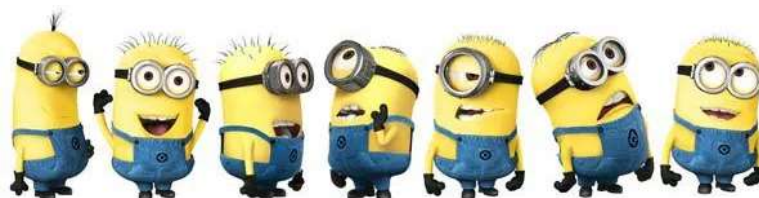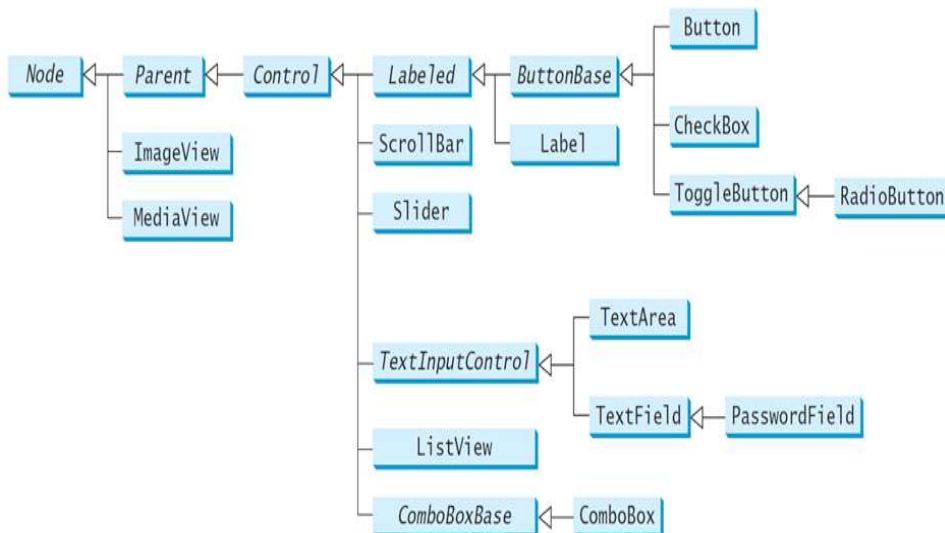# JavaFX UI Controls

Liang, Introduction to Java programming, 11th Edition, © 2017 Pearson Education, Inc.
All rights reserved

By: Mamoun Nawahdah (Ph.D.)
2020

Frequently Used UI Controls

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).

❖ **Label**s and **Button**s 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.

3

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.

4

ButtonBase and Button [Say 'Hello World']

❖ A *Button* is a UI component that triggers an action event when clicked.

❖ JavaFX provides regular buttons, toggle buttons, check box buttons, and radio buttons.

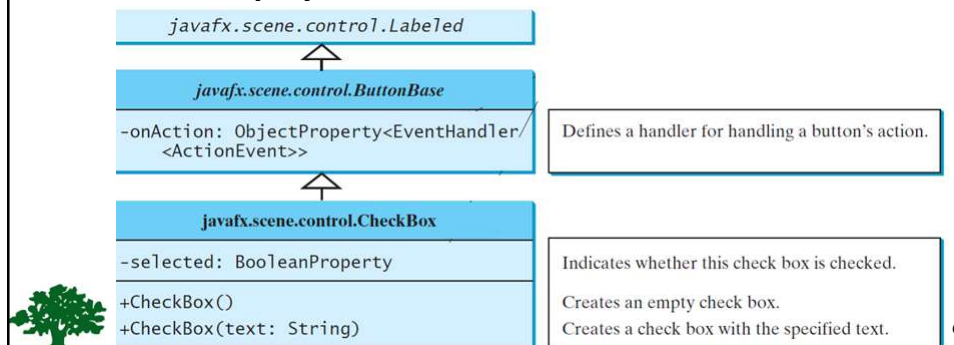❖ The common features of these buttons are defined in **ButtonBase** and **Labeled** classes.

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

5

CheckBox

Which pets do you have?
- ☑ Dog
- ☑ Cat
- ☐ Lizard
- ☐ Bird

❖ A **CheckBox** is used for the user to make a selection (**0 or more**).

❖ **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>>	Defines a handler for handling a button's action.

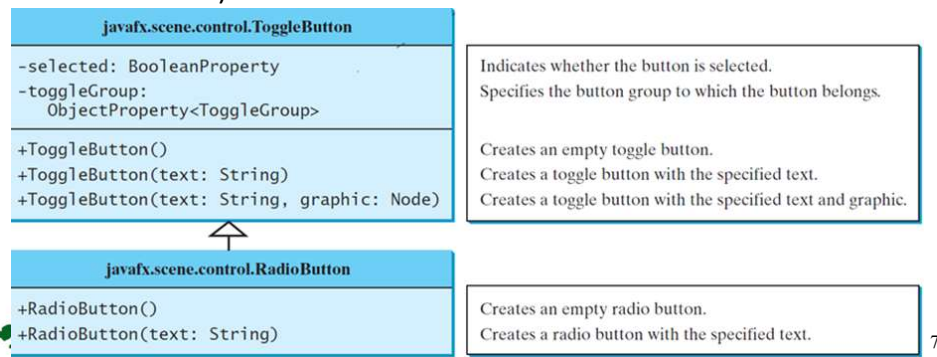| javafx.scene.control.CheckBox	
-selected: BooleanProperty	Indicates whether this check box is checked.
+CheckBox()	Creates an empty check box.
+CheckBox(text: String)	Creates a check box with the specified text.

6

RadioButton

Do you have pets?
- ⦿ Yes
- ○ No

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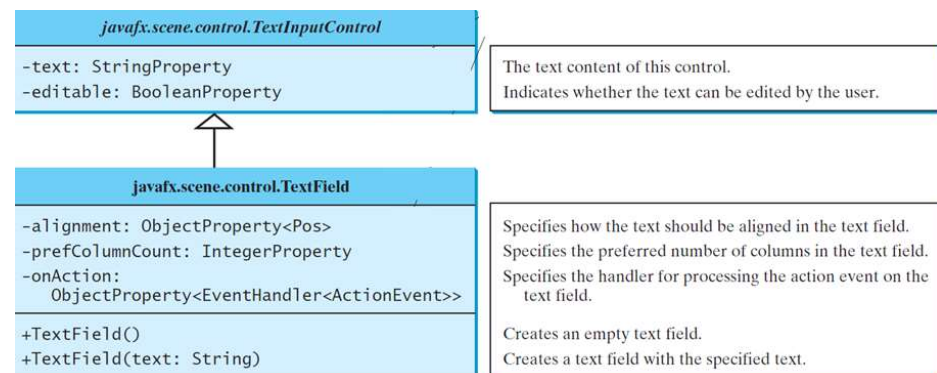
javafx.scene.control.ToggleButton	
-selected: BooleanProperty -toggleGroup: ObjectProperty<ToggleGroup>	Indicates whether the button is selected. Specifies the button group to which the button belongs.
+ToggleButton() +ToggleButton(text: String) +ToggleButton(text: String, graphic: Node)	Creates an empty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

7

TextField

Enter first your name.

❖ 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>>	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.
+TextField() +TextField(text: String)	Creates an empty text field. Creates a text field with the specified text.

8

TextArea

Text Area 1|

❖ 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.

9

ComboBox

A
B
C
D
E

❖ 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.

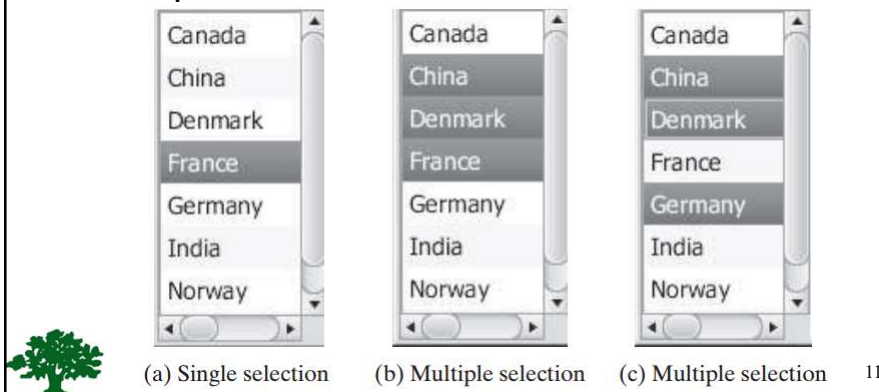| javafx.scene.control.ComboBox<T>	
-items: ObjectProperty<ObservableList<T>>	The items in the combo box popup.
-visibleRowCount: IntegerProperty	The maximum number of visible rows of the items in the combo box popup.
+ComboBox()	Creates an empty combo box.
+ComboBox(items: ObservableList<T>)	Creates a combo box with the specified items.
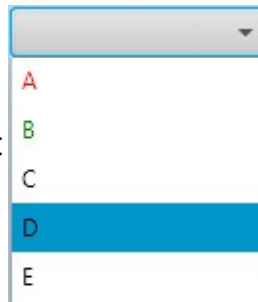
10

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.
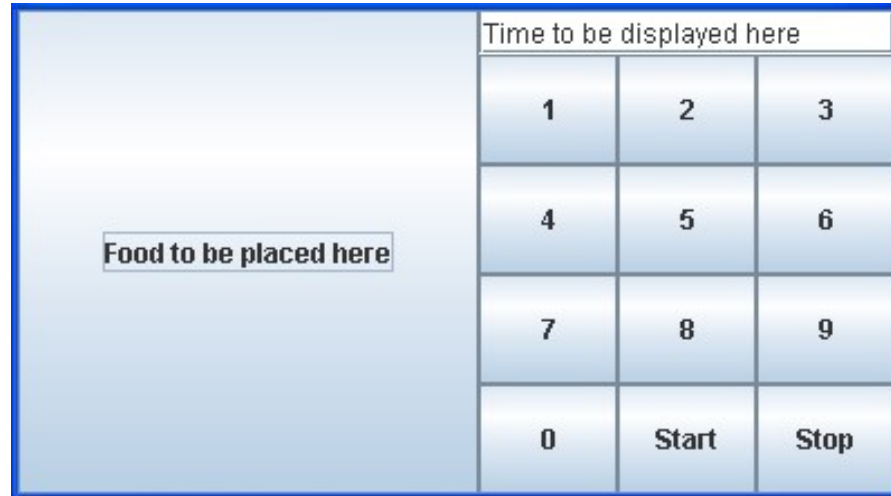

(a) Single selection   (b) Multiple selection   (c) Multiple selection   11

ListView

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.

12

Building a Microwave Interface

	Time to be displayed here		
	1	2	3
Food to be placed here	4	5	6
	7	8	9
	0	Start	Stop

Case Study: TicTacToe

TicTacToe — X won! The game is over

TicTacToe — Draw! The game is over
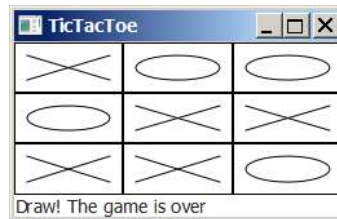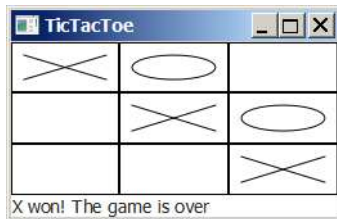
```
javafx.scene.layout.Pane
```

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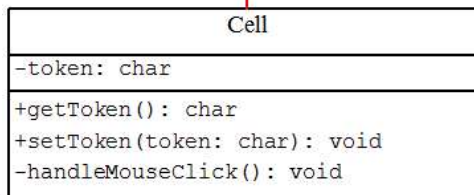
Token used in the cell (default: ' ').

Returns the token in the cell.
Sets a new token in the cell.
Handles a mouse click event.

14

Case Study: TicTacToe cont.

Cell

javafx.application.Application

9

1

TicTacToe

-whoseTurn: char
-cell: Cell[][]
-lblStatus: Label

+TicTacToe()
+isFull(): boolean
+isWon(token: char): boolean

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.

15