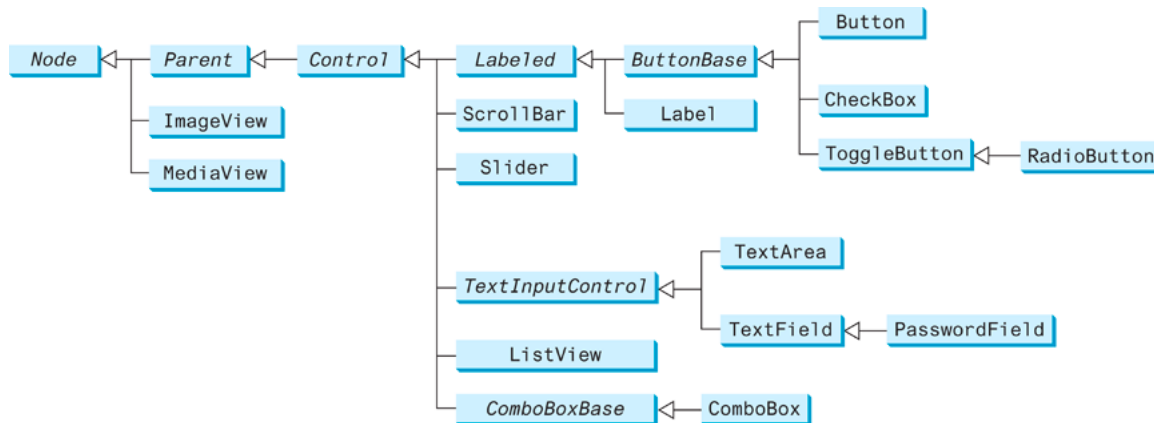


• UI Controls

JavaFX provides many UI controls for developing a comprehensive user interface. A graphical user interface (GUI) makes a program user-friendly and easy to use. Creating a GUI requires creativity and knowledge of how UI controls work. Since the UI controls in JavaFX are very flexible and versatile, you can create a wide assortment of useful user interfaces for rich GUI applications.



• Label

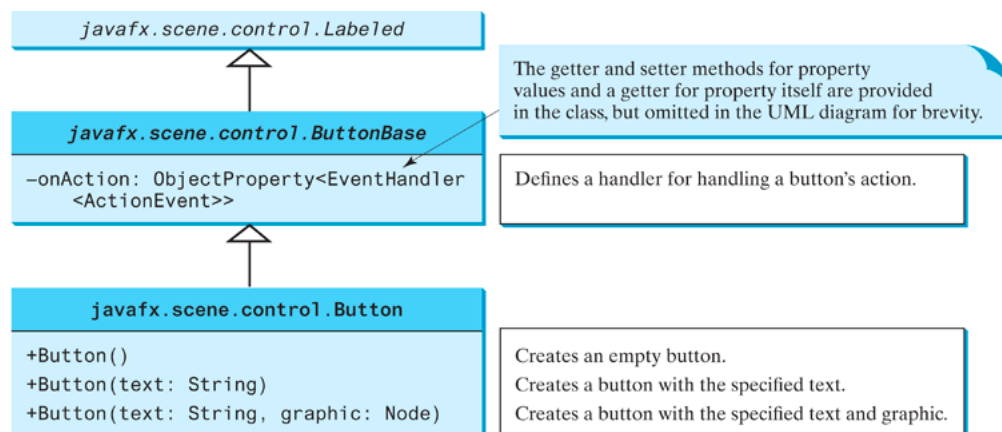
A label is a display area for a short text, a node, or both. It is often used to label other controls (usually text fields). Labels and buttons share many common properties. These common properties are defined in the Labeled class: <https://openjfx.io/javadoc/17/javafx.controls/javafx/scene/control/Labeled.html>

Useful Methods

```
void setText(String value) //Sets the value of the property text
String getText() //Gets the value of the property text
```

• 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:



Useful Methods

```
//Sets the value of the property onAction
void setOnAction(EventHandler<ActionEvent> value)
void setDisable(boolean value) //Sets the value of the property disable
```

- **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, and contentDisplay from ButtonBase and Labeled. In addition, it provides the selected property to indicate whether a check box is selected. <https://openjfx.io/javadoc/17/javafx.controls/javafx/scene/control/CheckBox.html>

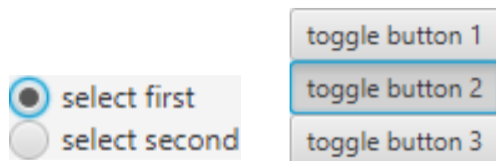
Useful Methods

```
boolean isSelected() //Gets the value of the property selected
void setSelected(boolean value) //Sets the value of the property selected
void setDisable(boolean value) //Sets the value of the property disable
```

- **RadioButton and RadioButtonGroup**

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). RadioButton is a subclass of ToggleButton. The difference between a radio button and a toggle button is that a radio button displays a circle, but a toggle button is rendered similar to a button. The following code create 2 objects of RadioButton and group them as a ToggleGroup so they can be single selected.

```
ToggleGroup group = new ToggleGroup();
RadioButton button1 = new RadioButton("select first");
button1.setToggleGroup(group);
button1.setSelected(true);
RadioButton button2 = new RadioButton("select second");
button2.setToggleGroup(group);
```



Useful Methods

```
boolean isSelected() //Gets the value of the property selected
void setSelected(boolean value) //Sets the value of the property selected
void setDisable(boolean value) //Sets the value of the property disable
```

- **TextField**

A text field can be used to enter or display a **string**. TextField is a subclass of TextInputControl. If a text field is used for entering a password, use PasswordField to replace -TextField. PasswordField extends TextField and hides the input text with echo characters *****. For example,

```
var passwordField = new PasswordField(); //var can be any datatype
passwordField.setText("password123");
```

Useful Methods

```
void setText(String value) //Sets the value of the property text
String getText() //Gets the value of the property text
void setDisable(boolean value) //Sets the value of the property disable
public final void setEditable(boolean value) //Sets the property editable
```

- **TextArea**

If you want to let the user enter **multiple lines** of text, you may create several instances of TextField. A better alternative, however, is to use TextArea, which enables the user to enter multiple lines of text.

Useful Methods

```
void setText(String value) //Sets the value of the property text
String getText() //Gets the value of the property text
void setDisable(boolean value) //Sets the value of the property disable
public final void setEditable(boolean value) //Sets the property editable
void appendText(String text) //Appends a sequence of characters
void clear() //Clears the text
```

- **DatePicker**

The DatePicker control allows the user to enter a date as text or to select a date from a calendar popup. The calendar is based on either the standard ISO-8601 chronology or any of the other chronology classes defined in the java.time.chrono package.

Once the user enter a date or pick a date from the calendar, you can use the **.getValue()** method to get the date entered or selected. The getValue() method returns an instance of LocalDate. The toString() method in LocalDate will return a string in “yyyy-mm-dd” format. For example,

```
String date = dob.getValue().toString(); //dob is an instance of DatePicker
```

- **MenuBar**

You can create menus in JavaFX. Menus make selection easier and are widely used in window applications. JavaFX provides five classes that implement menus: MenuBar, Menu, MenuItem, CheckMenuItem, and RadioButtonMenuItem.

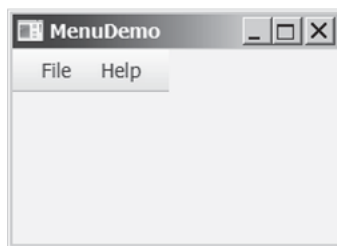
MenuBar is a top-level menu component used to hold the menus. A menu consists of menu items that the user can select (or toggle on or off). A menu item can be an instance of `MenuItem`, `CheckMenuItem`, or `RadioButtonMenuItem`. Menu items can be associated with nodes and keyboard accelerators. The sequence of implementing menus in JavaFX is as follows:

1. Create a menu bar and add it to a pane. For example, the following code creates a pane and a menu bar, and adds the menu bar to the pane:

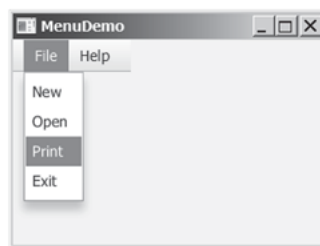
```
MenuBar menuBar = new MenuBar();
Pane pane = new Pane();
pane.getChildren().add(menuBar);
```

2. Create menus and add them under the menu bar. For example, the following creates two menus and adds them to a menu bar,

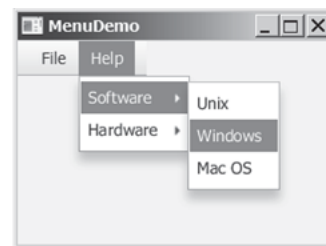
```
Menu menuFile = new Menu("File");
Menu menuHelp = new Menu("Help");
menuBar.getMenus().addAll(menuFile, menuHelp);
Menu softwareHelpSubMenu = new Menu("Software");
Menu hardwareHelpSubMenu = new Menu("Hardware");
menuHelp.getItems().add(softwareHelpSubMenu);
menuHelp.getItems().add(hardwareHelpSubMenu);
softwareHelpSubMenu.getItems().add(new MenuItem("Unix"));
softwareHelpSubMenu.getItems().add(new MenuItem("Windows"));
softwareHelpSubMenu.getItems().add(new MenuItem("Mac OS"));
```



(a)



(b)



(c)

You can also add a `CheckMenuItem` to a `Menu`. `CheckMenuItem` is a subclass of `MenuItem` that adds a Boolean state to the `MenuItem` and displays a check when its state is true. For example,

```
menuHelp.getItems().add(new CheckMenuItem("Check it"));
```

The menu items generate `ActionEvent`. To handle `ActionEvent`, implement the `setOnAction` method. The `MenuItem` has a `graphic` property for specifying a node to be displayed in the menu item. Usually, the graphic is an image view. The classes `Menu`, `MenuItem`, `CheckMenuItem`, and `RadioButtonMenuItem` have another constructor that you can use to specify a graphic. For example,

```
Menu menuFile = new Menu("File", new ImageView("image/usIcon.gif"));
```

• 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. A combo box is useful for limiting a user's range of choices and avoids the cumbersome validation of data input. For example, the following code create an object of ComboBox with 3 color selections.

```
ComboBox<Color> cmb = new ComboBox<>();
cmb.getItems().addAll(
    Color.RED,
    Color.GREEN,
    Color.BLUE);
```

Useful Methods

```
ObservableList<T> getItems() //Gets the value of the property items
void setItems(ObservableList<T> value) //Sets the value of items
SingleSelectionModel<T> getSelectionModel() //Gets selectionModel
```

After getting the SelectionModel the following method can be used. For example, `combobox.getSelectionModel().getSelectedItem()`.

- `int getSelectedIndex()` //Returns the index of selected item in this model
- `T getSelectedItem()` //Returns the currently selected object

• ListView

A list view is a control that basically performs the same function as a ComboBox, but it enables the user to choose a single value or multiple values. For example, the following code creates an object of ListView and set the content of the ListView.

```
ObservableList<T> content = ...
ListView listView = new ListView();
listView.setItems(content);
```



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

Useful Methods

```
ObservableList<T> getItems() //Gets the value of the property items
```

```
void setItems(ObservableList<T> value) //Sets the value of items
void setEditable(boolean value) //Sets the value of the property editable.
MultipleSelectionModel<T> getSelectionModel() //Returns the selection model
```

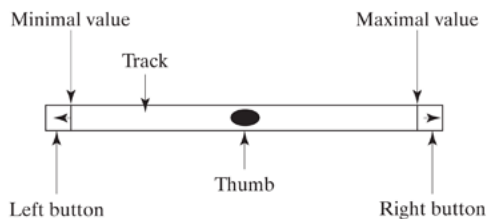
After getting the SelectionModel the following method can be used. For example, `listview.getSelectionModel().getSelectedItem()`.

- `int getSelectedIndex()` //Returns the index of selected item in this model
- `T getSelectedItem()` //Returns the currently selected object

• ScrollBar

ScrollBar is a control that enables the user to select from a range of values.

<https://openjfx.io/javadoc/17/javafx.controls/javafx/scene/control/ScrollBar.html>



• Audio and Video

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.

Media (video and audio) is essential in developing rich GUI applications. JavaFX provides the Media, MediaPlayer, and MediaView classes for working with media. Currently, JavaFX supports MP3, AIFF, WAV, and MPEG-4 audio formats and FLV and MPEG-4 video formats. The Media class represents a media source with properties duration, width, and height.

<https://openjfx.io/javadoc/17/javafx.media/javafx/scene/media/Media.html>

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

<https://openjfx.io/javadoc/17/javafx.media/javafx/scene/media/MediaPlayer.html>

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.

<https://openjfx.io/javadoc/17/javafx.media/javafx/scene/media/MediaView.html>