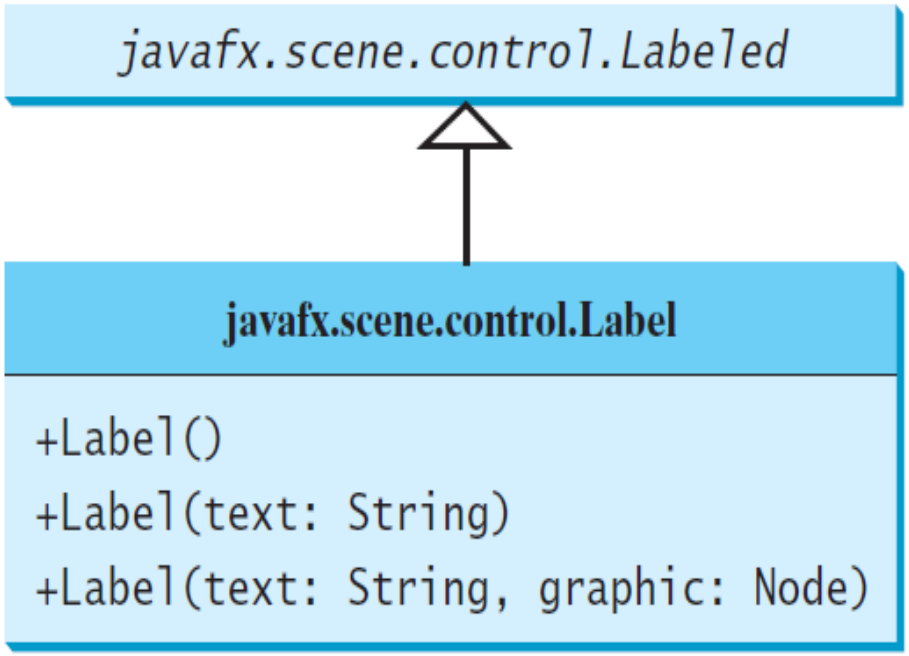
Labeled

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

javafx.scene.control.Labeled	
-alignment: ObjectProperty<Pos>	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.



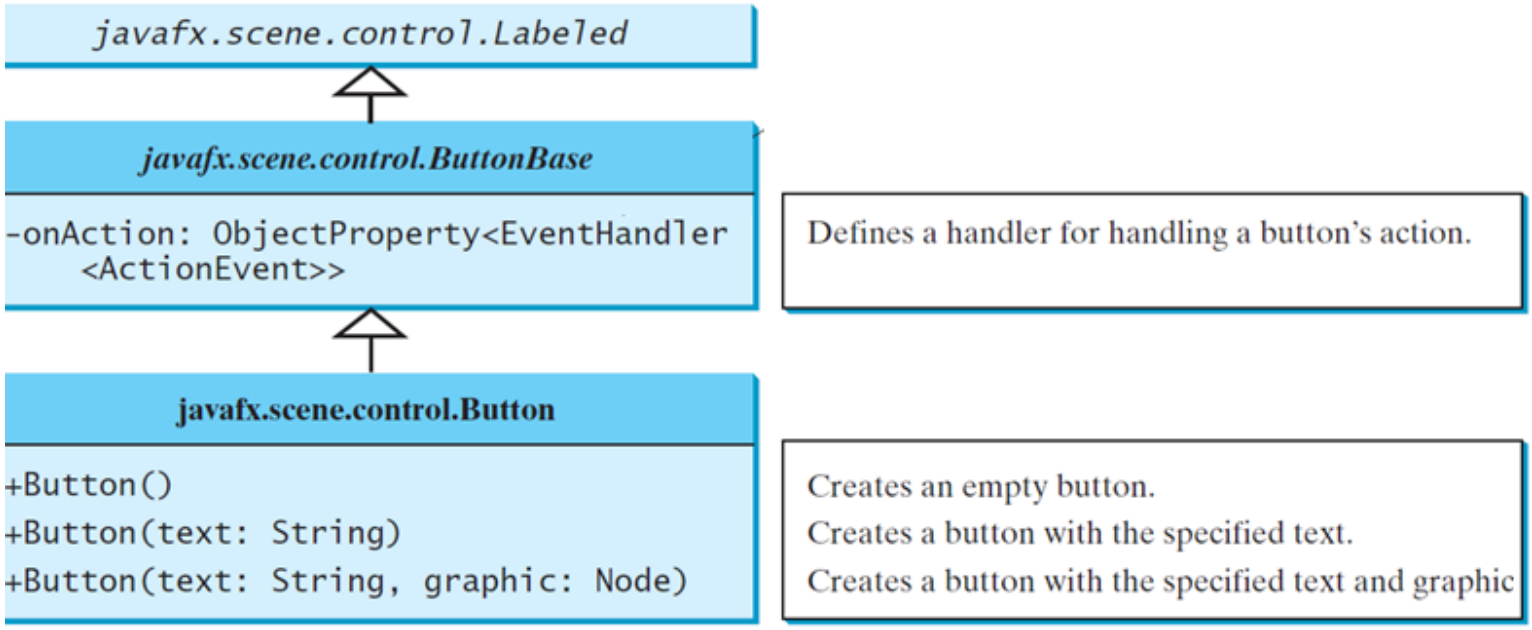
Creates an empty label.

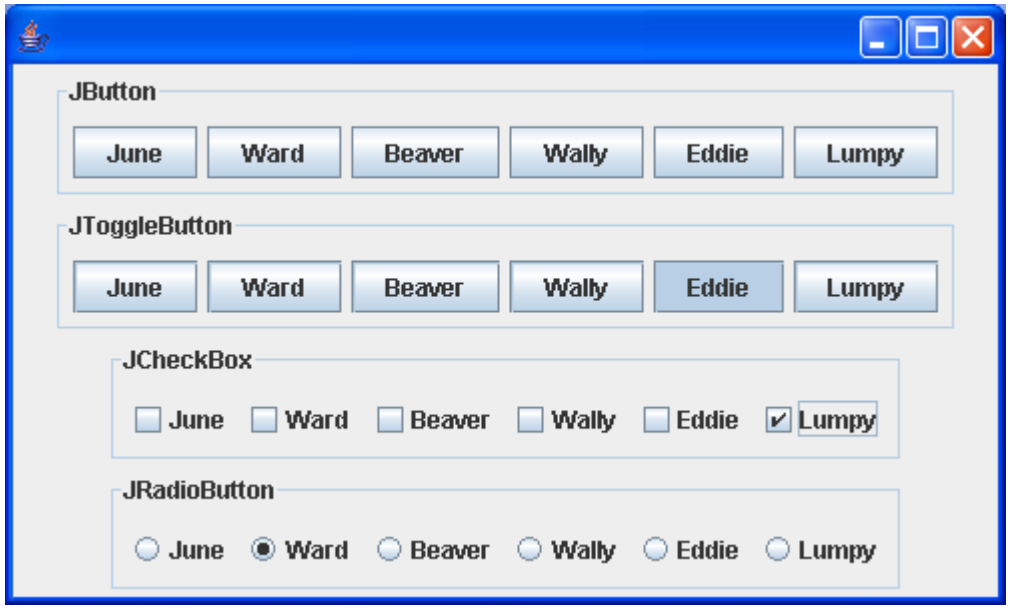
Creates a label with the specified text.

Creates a label with the specified text and graphic.

ButtonBase and Button

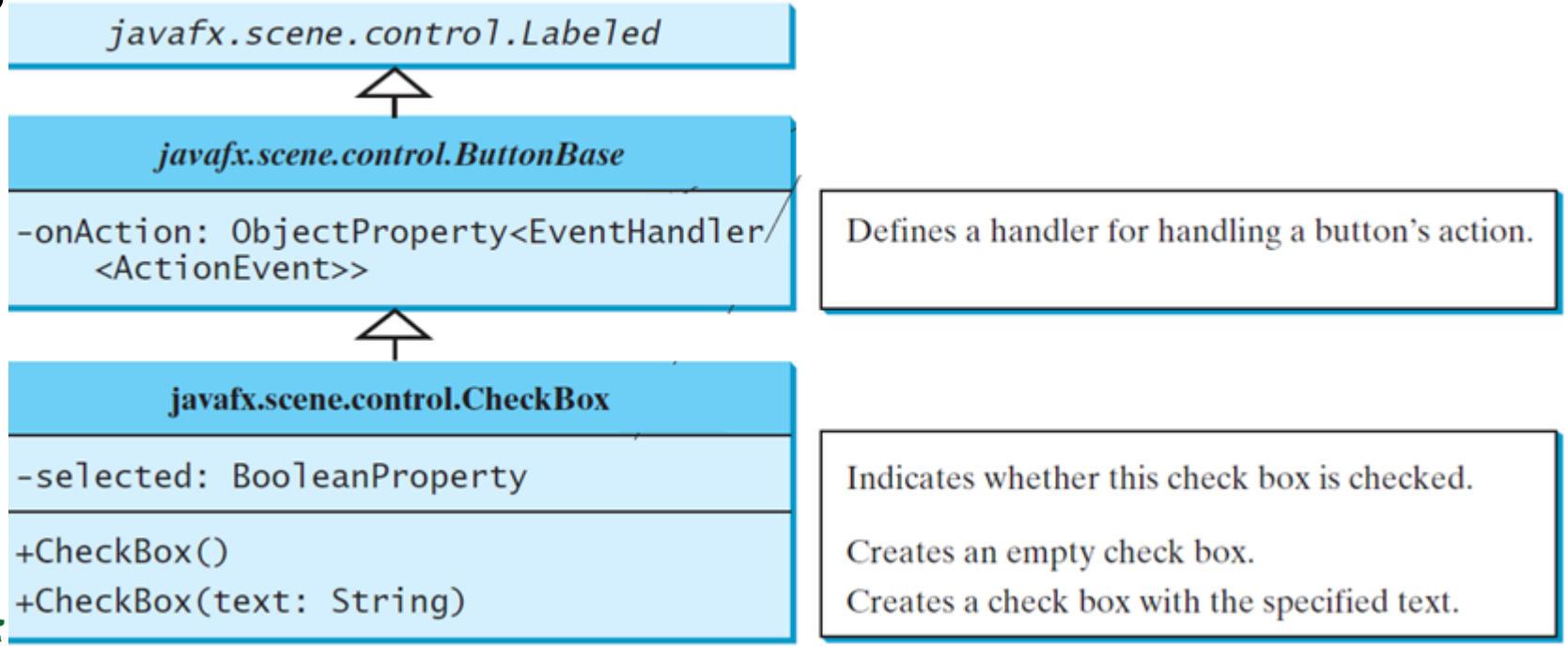
- A ***button*** is a control that triggers an action event when clicked.
- JavaFX provides regular buttons, toggle buttons, check box buttons, and radio buttons.
- The common features of these buttons are defined in **ButtonBase** and **Labeled** classes.




JButton
June
Ward
Beaver
Wally
Eddie
Lumpy
JToggleButton
June
Ward
Beaver
Wally
Eddie
Lumpy
JCheckBox
June
Ward
Beaver
Wally
Eddie
Lumpy
JRadioButton
June
Ward
Beaver
Wally
Eddie
Lumpy

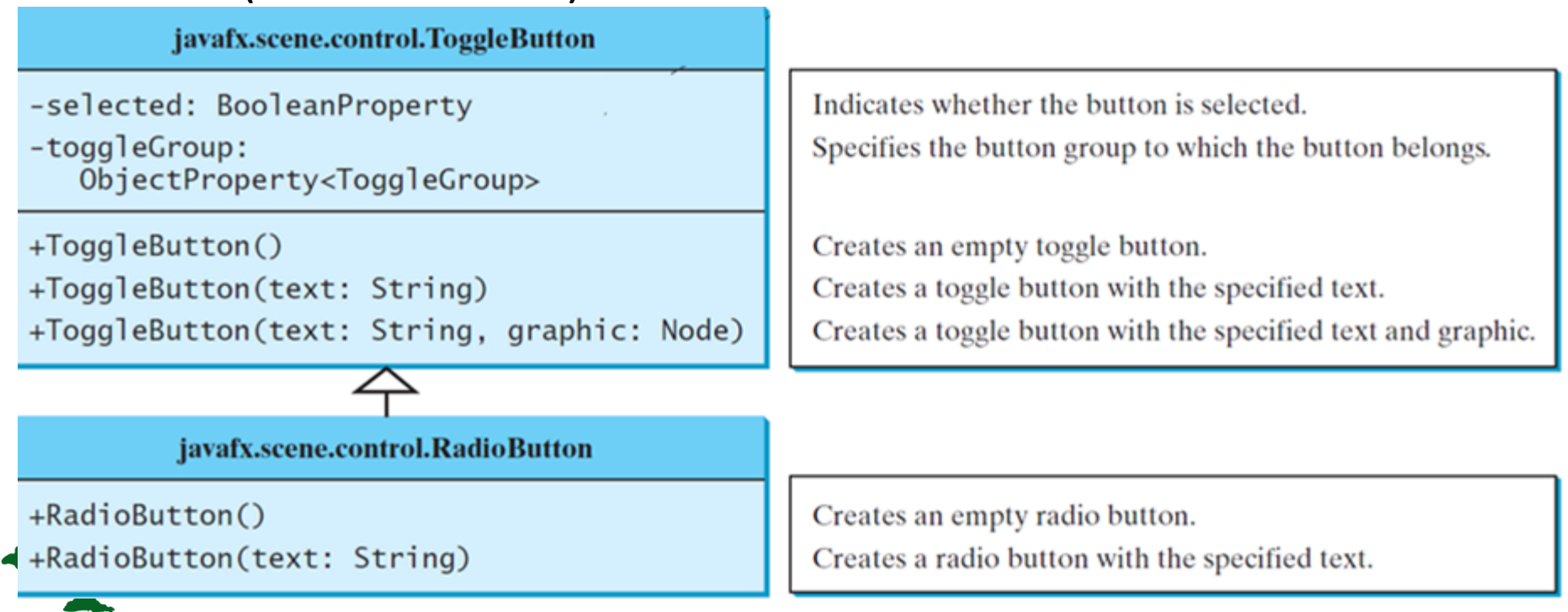
CheckBox

- 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**



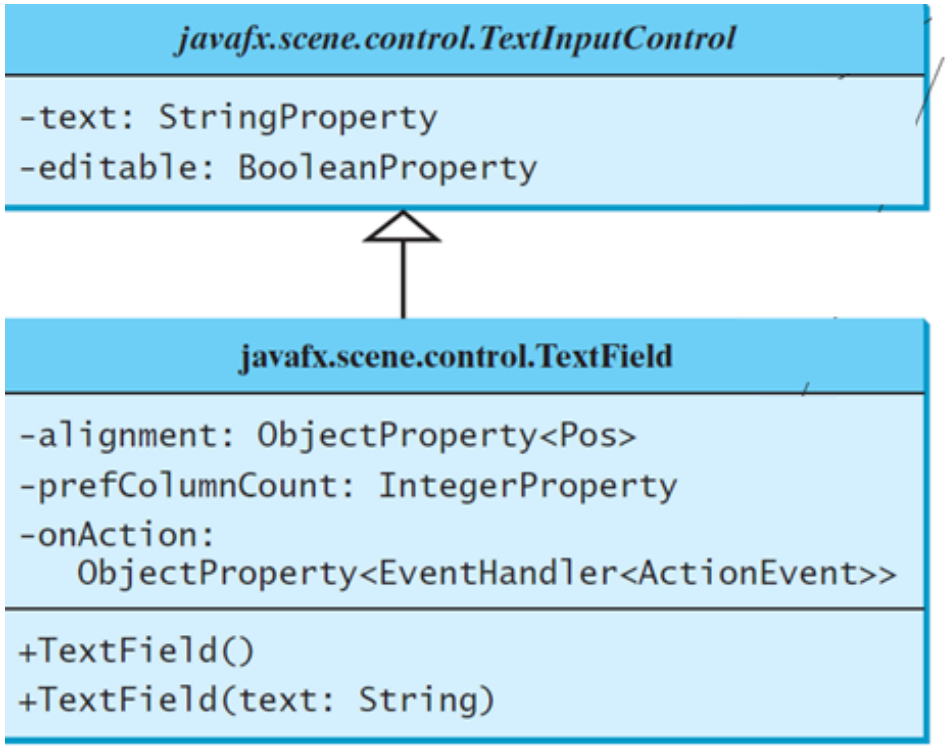
RadioButton

- 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).



TextField

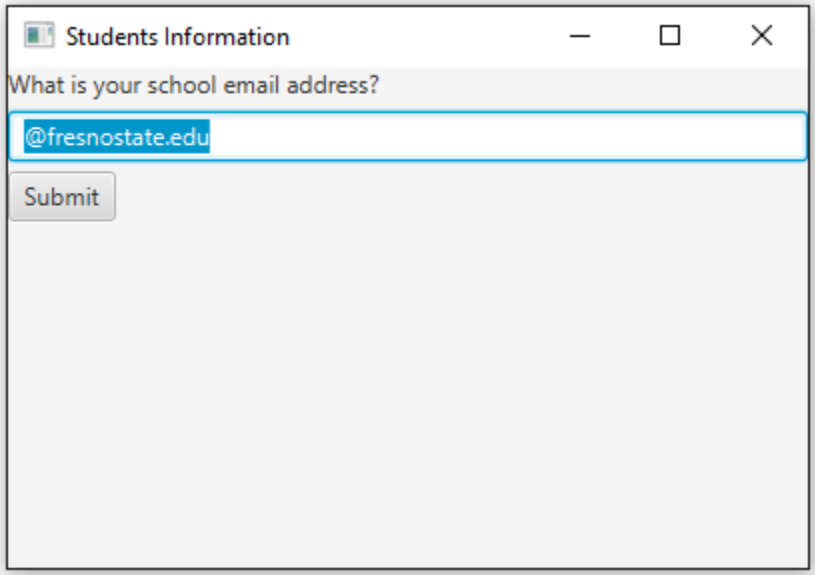
❖ A text field can be used to enter or display a string. **TextField** is a subclass of **TextInputControl**.



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

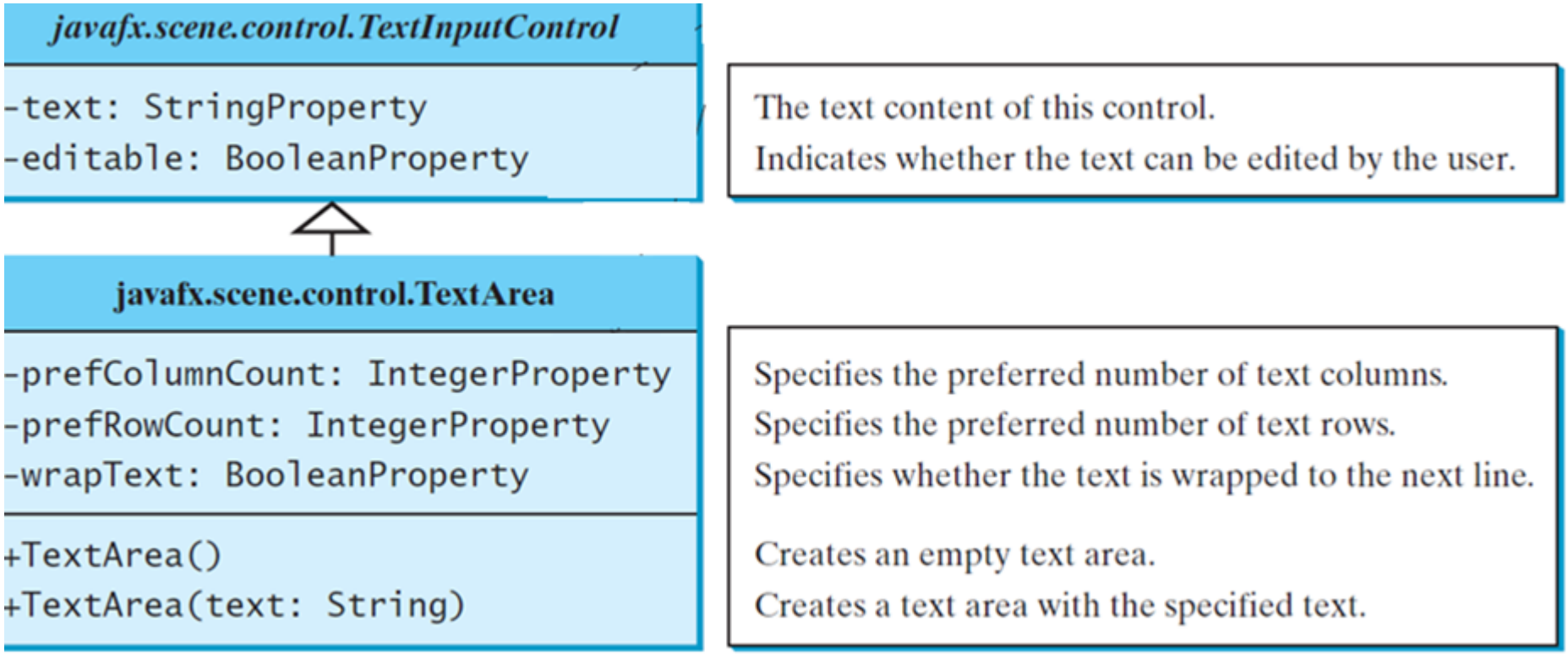
Specifies how the text should be aligned in the text field.
Specifies the preferred number of columns in the text field.
Specifies the handler for processing the action event on the text field.

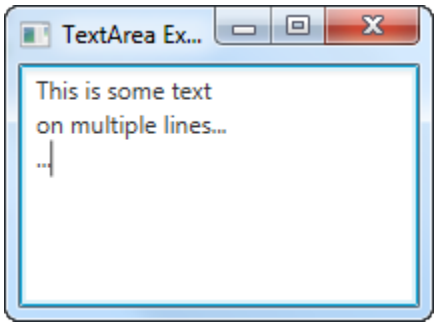
Creates an empty text field.
Creates a text field with the specified text.


Students Information
What is your school email address?
@fresnostate.edu
Submit

TextArea

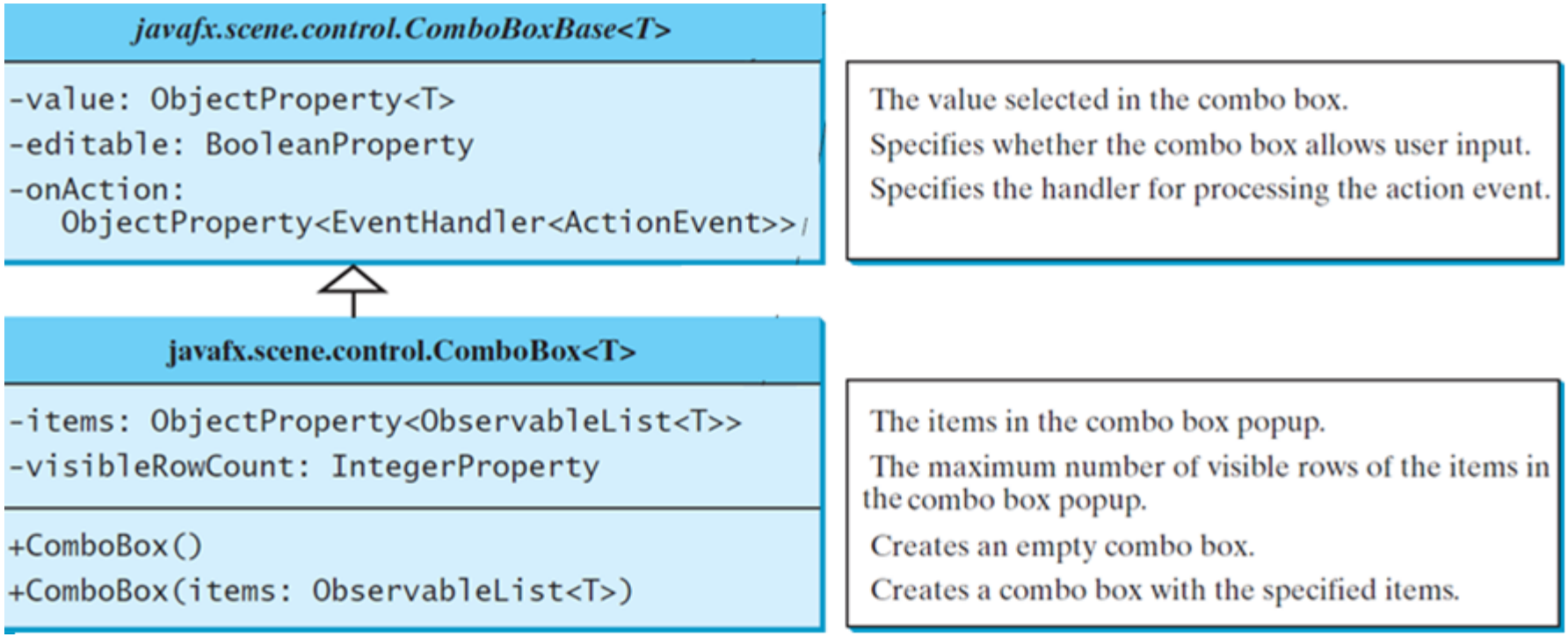
❖ A **TextArea** enables the user to enter multiple lines of text.

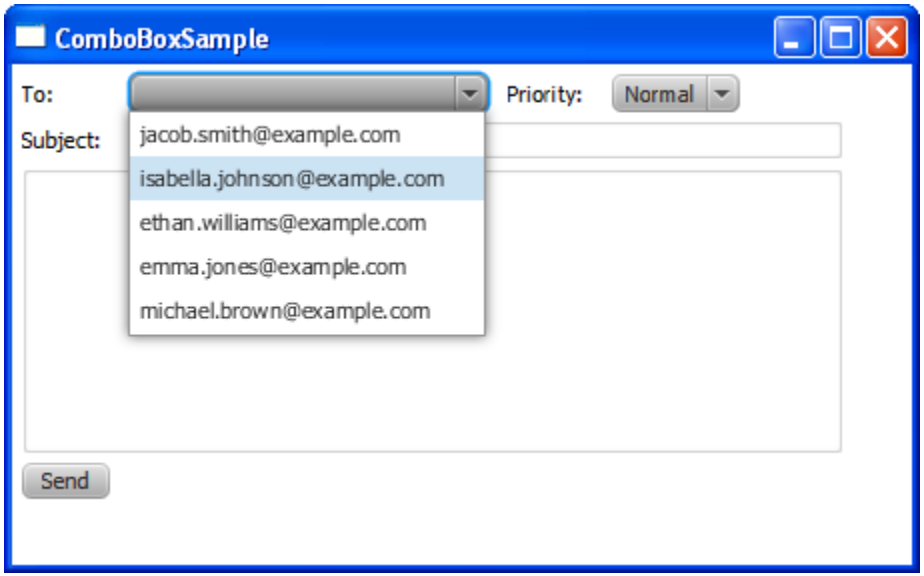



TextArea Ex...
This is some text
on multiple lines...

ComboBox

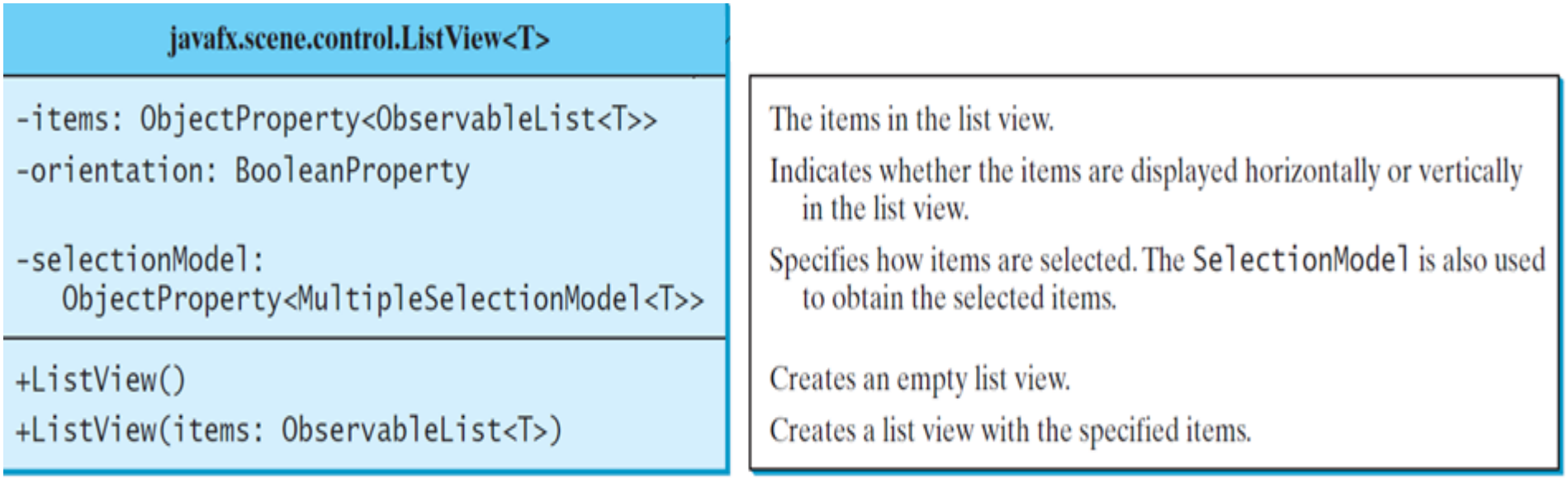
❖ A combo box, also known as a choice list or drop-down list, contains a list of items from which the user can choose.




ComboBoxSample
To:
Priority:
Normal
Subject:
jacob.smith@example.com
isabella.johnson@example.com
ethan.williams@example.com
emma.jones@example.com
michael.brown@example.com
Send

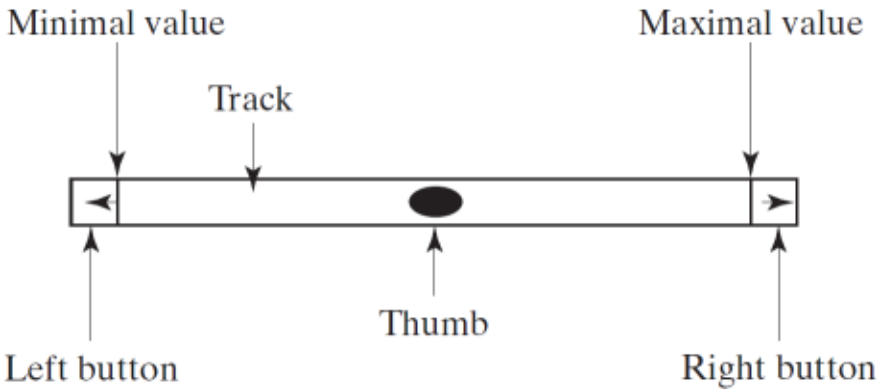
ListView

- 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.



ScrollBar

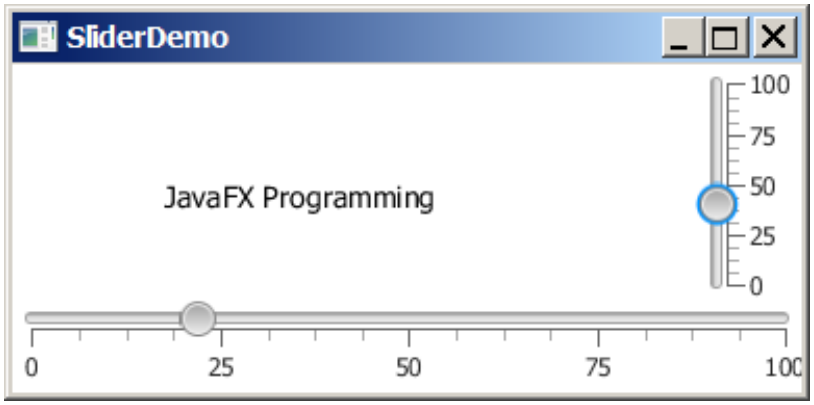
- A *scroll bar* is a control that enables the user to select from a range of values. The scrollbar appears in two styles: *horizontal* and *vertical*.



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`.

Slider

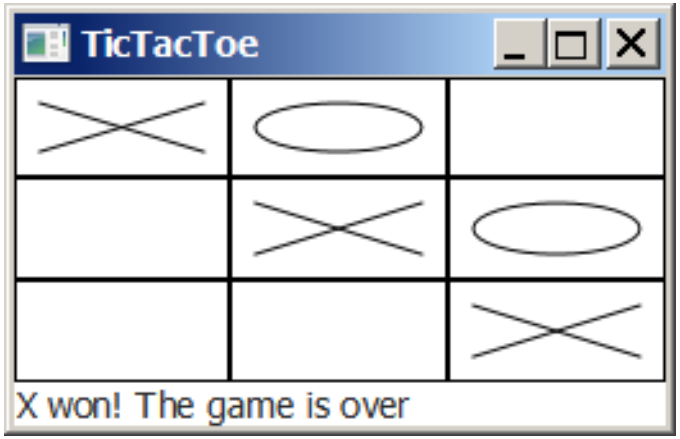
- **Slider** is similar to **ScrollBar**, but Slider has more properties and can appear in many forms.

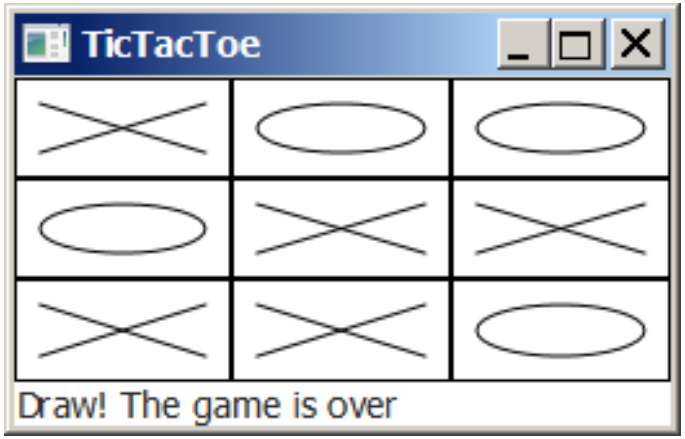


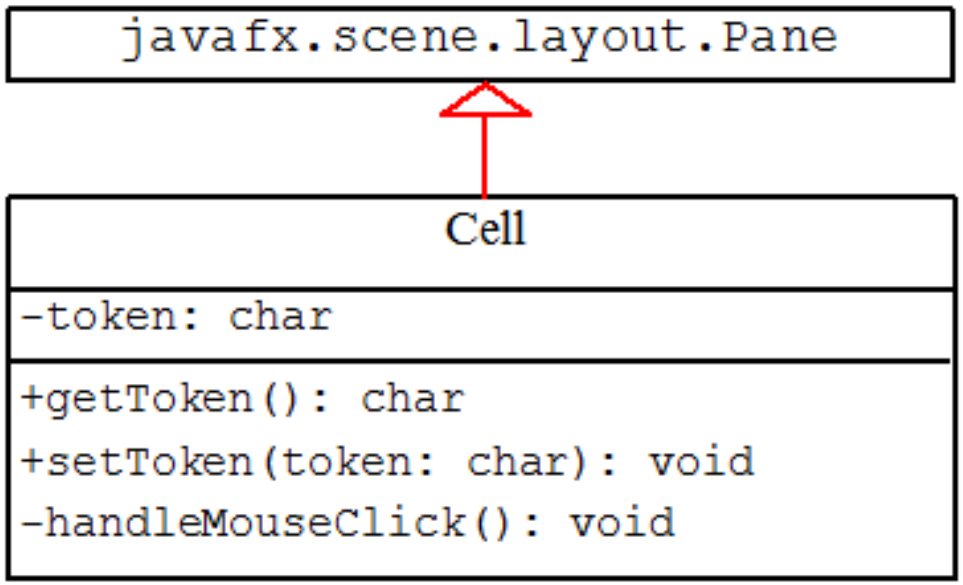
javafx.scene.control.Slider
-blockIncrement: DoubleProperty
-max: DoubleProperty
-min: DoubleProperty
-value: DoubleProperty
-orientation: ObjectProperty<Orientation>
-majorTickUnit: DoubleProperty
-minorTickCount: IntegerProperty
-showTickLabels: BooleanProperty
-showTickMarks: BooleanProperty
+Slider()
+Slider(min: double, max: double, value: double)

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

Case Study: TicTacToe







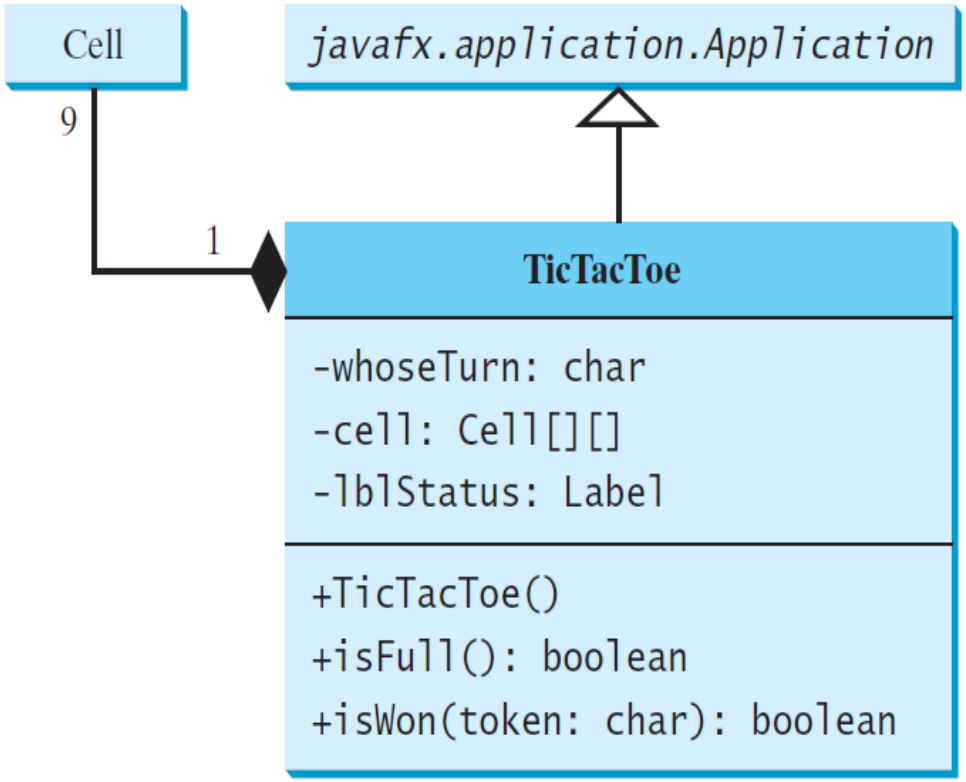
Token used in the cell (default: ' ').

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

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 supports **MP3**, **AIFF, WAV**, and MPEG-4 **audio formats** and **FLV** and **MPEG-4 video formats**

javafx.scene.media.Media	
-duration: ReadOnlyObjectProperty<Duration>	The durations in seconds of the source media.
-width: ReadOnlyIntegerProperty	The width in pixels of the source video.
-height: ReadOnlyIntegerProperty	The height in pixels of the source video.
+Media(source: String)	Creates a Media from a URL source.

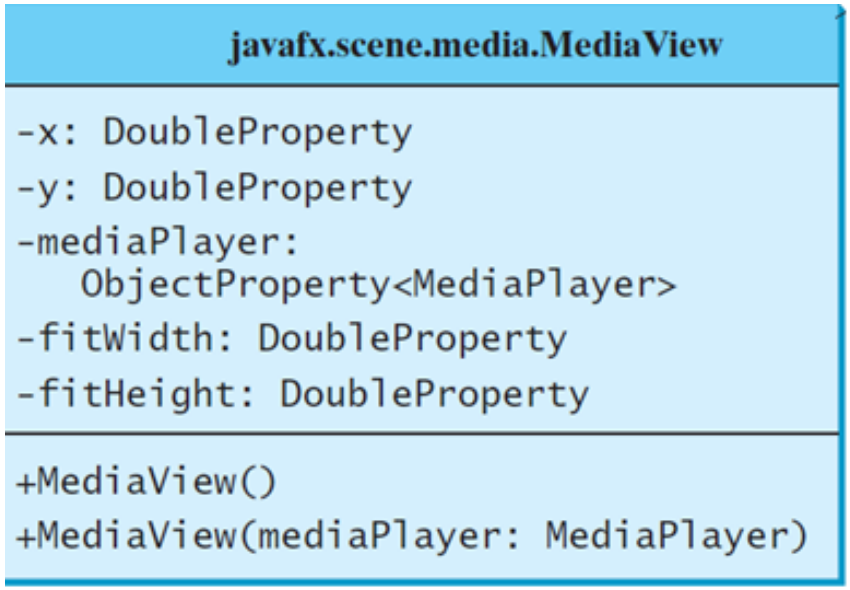
MediaPlayer

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

javafx.scene.media.MediaPlayer	
-autoPlay: BooleanProperty	Specifies whether the playing should start automatically.
-currentCount: ReadOnlyIntegerProperty	The number of completed playback cycles.
-cycleCount: IntegerProperty	Specifies the number of time the media will be played.
-mute: BooleanProperty	Specifies whether the audio is muted.
-volume: DoubleProperty	The volume for the audio.
-totalDuration: ReadOnlyObjectProperty<Duration>	The amount of time to play the media from start to finish.
+MediaPlayer(media: Media)	Creates a player for a specified media.
+play(): void	Plays the media.
+pause(): void	Pauses the media.
+seek(): void	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.



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.