JavaFX UI Controls
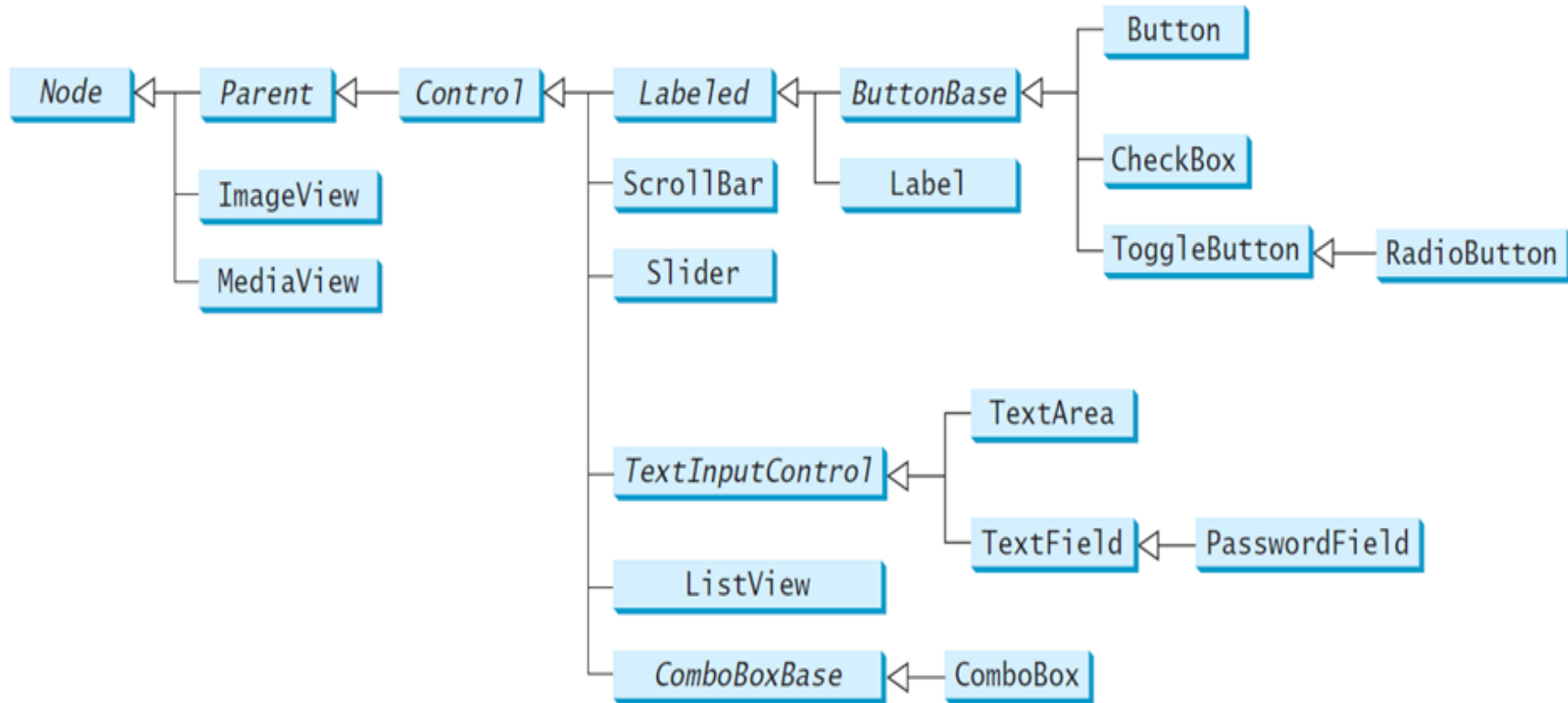
Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All

Uploaded By: Jibreel Bornat

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**.

 2

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.


```
javafx.scene.control.Labeled
```

```
javafx.scene.control.Label

+Label()                             Creates an empty label.
+Label(text: String)                 Creates a label with the specified text.
+Label(text: String, graphic: Node)  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.

```
javafx.scene.control.Labeled
```

```
javafx.scene.control.ButtonBase

-onAction: ObjectProperty<EventHandler
    <ActionEvent>>
```
Defines a handler for handling a button's action.

```
javafx.scene.control.Button

+Button()
+Button(text: String)
+Button(text: String, graphic: Node)
```
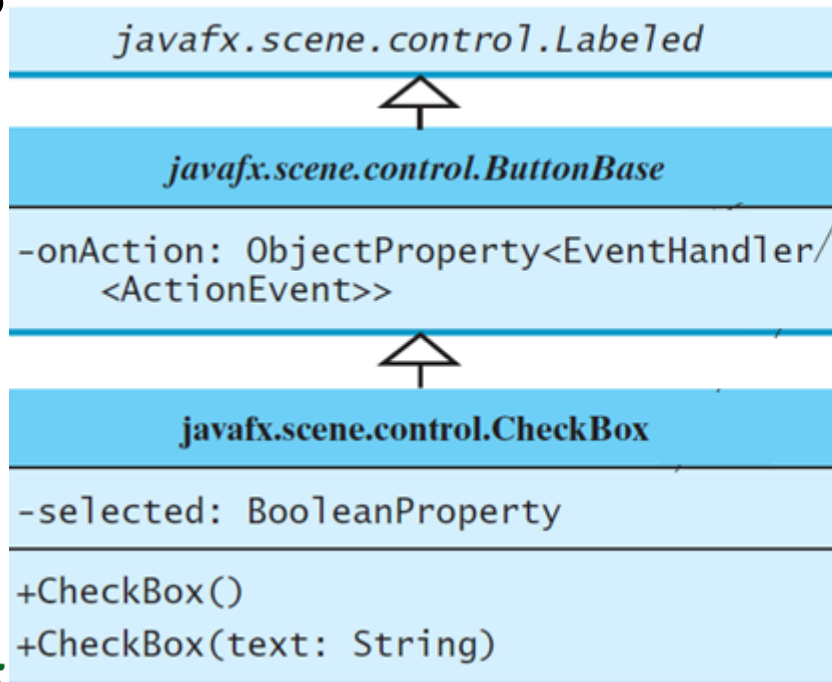Creates an empty button.
Creates a button with the specified text.
Creates a button with the specified text and graphic.

5
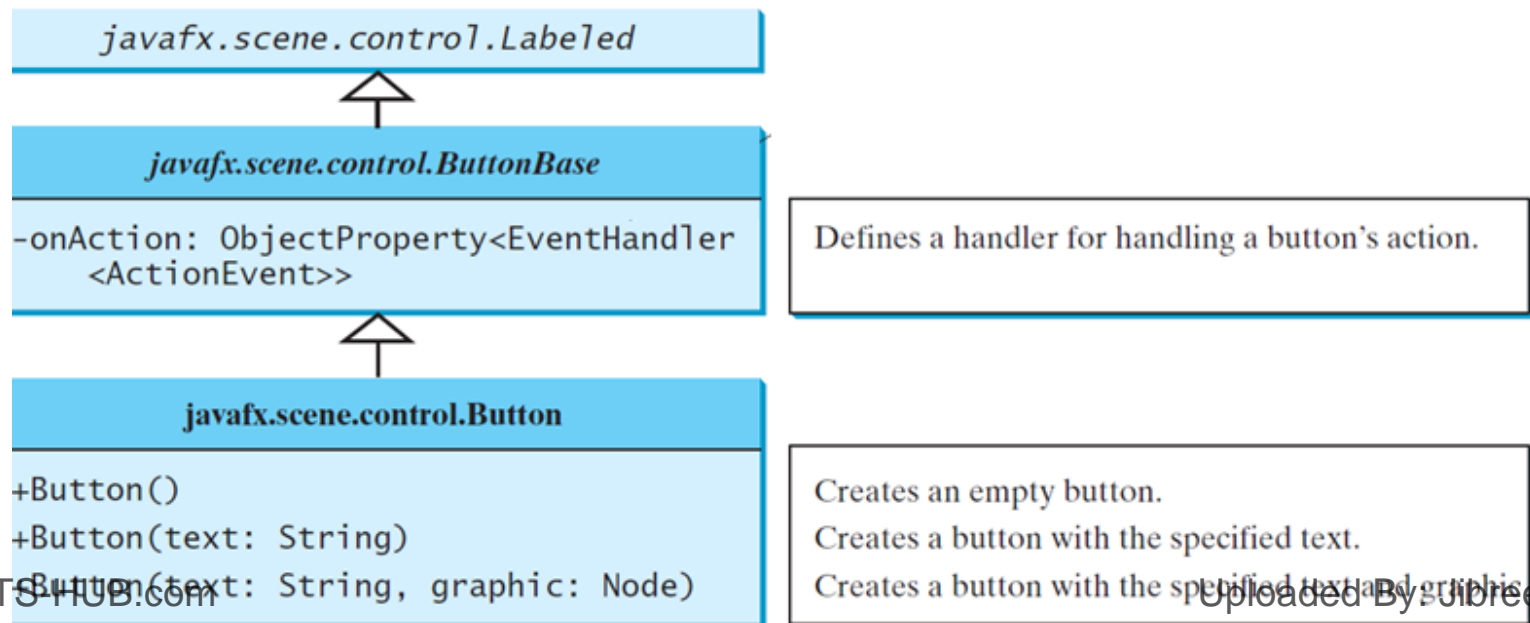
CheckBox

❖ 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	Defines a handler for handling a button's action.
-onAction: ObjectProperty<EventHandler<ActionEvent>>	

⇧

javafx.scene.control.CheckBox	Indicates whether this check box is checked.
-selected: BooleanProperty	
+CheckBox() +CheckBox(text: String)	Creates an empty check box. Creates a check box with the specified text.

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).

javafx.scene.control.ToggleButton	
-selected: BooleanProperty	Indicates whether the button is selected.
-toggleGroup: ObjectProperty<ToggleGroup>	Specifies the button group to which the button belongs.
+ToggleButton()	Creates an empty toggle button.
+ToggleButton(text: String)	Creates a toggle button with the specified text.
+ToggleButton(text: String, graphic: Node)	Creates a toggle button with the specified text and graphic.

javafx.scene.control.RadioButton	
+RadioButton()	Creates an empty radio button.
+RadioButton(text: String)	Creates a radio button with the specified text.

8

TextField

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

---

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.

TextArea

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

javafx.scene.control.TextInputControl	
-text: StringProperty	The text content of this control.
-editable: BooleanProperty	Indicates whether the text can be edited by the user.

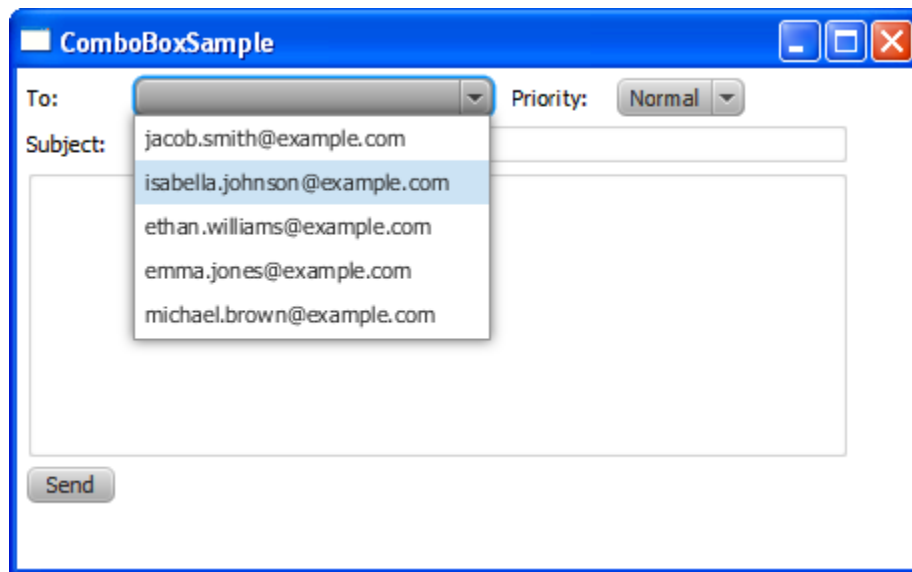| javafx.scene.control.TextArea	
-prefColumnCount: IntegerProperty	Specifies the preferred number of text columns.
-prefRowCount: IntegerProperty	Specifies the preferred number of text rows.
-wrapText: BooleanProperty	Specifies whether the text is wrapped to the next line.
+TextArea()	Creates an empty text area.
+TextArea(text: String)	Creates a text area with the specified text.

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.

javafx.scene.control.ComboBoxBase<T>	
-value: ObjectProperty<T>	The value selected in the combo box.
-editable: BooleanProperty	Specifies whether the combo box allows user input.
-onAction: ObjectProperty<EventHandler<ActionEvent>>	Specifies the handler for processing the action event.

⬆

javafx.scene.control.ComboBox<T>	
-items: ObjectProperty<ObservableList<T>>	The items in the combo box popup.
-visibleRowCount: IntegerProperty	The maximum number of visible rows of the items in the combo box popup.
+ComboBox()	Creates an empty combo box.
+ComboBox(items: ObservableList<T>)	Creates a combo box with the specified items.

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.

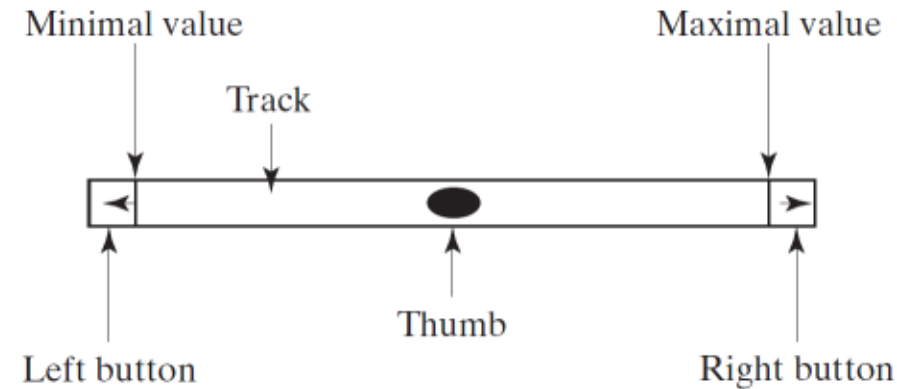| javafx.scene.control.ListView<T>	
-items: ObjectProperty<ObservableList<T>>	The items in the list view.
-orientation: BooleanProperty	Indicates whether the items are displayed horizontally or vertically in the list view.
-selectionModel: ObjectProperty<MultipleSelectionModel<T>>	Specifies how items 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.

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.
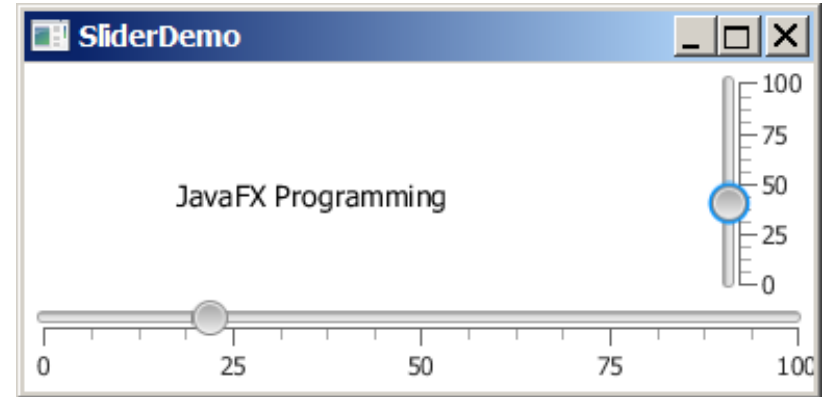

```
                javafx.scene.control.Slider
-blockIncrement: DoubleProperty                        The amount to adjust the slider if the track of the bar is clicked (default: 10).
-max: DoubleProperty                                   The maximum value represented by this slider (default: 100).
-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,                      Creates a slider with the specified min, max, and value.
    value: double)
```

Case Study: TicTacToe

X won! The game is over

Draw! The game is over

```
javafx.scene.layout.Pane
```

```
                   Cell

-token: char

+getToken(): char
+setToken(token: char): void
-handleMouseClick(): void
```
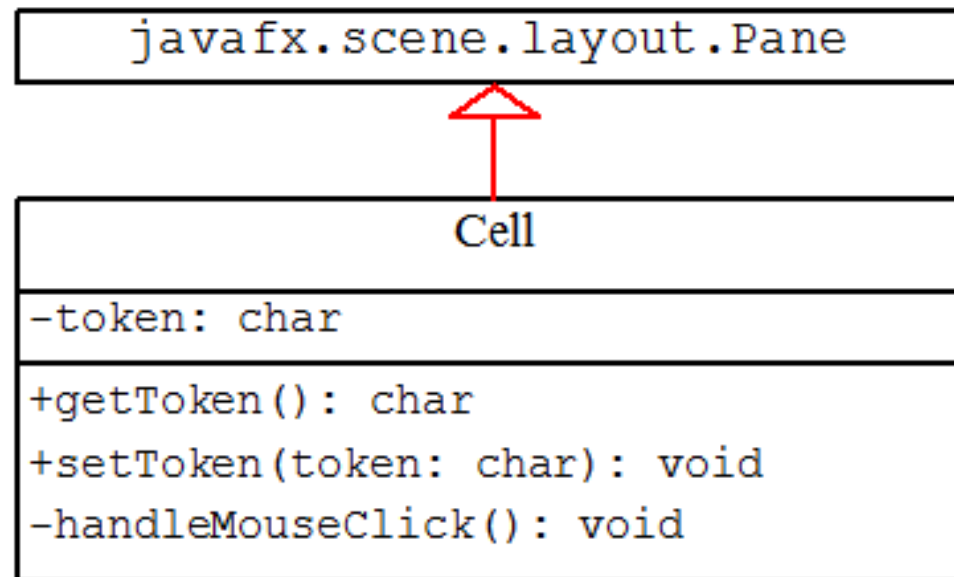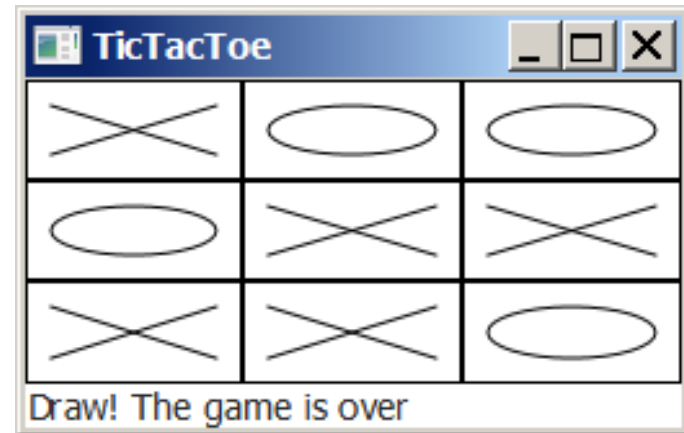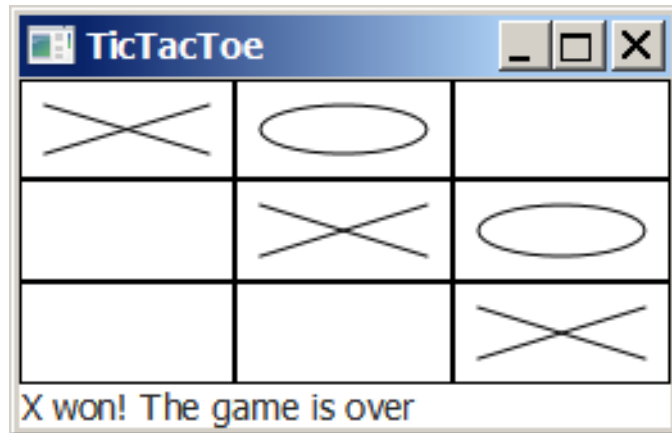
Token used in the cell (default: ' ').

Returns the token in the cell.

Sets a new token in the cell.

Handles a mouse click event.

Case Study: TicTacToe cont.

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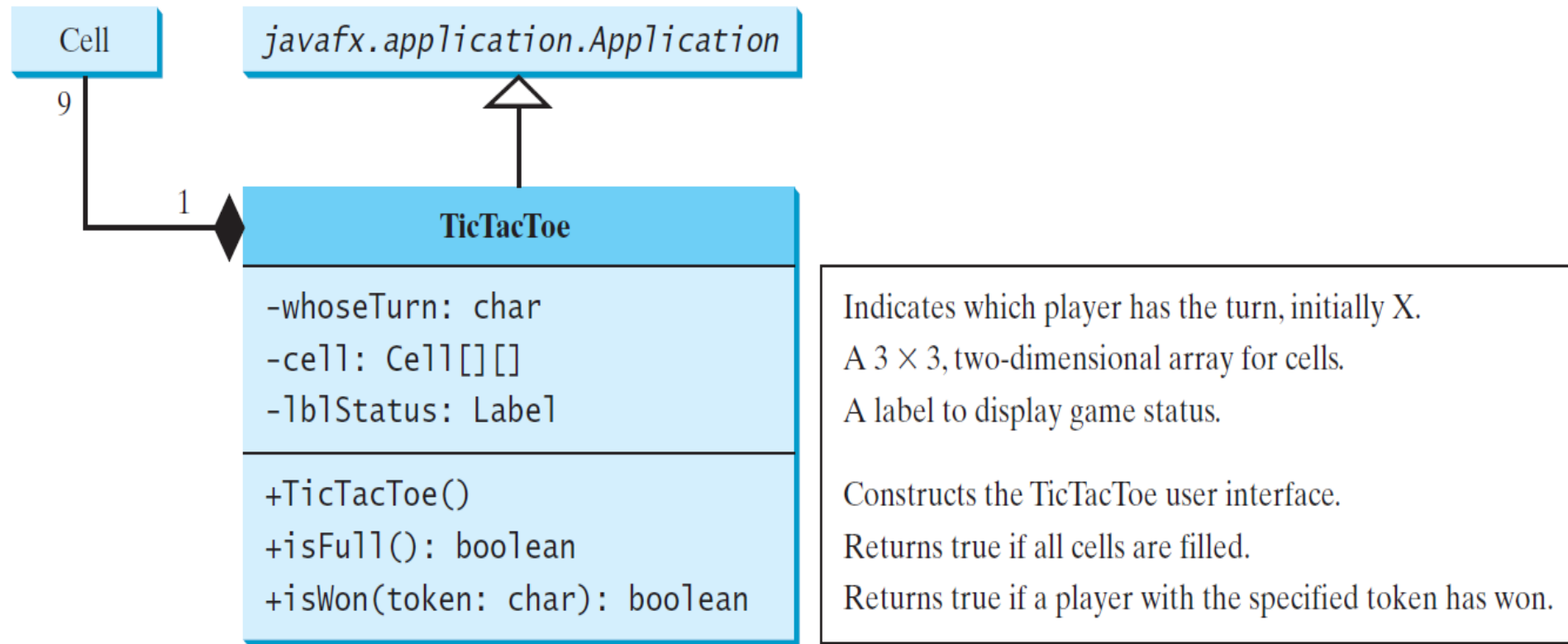
TicTacToe

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

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

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

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

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)

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.