BIRZEIT UNIVERSITY

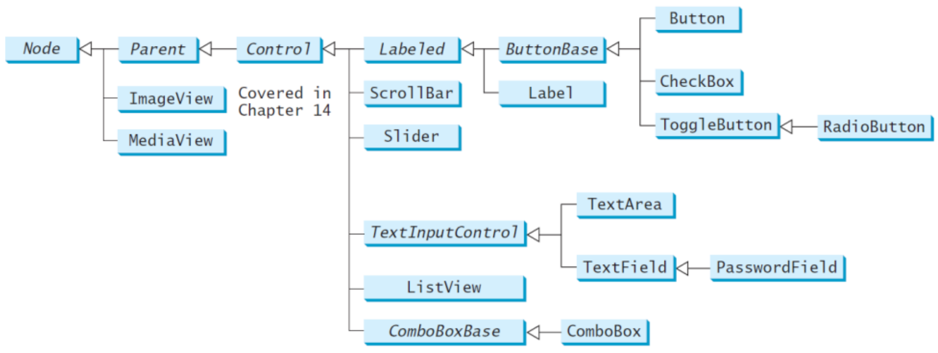
COMPUTER SCIENCE DEPARTMENT FACULTY OF ENGINEERING AND TECHNOLOGY

ADVANCED PROGRAMMING COMP231

Instructor :Murad Njoum
Office : Masri322

Chapter 16 JavaFX UI Controls

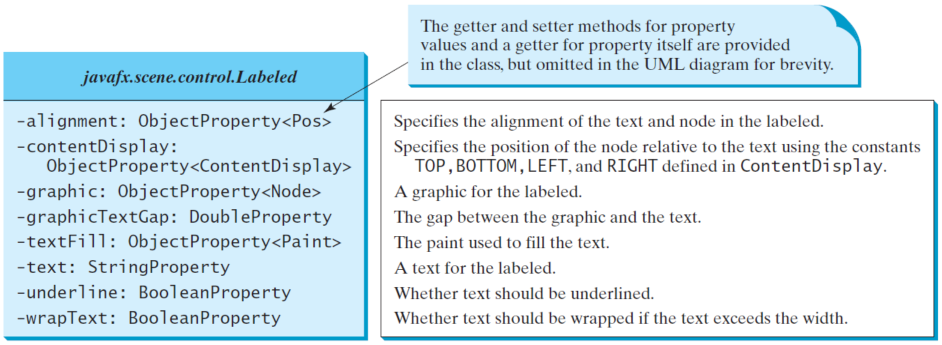
Frequently Used UI Controls



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.



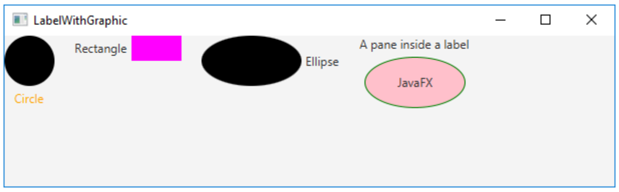
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

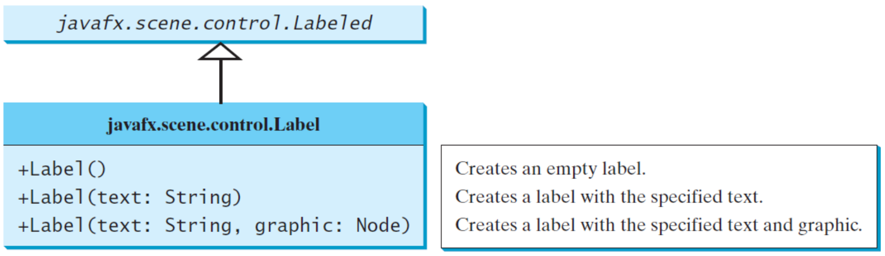
javafx.scene.control.Labeled

`-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

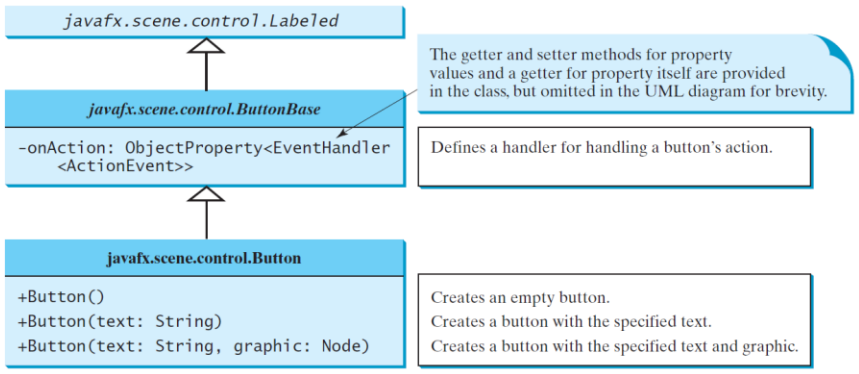
`+Label()`	Creates an empty label.
`+Label(text: String)`	Creates a label with the specified text.
`+Label(text: String, graphic: Node)`	Creates a label with the specified text and graphic.

LabelWithGraphic Run

```
Label lb2 = new Label("Circle", new Circle(50, 50, 25));
    lb2.setContentDisplay(ContentDisplay.TOP);
    lb2.setTextFill(Color.ORANGE);
    Rectangle rectangle=new Rectangle(10, 10, 50, 25);
    rectangle.setFill(Color.MAGENTA);
    Label lb3 = new Label("Rectangle",rectangle);
    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.PINK);
    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( lb2, lb3, lb4, lb5);
```

ButtonBase and Button

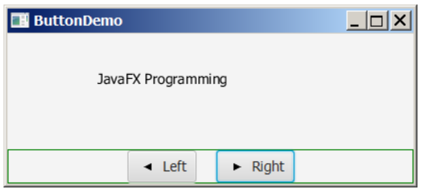
A *button* is a control that triggers an action event when clicked. JavaFX provides regular buttons, toggle buttons, check box buttons, and radio buttons. The common features of these buttons are defined in **ButtonBase** and **Labeled** classes.


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

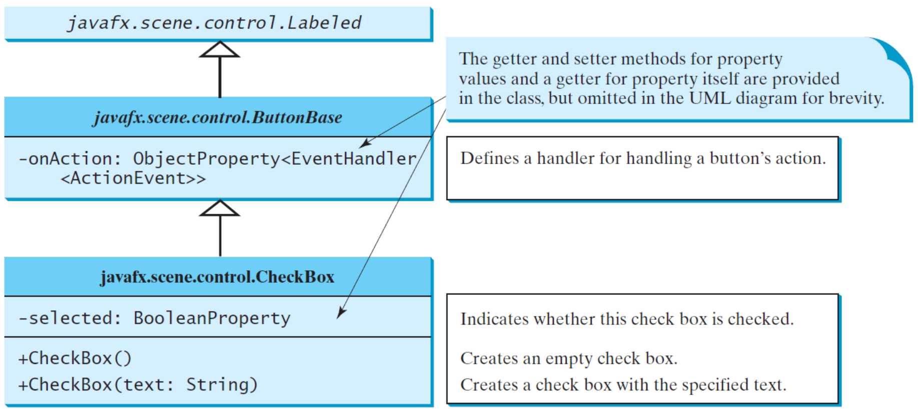
```
protected Text text = new Text(50, 50, "JavaFX Programming");

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);


    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;
  }
```

```
public void start(Stage primaryStage) {
Scene scene = new Scene(getPane(), 450, 200);
    primaryStage.setTitle("ButtonDemo");
primaryStage.setScene(scene);
primaryStage.show();
  }
```

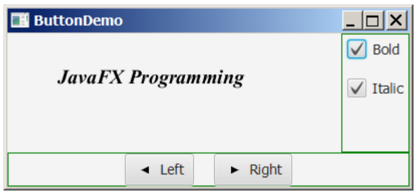


CheckBox

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 Example

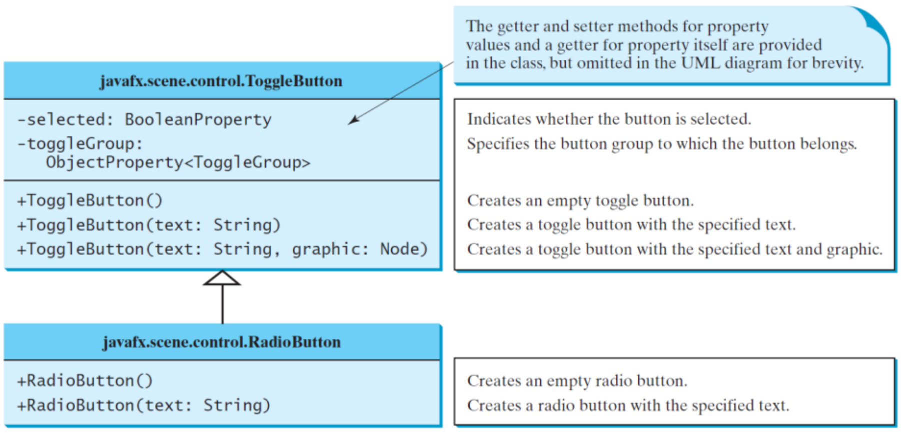


```
CheckBox chkBold = new CheckBox("Bold");
CheckBox chkItalic = new CheckBox("Italic");
```

CheckBoxDemo Run

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).

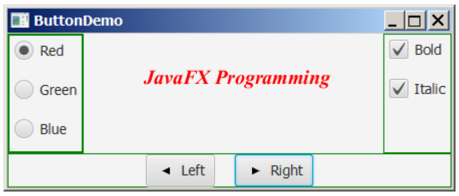


The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.control.ToggleButton	
-selected: BooleanProperty	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.

RadioButton Example



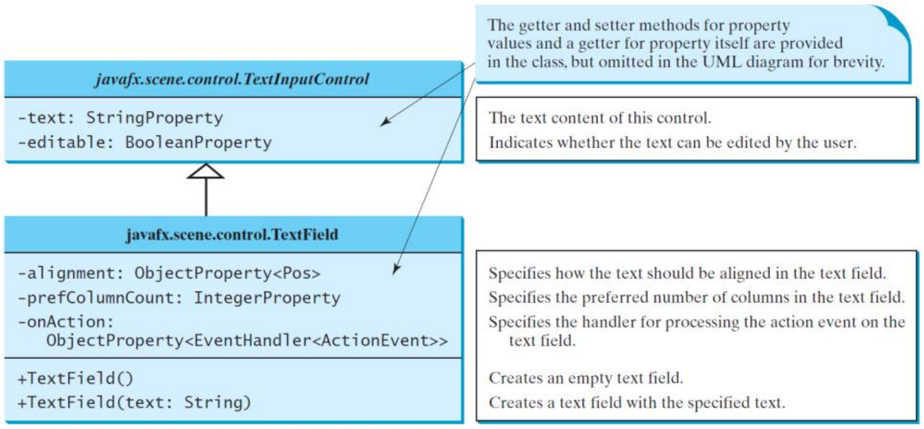
```
RadioButton rbRed = new RadioButton("Red");
RadioButton rbGreen = new RadioButton("Green");
RadioButton rbBlue = new RadioButton("Blue");
paneForRadioButtons.getChildren().addAll(rbRed, rbGreen, rbBlue);
```

```
ToggleGroup group = new ToggleGroup();

rbRed.setToggleGroup(group);
rbGreen.setToggleGroup(group);
rbBlue.setToggleGroup(group);
```

```
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);
  }
});
```

TextField

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



javafx.scene.control.TextInputControl

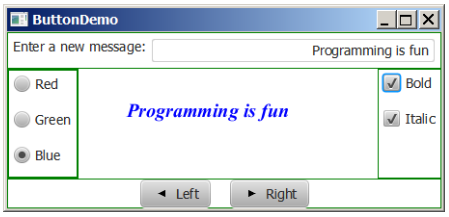
-text: StringProperty	The text content of this control.
-editable: BooleanProperty	Indicates whether the text can be edited by the user.

javafx.scene.control.TextField

-alignment: ObjectProperty<Pos>	Specifies how the text should be aligned in the text field.
-prefColumnCount: IntegerProperty	Specifies the preferred number of columns in the text field.
-onAction: ObjectProperty<EventHandler<ActionEvent>>	Specifies the handler for processing the action event on the text field.
+TextField()	Creates an empty text field.
+TextField(text: String)	Creates a text field with the specified 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.

TextField Example

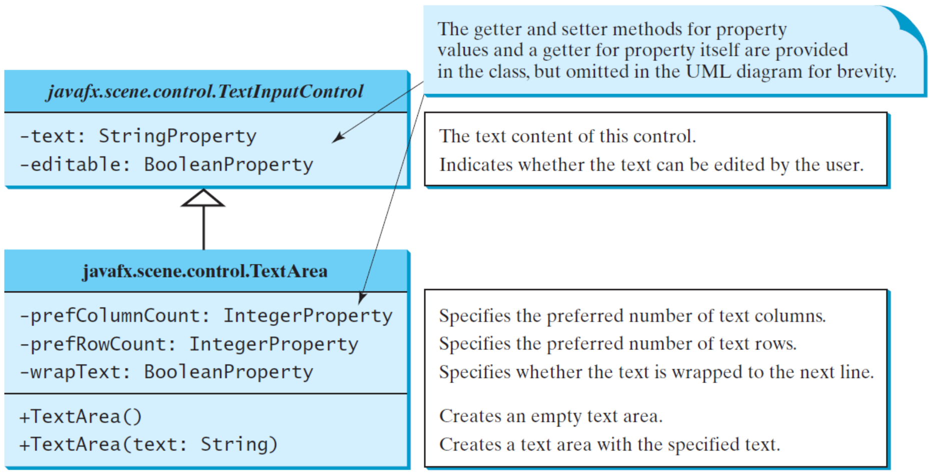


```
BorderPane paneForTextField = new BorderPane();
paneForTextField.setPadding(new Insets(5, 5, 5, 5));
paneForTextField.setStyle("-fx-border-color: green");
paneForTextField.setLeft(new Label("Enter a new message: "));

TextField tf = new TextField();
tf.setAlignment(Pos.BOTTOM_RIGHT);
paneForTextField.setCenter(tf);
pane.setTop(paneForTextField);
```

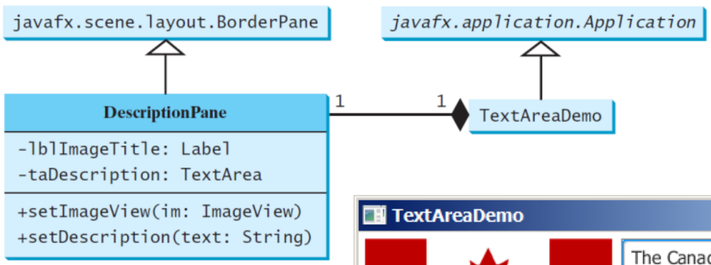
TextFieldDemo Run

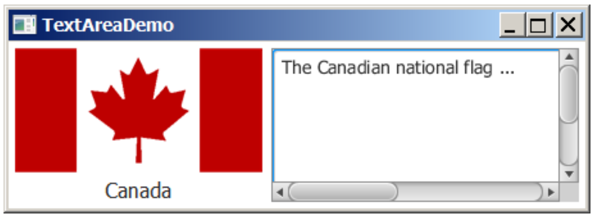
TextArea

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

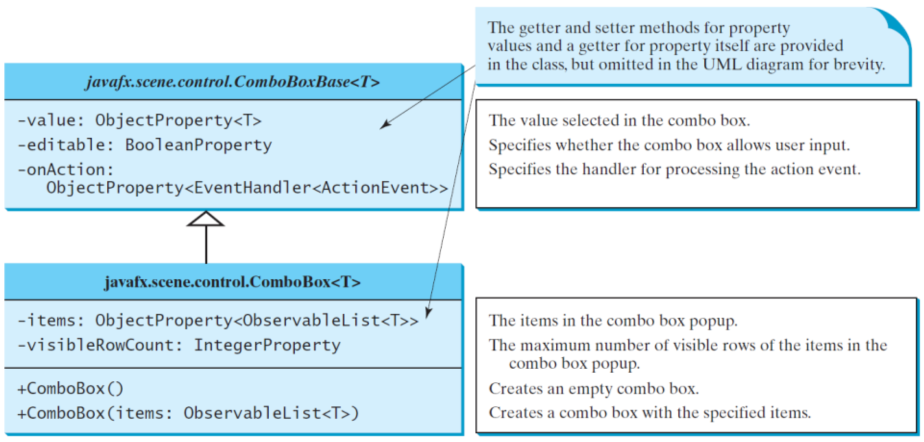

javafx.scene.layout.BorderPane
javafx.application.Application
DescriptionPane
-lblImageTitle: Label
-taDescription: TextArea
+setImageView(im: ImageView)
+setDescription(text: String)
1
1
TextAreaDemo



DescriptionPane TextAreaDemo Run

ComboBox

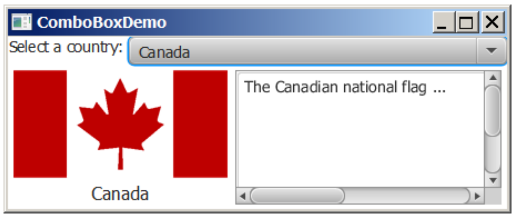
A combo box, also known as a choice list or drop-down list, contains a list of items from which the user can choose.


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<>();
```



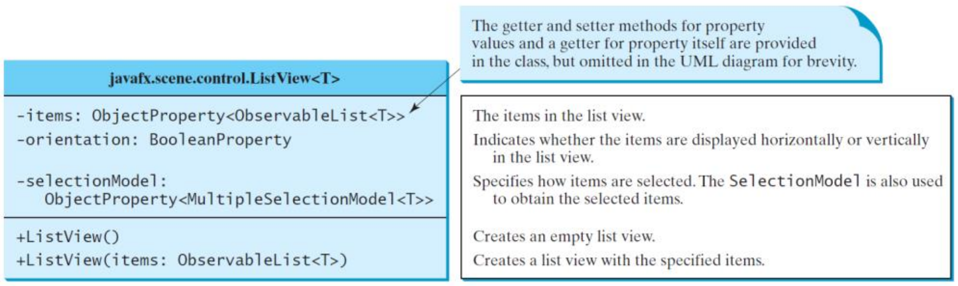
The Canadian national flag ...

```
ObservableList<String> items = FXCollections.observableArrayList(flagTitles);
cbo.getItems().addAll(items);
```

ComboBoxDemo Run

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.

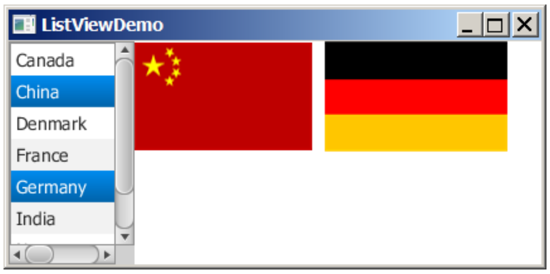


The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

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

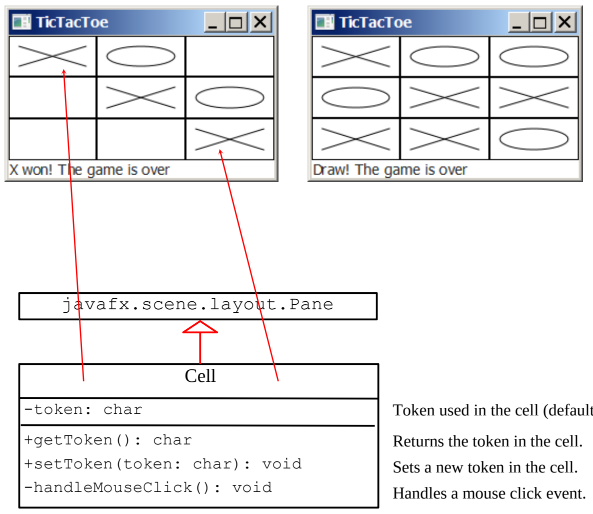
This example gives a program that lets users select countries in a list and display the flags of the selected countries in the labels.



```
ListView<String> lv = new ListView<> (FXCollections.observableArrayList(flagTitles));
```

ListViewDemo Run

Case Study: TicTacToe



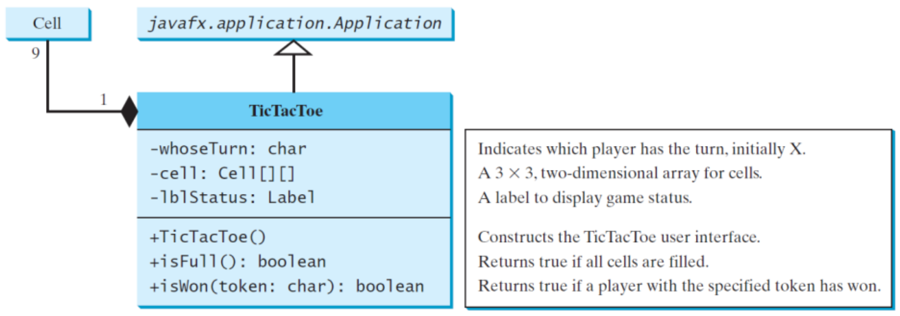
Token used in the cell (default: ' ').

Returns the token in the cell.

Sets a new token in the cell.

Handles a mouse click event.

Case Study: TicTacToe, cont.


Cell
9
1
javafx.application.Application
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.

TicTacToe Run