Topic 6: JavaFX

Part d: More Controls

CSS

Properties

- JavaFX supports a large number of CSS properties.
- Part of the process of creating stylesheets is determining which properties are available for the nodes in your application.
- Oracle publishes a comprehensive CSS reference guide documenting all the available properties: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html

Applying a Stylesheet to a JavaFX Application (1 of 2)

- A stylesheet is a text file containing style definitions.
- Stylesheets are usually saved with the `.css` file extension.
- To apply a stylesheet to a JavaFX application:
  - Save the stylesheet in the same directory as the JavaFX application
  - Use the scene object's `getStylesheets().add()` method to apply the stylesheet

Applying a Stylesheet to a JavaFX Application (2 of 2)

- Example:
  - We have a stylesheet named `mystyles.css`
  - In our Java code, the `scene` variable references the Scene object
  - We would use the following statement to add the stylesheet to the scene:

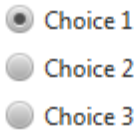
```
scene.getStylesheets().add("mystyles.css");
```

Radio Buttons

RadioButton Controls (1)

- RadioButton controls allow the user to select one choice from several possible options.
- The RadioButton class is in the javafx.scene.control package.

```
RadioButton radio1 = new RadioButton("Choice 1");
RadioButton radio2 = new RadioButton("Choice 2");
RadioButton radio3 = new RadioButton("Choice 3");
```



RadioButton Controls (2)

- RadioButton controls are normally grouped together in a *toggle group*.
  - Only one of the RadioButton controls in a toggle group may be selected at any time.
  - Clicking on a RadioButton selects it and automatically deselects any other RadioButton in the same toggle group.

- To create a toggle group, you use the ToggleGroup class, which is in the javafx.scene.control package:

```
ToggleGroup myToggleGroup = new ToggleGroup();
```

  - After creating a ToggleGroup object, you call each RadioButton control's setToggleGroup method to add them to the ToggleGroup.

RadioButton Controls (3)

- To determine whether a RadioButton is selected, you call the RadioButton class's isSelected method.

```
if (radio1.isSelected())
{
    //  Code here executes if the radio
    //  button is selected.
}
```

- You usually want one of the RadioButtons in a group to be initially selected.
- You can select a RadioButton in code with the RadioButton class's setSelected method:

```
radio1.setSelected(true);
```

Responding to RadioButton Clicks

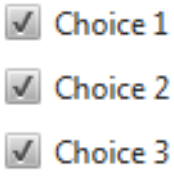
- If you want an action to take place immediately when the user clicks a RadioButton, register an ActionEvent handler with the RadioButton control.
- The process is the same as with the Button control.
- RadioButtonEvent.java

Checkboxes

`CheckBox` Controls

- `CheckBox` controls allow the user to make yes/no or on/off selections.
- The `CheckBox` class is in the `javafx.scene.control` package.

```
CheckBox choice1 = new CheckBox("Choice 1");
CheckBox choice2 = new CheckBox("Choice 2");
CheckBox choice3 = new CheckBox("Choice 3");
```



CheckBox Controls (2)

- To determine whether a CheckBox is selected, you call the CheckBox class's isSelected method:

```
if (check1.isSelected())
{
    // Code here executes if the check
    // box is selected.
}
```

- You can select a CheckBox in code with the CheckBox class's setSelected method:

```
check1.setSelected(true);
```

Responding to CheckBox Clicks

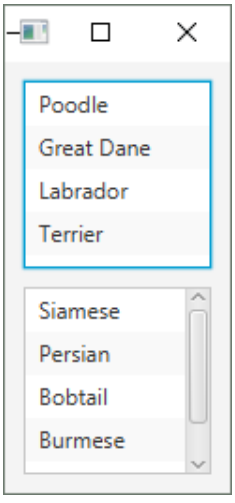
- If you want an action to take place immediately when the user clicks a CheckBox, register an ActionEvent handler with the CheckBox control.

- The process is the same as with the Button control.

ListView Controls

`ListView` Controls

- The `ListView` control displays a list of items and allows the user to select one or more items from the list.



ListView Controls (2)

- The ListView class is in the javafx.scene.control package.

```
ListView<String> dogListView = new ListView<>();
```

- Adding items to the ListView:

```
dogListView.getItems().addAll(
   "Poodle", "Great Dane", "Labrador", "Terrier");
```

- Creating a ListView with a preferred size, use setPrefSize() method

ListView Controls (3)

- Use the ListView class's getSelectionModel().getSelectedItem() method get the item that is currently selected.

```
String selected = listView.getSelectionModel().getSelectedItem();
```

- If no item is selected in the ListView, the method will return null.
- Each item in a ListView control is assigned an index.
- The first item (at the top of the list) has the index 0, the second item has the index 1, and so forth.
- Use the ListView class's getSelectionModel().getSelectedIndex() method get the index of the item that is currently selected:

```
int index =
 lististView.getSelectionModel().getSelectedIndex();
```

- If no item is selected in the ListView, the method will return -1.

ListView Controls (4)

- When the user selects an item in a `ListView`, a *change event* occurs.
- To immediately perform an action when the user selects an item in a ListView, write an event handler that responds to the change event:

```
listView.getSelectionModel().selectedItemProperty()
.addListener(event ->
{
   // Write event handling code here ...
});
```

ListView Controls (5)

- The `ListView` control's `getItems().addAll()` method adds items to the `ListView`.
- If the `ListView` already contains items, the `getItems().addAll()` method will not erase the existing items, but will add the new items to the existing list.
- If you want to replace the existing items in a `ListView`, use the `getItems().addAll()` method instead.

ListView Controls (6)

- To initialize a ListView control with the contents of an array or an ArrayList:
  - Convert the array or ArrayList to an ObservableList. (This requires the FXCollections class, in the javafx.collections package)
  - Pass the ObservableList as an argument to the ListView class's constructor.

ListView Controls (7)

- Example: initializing a ListView control with the contents of an array:

```
// Create a String array.
String[] strArray = {"Monday", "Tuesday", "Wednesday"};

// Convert the String array to an ObservableList.
ObservableList<String> strList =
       FXCollections.observableArrayList(strArray);

// Create a ListView control.
ListView<String> listView = new ListView<>(strList);
```

ListView Controls (8)

- Example: initializing a `ListView` control with the contents of an `ArrayList`:

```
// Create an ArrayList of Strings.
ArrayList<String> strArrayList = new ArrayList<>();
strArrayList.add("Monday");
strArrayList.add("Tuesday");
strArrayList.add("Wednesday");

// Convert the ArrayList to an ObservableList.
ObservableList<String> strList =
       FXCollections.observableArrayList(strArrayList);

// Create a ListView control.
ListView<String> listView = new ListView<>(strList);
```

ListView Controls (9)

- The `ListView` control can operate in either of the following selection modes:
  - Single Selection Mode – The default mode. Only one item can be selected at a time.
  - Multiple Interval Selection Mode – Multiple items may be selected.
- You change the selection mode with the control's `getSelectionModel().setSelectionMode()` method.

ListView Controls (10)

- To change to multiple interval selection mode:

```
listView.getSelectionModel().setSelectionMode(
   SelectionMode.MULTIPLE);
```

- To change back to single selection mode:

```
listView.getSelectionModel().setSelectionMode(
   SelectionMode.SINGLE);
```

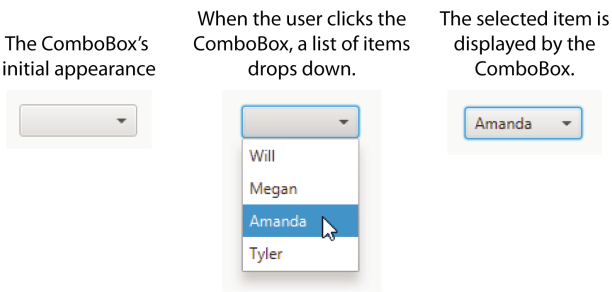
ListView Controls (11)

- When a `ListView` control is in multiple selection mode, the user can select more than one item.
- To get all of the selected items and their indices, use the following methods:
  - `getSelectionModel().getSelectedItems()` — returns a read-only `ObservableList` of the selected items
  - `getSelectionModel().getSelectedIndices()` — returns a read-only `ObservableList` of the integer indices of the selected items

ComboBox Controls

ComboBox Controls

- A `ComboBox` presents a drop-down list of items that the user may select from.
- Use the `ComboBox` class (in the `javafx.scene.control` package) to create a `ComboBox` control:
  ```
  ComboBox<String> nameComboBox = new ComboBox<>();
  ```
- Once you have created a `ComboBox` control, you are ready to add items to it:
  ```
  nameComboBox.getItems().addAll("Will", "Megan",
  "Amanda", "Tyler");
  ```



ComboBox Controls (2)

Some of the ComboBox Methods

Method	Description
`getValue()`	Returns the item that is currently selected in the ComboBox.
`setValue(`*`value`*`)`	Selects *value* in the ComboBox.
`setVisibleRowCount(`*`count`*`)`	The *count* argument is an int. Sets the number of rows, or items, to display in the drop-down list.
`setEditable(`*`value`*`)`	The *value* argument is a boolean. If *value* is true, the ComboBox will be editable. If *value* is false, the ComboBox will be uneditable.
`show()`	Displays the drop-down list of items.
`hide()`	Hides the drop-down list of items.
`isShowing()`	Returns true or false to indicate whether the dropdown list is visible.

ComboBox Controls (3)

- You can use the `ComboBox` class's `getValue()` method get the item that is currently selected:

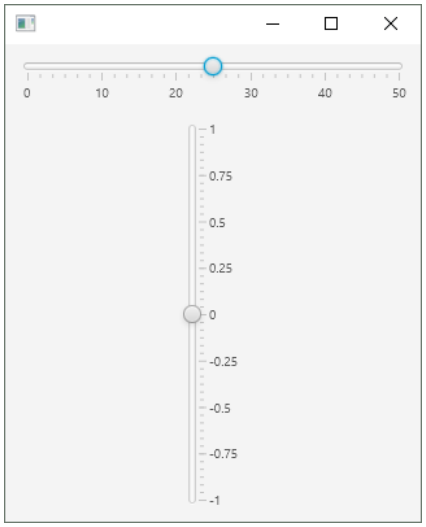
```
String selected = comboBox.getValue();
```

- If no item is selected in the `ComboBox`, the method will return `null`.
- When the user selects an item in a `ListView`, an `ActionEvent` occurs.
- ComboBoxDemo2.java

Slider Controls

`Slider` Controls

- A `Slider` allows the user to graphically adjust a number within a range.
- `Slider`s are created from the `Slider` class (in the `javafx.scene.control` package)
- They display an image of a "slider knob" that can be dragged along a track.
- Between the minimum and maximum values, major tick marks are displayed with a label indicating the value at that tick mark.
- Between the major tick marks are minor tick marks.



`Slider` Controls (2)

- The `Slider` class has two constructors. No-arg:

```
Slider slider = new Slider();
```

- Creates a `Slider` with a minimum value of 0, and a maximum value of 100.
- The `Slider`'s initial value will be set to 0, and no tick marks will be displayed.
- The second constructor accepts three `double` arguments:
  - the minimum value
  - the maximum value
  - the initial value

```
Slider slider = new Slider(0.0, 50.0, 25.0);
```

`Slider` Controls (3)

- When a `Slider`'s value changes, a *change event* occurs.
- To perform an action when the `Slider`'s value changes, write an event handler that responds to the change event:

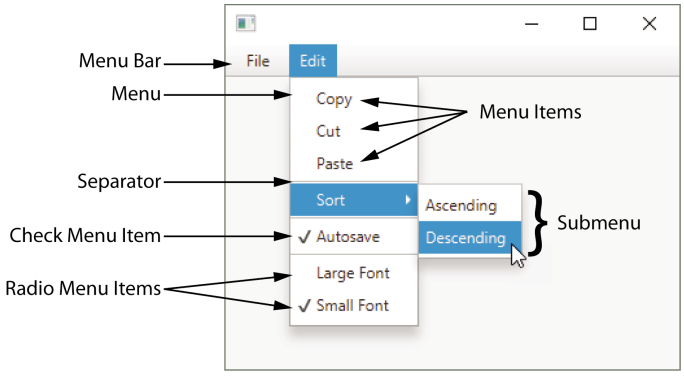
```
slider.valueProperty().addListener((observeable,
oldvalue, newvalue) ->
{
   // Write event handling code here...
});
```

- SliderDemo.java

Menus

Menus

- A menu system is a collection of commands organized in one or more drop-down menus.



Menus (2)

- A menu system commonly consists of:
  - Menu Bar – lists the names of one or menus
  - Menu – a drop-down list of menu items
  - Menu Item – can be selected by the user
  - Check box menu item – appears with a small box beside it
  - Radio button menu item – may be selected or deselected
  - Submenu – a menu within a menu
  - Separator bar – used to separate groups of items

Menus (3)

- A menu system is constructed with the following classes:
  - MenuBar – used to create a menu bar
  - Menu – used to create a menu, containing:
    - MenuItem
    - CheckMenuItem
    - RadioMenuItem
    - other Menu objects
  - MenuItem – Used to create a regular menu, and generates an ActionEventwhen selected.

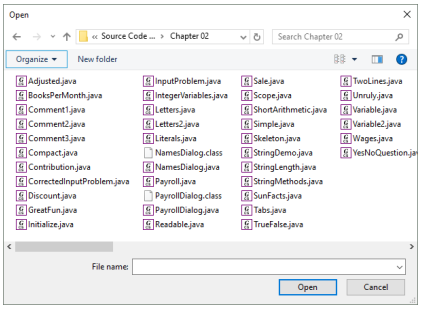
Menus (4)

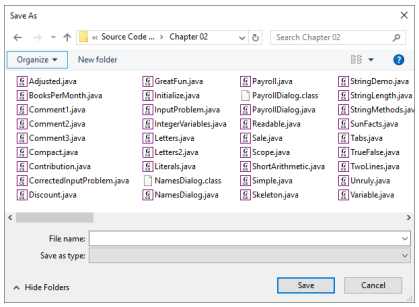
- `CheckMenuItem` – Used to create a checked menu item.
  - The class's `isSelected` method returns `true` if the item is selected, or `false` otherwise.
  - A `CheckMenuItem` component generates an `ActionEvent` when selected.
- `RadioMenuItem` – Used to create a radio button menu item.
  - `RadioMenuItem` components can be grouped together in a `ToggleGroup` so that only one of them can be selected at a time.
  - The class's `isSelected` method returns `true` if the item is selected, or `false` otherwise.
  - A `RadioMenuItem` component generates an `ActionEvent` when selected

FileChooser

The `FileChooser` Class

- The `FileChooser` class (in the `javafx.stage` package) displays a dialog box that allows the user to browse for a file and select it.
- The class can display two types of predefined dialog boxes:
  - open dialog box
  - save dialog box





The `FileChooser` Class (2)

- First, create an instance of the `FileChooser` class
- Then, call either the `showOpenDialog` method, or the `showSaveDialog` method
- With either method, you pass a reference to the application's stage as an argument

```
FileChooser fileChooser = new FileChooser();
File selectedFile = fileChooser.showOpenDialog(primaryStage);
```

The `FileChooser` Class (3)

- The `showOpenDialog` and `showSaveDialog` methods return a `File` object (in the `java.io` package) containing information about the selected file.
- If the user does not select a file, the method returns `null`.

```
FileChooser fileChooser = new FileChooser();
File selectedFile = fileChooser.showOpenDialog(primaryStage);
if (selectedFile != null)
{
   String filename = selectedFile.getPath();
   outputLabel.setText("You selected " + filename);
}
```