WESTERN UNIVERSITY - CANADA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SE 2203B – SOFTWARE DESIGN

Laboratory 2: JavaFX – Basic Controls

Due Date: January 27, 2023 – 11:55PM

For these exercises, no need to physically attend the lab, but if you get stuck on something in the following lab requirements, you should ask your TA in person during the designated lab hours in ACEB 4435. The designated lab hours are mentioned in the Lab schedule document in OWL.

1 Goal

This lab presents the basics of developing graphical user interface (GUI) applications with JavaFX. Fundamental controls, such as `ListView`, `ComboBox`, and `Slider` controls.

2 Resources

- Lab2-Exercise1-1.mp4, Lab2-Exercise1-2.mp4, Lab2-Exercise1-3.mp4, Lab2-Exercise1-4.mp4, Lab2-Exercise2.mp4, and Lab2-Exercise3.mp4.

3 Introduction

The JavaFX library provides classes for all of the objects you can add to a GUI. The classes are in various subpackages of the JavaFX package, so you must write an import statement for the class you want to work with. Once you have written the necessary import statement, you will create an instance of the class. This is often done in the start method. Fortunately, the IDE is helping you to add the proper JavaFX packages automatically, and the Scene Builder design tool will help you can visually create a GUI controls (the instance of the JavaFX classes).

Many of the containers in the JavaFX library implement the `ObservableList` interface. In this lab you will a few of the methods specified in the interface.

Below is a list of common errors you can avoid while you are using Scene Builder to create JavaFX applications.

- Not assigning an `fx:id` to a component. If you need to access a GUI component in your Java code, you must assign the component an `fx:id` in Scene Builder.
- Forgetting to select the controller class for the root node in Scene Builder. Before you can select event listener methods for a GUI's components, you must select the controller class that contains those event listeners, for the root node.
- Forgetting to register the event listener for a component. After you have registered the controller class for the root node, you must select the event listener methods for each component that must respond to events.
- Forgetting to write an event listener for each event you wish an application to respond to. To respond to an event, you must write an event listener of the proper type registered with the component that generates the event.

- Leaving out the `@FXML` attribute in private declarations in the controller class. If you declare a private field in the controller class, and that declaration refers to a component in the FXML file, the declaration must be preceded with the `@FXML` annotation.

4 Directed Lab Work

4.1 The ObservableList Interface

As you spend more time developing JavaFX applications, you will frequently encounter the `ObservableList` interface. It is basically an observable list that provides option to attach listeners for list content change. `ObservableList` has an integral role in the JavaFX development because it is used for major components like `ListView`, `TableView`, `ComboBox`, etc. The object that implements the `ObservableList` interface is a special type of list that can fire an event handler any time an item in the list changes.

4.1.1 Creating and using ObservableList

- You cannot instantiate an `ObservableList` class directly using a new call. Instead, `FXCollections` should be used to create an `ObservableList`. Let's see how we can create an ObservableList of String type. The most common way to create it is using `observableArrayList()` function.

```
ObservableList<String> listInstance = FXCollections.observableArrayList();
//Add a single entry
listInstance.add("Java");
System.out.println(listInstance);

//Add multiple entries
listInstance.addAll("Cpp", "C#", "Python");
System.out.println(listInstance);

//Remove entry
listInstance.remove("Cpp");
System.out.println(listInstance);
```

- As you have already noted, the basic operations are similar to normal lists, except the `addAll()` function availability. **You can add multiple entries to the ObservableList using comma separated values.** The output of the above code is as follows.

```
[Java]
[Java, Cpp, C#, Python]
[Java, C#, Python]
```

- The following are list of a few **ObservableList** methods.

add(item)	Adds a single item to the list. (This method is inherited from the Collection interface.)
addAll(item...)	Adds one or more items to the list, specified by the variable argument list.

c1ear()	Removes all of the items from the list.
remove(item)	Removes the object specified by item from the list. (This method is inherited from the Collection interface.)
removeAll (item...)	Removes one or more items to the list, specified by the variable argument list.
setAll(item...)	Clears the current contents of the list and adds all of the items specified by the variable argument list.
size()	Returns the number of items in the list. (This method is inherited from the Collection interface.)

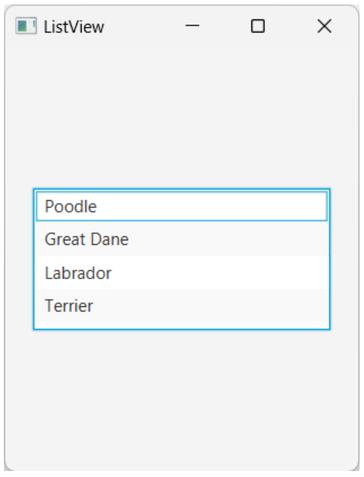
4.1.2 Creating and using ObservableList

An `ObservableList` has much in common with the `ArrayList` class that you learned already. If you need to work with the individual elements in an `ObservableList` , everything that you have already learned about `ArrayLists` also applies to `ObservableLists`. For example:

- You can use the enhanced `for loop` to iterate over an `ObservableList`.
- You can use the `get` method to get a specific element in an `ObservableList`.
- You can use the `size` method to get the number of elements in an `ObservableList`.
- You can use the `add` method to add an element to an `ObservableList` (as long as the `ObservableList` is not read-only).
- You can use the `remove` method to remove an element from an `ObservableList`.
- You can use the `set` method to replace an element in an `ObservableList`.

4.2 How to use ListView Controls

The ListView control displays a list of items and allows the user to select one or more items from the list. The ListView class is in the javafx.scene.control package.



4.2.1 How to use ListView Controls

- Using the IntelliJ IDE, create a new JavaFX project and named it **Exercise1**.
- Using Refactor, change the name of the FXML file to **ListView-view**.
- Using Refactor, change the name of the Application file to **ListViewApplication.**
- Using Refactor, change the name of the Controller file to **ListViewController.**
- Open the ListView-view.fxml file and delete its contents.
- Right-Click the ListView-view.fxml file and select Open In SceneBuilder.
- Add an VBox container to the scene.
  - Set the "**Pref Width**" field in the Inspector Panel Layout to **250**
  - Set the "**Pref Height**" field in the Inspector Panel Layout to **400**
- Drag the List View control and drop it inside VBox container. Notice that after the listView component is dropped, it is shown in the Hierarchy panel as a child of VBox element.
  - Set the "**Pref Height**" field in the Inspector Panel Layout to **100**
- Click the Code section of the Inspector panel. In the fx:id field, add the ID "lvItem".

- In the control class add the following that creates the ListView object.
- Use the IDE assistant to import the required JavaFX class for the ListView.

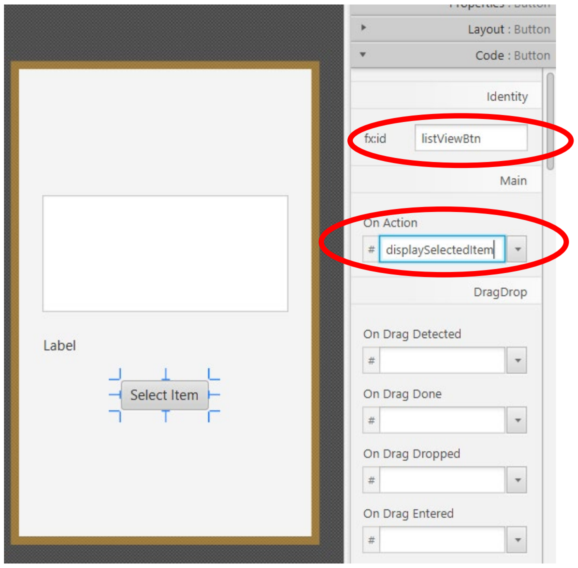
```
@FXML
private ListView<String> lvItem;
```

- Notice at the beginning of the statement, the notation <String> appears immediately after the word ListView . This specifies that this particular ListView control can hold only String objects.
- Once you have created a ListView control, you are ready to add items to it.
- The following statement shows how we can add strings to our dogListView control.
- In the **ListViewController** class add the intializable method that include the following statement to the ListView control to appear as shown in the Figure above.

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

- Note that, the ListView control keeps its list of items in an `ObservableList` object. We discussed the ObservableList interface in Section 4.1 above. The ListView class's `getItems()` method returns the ObservableList that holds the ListView 's items.

4.2.2 How to Retrieve a Selected Item



fx:id listViewBtn

On Action # displaySelectedItem

- You can use the ListView class's `getSelectionModel().getSelectedItem()` method to get the item that is currently selected.
- Let us first add a new button control in the VBox container named listViewBtn with a text "Select Item" and add a new label control named selectedItem with an empty text value.
- Define a listener method in the On Action field of the button control and name it displaySelectedItem, as shown in the Figure.
- In the **ListViewController** class, we will define the objects selectedItem label and the listViewBtn button. We will also implement the displaySelectedItem method so that when we select an item from the list and then clicke the button the selected item string value will be displaied in the selectedItem label.
- Add the following statements into **ListViewController** class.

```
@FXML
private Button listViewBtn;

@FXML
private Label selectedItem;

@FXML
void displaySelectedItem() {
    selectedItem.setText(listView.getSelectionModel().getSelectedItem());
}
```

- Note that, if no item is selected in the ListView , the `getSelectedItem()` method will return null.
- Note: to deselect an item, press and hold <Ctrl> key and click the item to deselect.

- The attached video “Lab2-Exercise1-1.mp4” shows how the program is running.
- Complete your code as described, run, and test the application, make sure the output is correct as shown in the solution video.
- Once you finished, select File → Export to Zip File…, the Save as pop-up window appears, click OK to save your project as Exercise1-1.zip file.

Notice: If you are unable to get this exercise run successfully, you should talk to your TAs during their announced office hours (the lab hours).

4.2.3 How to Retrieve the Index of the Selected Item

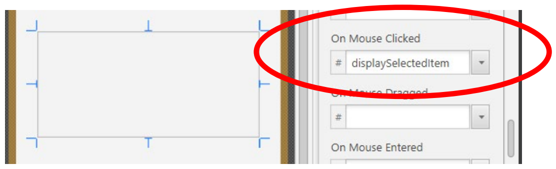
- Internally, each item in a ListView control is assigned an **index** that marks the item’s position. The first item (which is the item stored at the top of the list) has the index 0, the second item has the index 1, and so forth.
- You can use the ListView class’s `getSelectionModel().getSelectedIndex()` method get the index of the item that is currently selected.
- Let us now adjust our statements in the displaySelectedItem() to display the index of the selected item along with the item valu itself sperated by a dash.
- Replace the statement you have already in the displaySelectedItem() with the follwing statement.

```
selectedItem.setText(listView.getSelectionModel().getSelectedIndex() + "-" +
            listView.getSelectionModel().getSelectedItem());
```

- Note that, if no item is selected `getSelectedIndex()` method will return the value -1.
- Delete the listView Button, run, and test the application, make sure the output is correct as shown in the solution video.
- Complete your code as described, run, and test the application, make sure the output is correct as shown in the solution video “Lab2-Exercise1-2.mp4”.
- Once you finished, select File → Export to Zip File…, the Save as pop-up window appears, click OK to save your project as Exercise1-2.zip file.

4.2.4 How to Respond to Item Selection with an Event Handler

- When the user selects an item in a ListView , a change event occurs. If you want the program to immediately perform an action when the user selects an item in a ListView , you can write an event handler that responds to the change event.
- To do so, using the Scene Builder tool select the ListView Control and assign the listener method for “On Mouse Clicked” event field in the Code section of the Inspector panel.


On Mouse Clicked
displaySelectedItem
On Mouse Dragged
On Mouse Entered

- Delete the listView Button, run, and test the application, make sure the output is correct as shown in the solution video “Lab2-Exercise1-3.mp4”.
- Once you finished, select File → Export to Zip File…, the Save as pop-up window appears, click OK to save your project as Exercise1-3.zip file.

4.2.5 Adding Items versus Setting Items

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. As a result, the ListView will contain the old items plus the new items.

Sometimes this is the behavior you want, but in other situations, you might want to replace the existing contents of a `ListView` with an entirely new list of items. When that is the case, you can use the `getItems().setAll()` method. The `getItems().setAll()` method replaces a ListView control's existing items with a new list of items.

4.2.6 How to Initialize a ListView with an Array or an ArrayListItems

In some situations, you might have an array or an ArrayList containing items you want to display in a ListView control. When this is the case, you can convert the array or ArrayList to an ObservableList , then pass the `ObservableList` as an argument to the ListView class's constructor.

The `FXCollections` class (which is in the javafx.collections package) has a static method named observableArrayList that takes an array or an ArrayList as an argument, and returns an ObservableList containing the same items.

- The following code shows an example of creating a ListView control that displays the items in a String array (we assume that the ListView item is already created using the scene builder).

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

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

// Populate the ListView control:
listView.getItems().addAll(strList);
```

- The following code snippet shows the same example, but this time converting an `ArrayList` to an `ObservableList`:

```
// 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);

// Populate the ListView control:
listView.getItems().setAll(strList);
```

- Note that, most of the time, we populate the ListView control in the `initalize` methods.

- Sometimes, you might need to get the ObservableList of selected items from a ListView control, then convert that ObservableList to an array.
- The following code snippet shows the same example, but this time converting an `ArrayList` to an `ObservableList`:

```
// Get the selected items as an ObservableList.
ObservableList<String> selections = listView.getSelectionModel().getSelectedItems();

// Convert the ObservableList to an array of Strings.
String[] itemArray = selections.toArray(new String[0]);
```

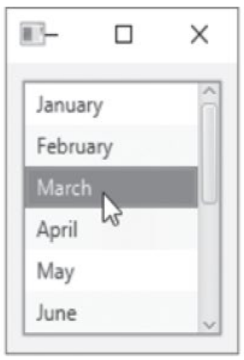
- `ObservableList` objects have a method named `toArray` that returns an array containing the items in the list. You specify the data type of the retured array by passing an empty array of the correct data type as an argument to the toArray method.
- In this code, the first statement gets an ObservableList containing the selected items in the listView control. The second statement converts the ObservableList to a String array, and assigns the array to the itemArray variable. The empty String array we are passing as an argument to the toArray method tells the method to return a String array.

4.2.7 How to use Multiple Interval Selection Mode ListView

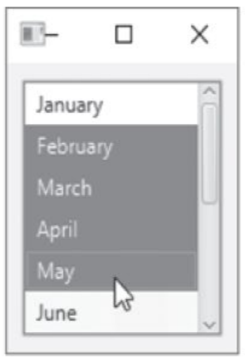
The ListView control can operate in either of the following selection modes:

- **Single Selection Mode**. In this mode, only one item can be selected at a time. When an item is selected, any other item that is currently selected is deselected. This is the default selection mode.
- **Multiple Interval Selection Mode**. In this mode, multiple items may be selected. To select multiple items that are consecutive, the user clicks the first item, holds down the Shift key on the keyboard, then clicks the last item. To select items that are not consecutive, the user holds down the Ctrl key on the keyboard (or the Command key on a Mac keyboard), then clicks each item.

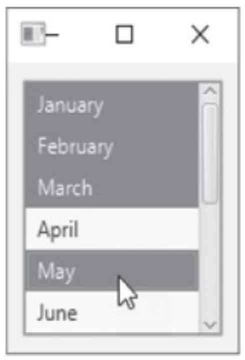
The Figure below shows examples of a ListView control in single selection mode and multiple selection mode.



In single selection mode, only one item can be selected at a time.



In multiple selection mode, multiple consecutive items can be selected.



In multiple selection mode, multiple nonconsecutive items can be selected.

- By default, the ListView control is in single selection mode. You change the selection mode with the control's `getSelectionModel().setSelectionMode()` method.
- The method accepts one of the following enum constants:
    - `SelectionMode.MULTIPLE`
    - `SelectionMode.SINGLE`
- Assuming that listView is the name of a ListView control, the following code shows how to change the control to multiple selection mode:

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

4.2.8 How to Retrieve Multiple Selected Items

When a ListView control is in multiple selection mode, the user can select more than one item. The `getSelectionModel().getSelectedItem()` method will return only the last item selected. Likewise, the `getSelectionModel().getSelectedIndex()` method will return only the index of the last item selected. To get all of the selected items and their indices, you need to use the following methods:

- `getSelectionModel().getSelectedItems()`— This method returns a read-only ObservableList of the selected items
- `getSelectionModel().getSelectedIndices()`— This method returns a read-only ObservableList of the integer indices of the selected items:

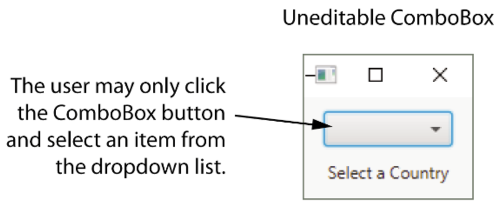
- Now revise your code in the Exercise1 to populate your listView control using `ArrayList` object.
- Define the `listView` control for Multiple Selection mode, such that your program must behave as demonstrated in Lab2-Exercise1-4.mp4 video.
- Once you finished, select File → Export to Zip File…, the Save as pop-up window appears, click OK to save your project as Exercise1-4.zip file.

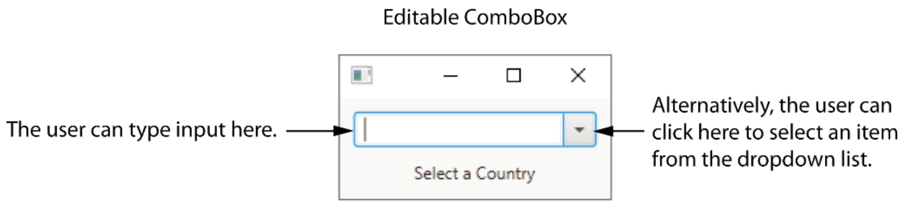
- **Notice:** If you are unable to get this exercise run successfully, you should talk to your TAs during their announced office hours (the lab hours).

4.3 How to use ComboBox Controls

- A `ComboBox` allows the user to select an item from a drop-down list. A `ComboBox` presents a list of items from which the user may select. Unlike a `ListView` , a `ComboBox` presents its items in a drop-down list.
- You create a `ComboBox` control with the `ComboBox class` (which is in the `javafx.scene.control package`).
- Just like the `ListView` control, the `ComboBox` control keeps its list of items in an `ObservableList` object.
- Also, the `getItems()` method returns an `ObservableList` that holds the ComboBox 's items.
- The `ComboBox` control is populated using the methods as in ListView control. For instance, we can use the `getItems().setAll()` method and the `getItems().setAll()` method.

- Just like the `ListView` control, the selected item can be retrieved using the method `getValue()`.
- When the user selects an item in a `ComboBox`, the "OnAction" event occurs. If you want the program to immediately perform an action when the user selects an item in a `ComboBox`, you can use the Scene Builder tool to assign the listener method for "OnAction" event field in the Code section of the Inspector panel.
- By default, ComboBox controls are **uneditable**, the user cannot type input into the control; the user can only select items from a drop-down list. An **editable** ComboBox allows the user to select an item, or type input into a field that is similar to a `TextField`.


Uneditable ComboBox
The user may only click the ComboBox button and select an item from the dropdown list.
Select a Country


Editable ComboBox
The user can type input here.
Alternatively, the user can click here to select an item from the dropdown list.
Select a Country

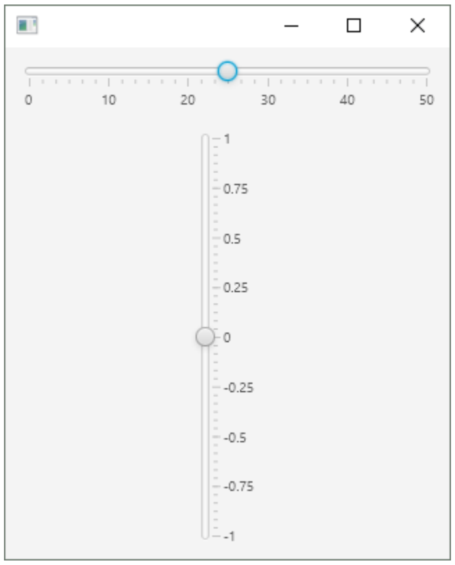
- Use the `setEditable` method to make a ComboBox editable:

```
comboBox.setEditable(true);
```

4.3.1 Working with ComboBox Controls

- Using the IntelliJ IDE, create a new JavaFX project and named it **Exercise2**.
- Using Refactor, change the name of the FXML file to **ComboBox-view**.
- Using Refactor, change the name of the Application file to **ComboBoxApplication.**
- Using Refactor, change the name of the Controller file to **ComboBoxController.**
- Open the ComboBox-view.fxml file and delete its contents.
- Open the ComboBoxController file and delete its contents.
- Right-Click the ComboBox-view.fxml file and select Open In SceneBuilder.
- Add an SplitPane (vertical) container to the scene.
  - Set the "**Pref Width**" field in the Inspector Panel Layout to **310**
  - Set the "**Pref Height**" field in the Inspector Panel Layout to **310**
- Add the labels controls and the ComboBox control into your program such that your program must behave as demonstrated in Lab2-Exercise2.mp4 video.
- Once you finished, select File → Export to Zip File…, the Save as pop-up window appears, click OK to save your project as Exercise2.zip file.

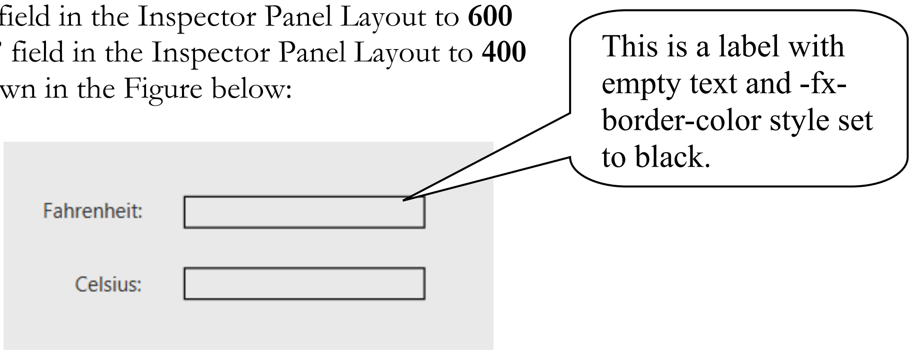
4.4 How to use Slider Controls



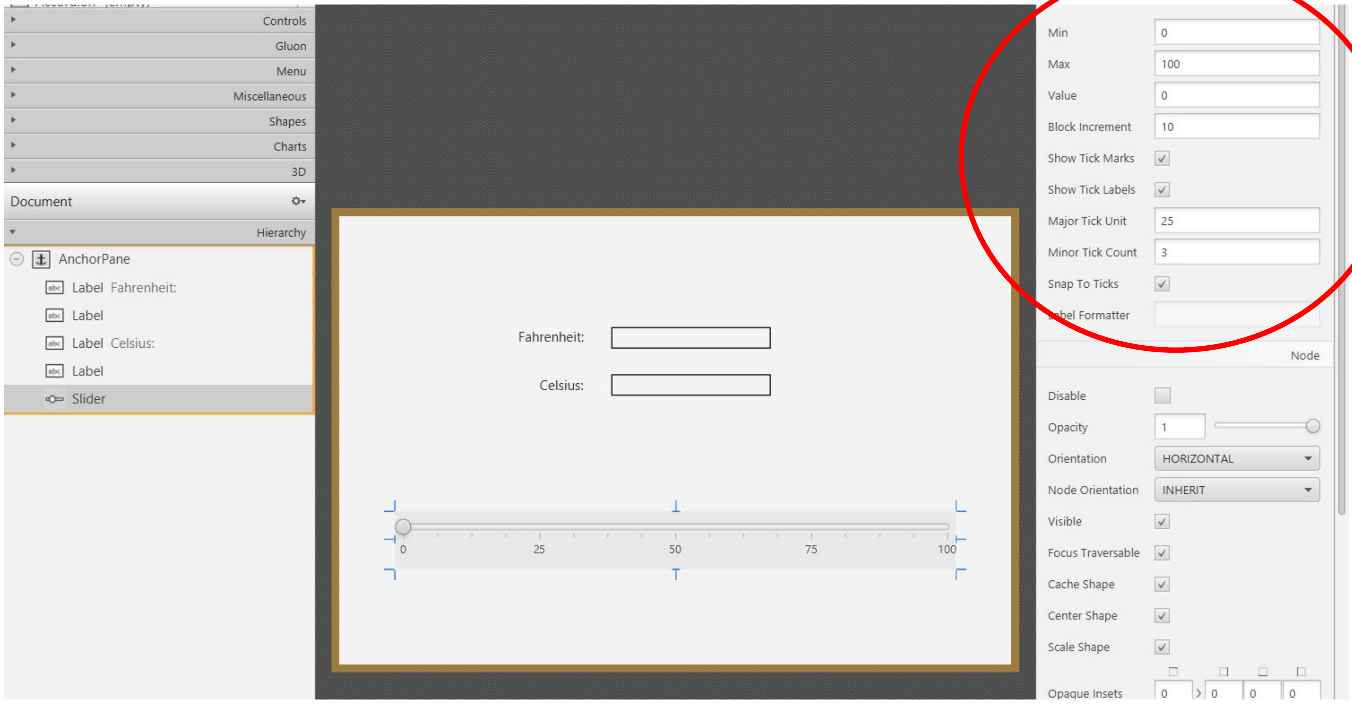
- A `Slider` allows the user to graphically adjust a number within a range of values. Sliders are created from the `Slider` class (in the `javafx.scene.control` package)
- They display an image of a "slider knob" that can be dragged along a track. Sliders can be horizontally or vertically oriented, as shown in Figure.
- A `Slider` is designed to represent a range of numeric values. At one end of the Slider is the range's minimum value, and at the other end is the range's maximum value.
- In the Figure, the horizontal Slider 's minimum value is 0, and its maximum value is 50. The vertical Slider 's minimum value is -1, and its maximum value is 1.
- `Slider` controls have a numeric value, and as the user moves the knob along the track, the numeric value is adjusted accordingly.
- Notice the Slider controls in this Figure have accompanying tick marks. Starting at 0, and then at every 10th value, a major tick mark is displayed along with a label indicating the value at that tick mark. Between the major tick marks are minor tick marks. In this example, there are five minor tick marks displayed between the major tick marks.
- When a Slider 's value changes (because the knob was moved), the Slider control generates a `change` event. If you want an action to take place when the Slider 's value changes, you can register an event handler for the change event.
- Just like the `ListView` control, the current slider's value can be retrieved using the method `getValue()` which returns a double value.

4.4.1 Working with Slider Controls

- Using the IntelliJ IDE, create a new JavaFX project and named it **Exercise3**.
- Using Refactor, change the name of the FXML file to **Slider-view**.
- Using Refactor, change the name of the Application file to **SliderApplication.**
- Using Refactor, change the name of the Controller file to **SliderController.**
- Open the Slider-view.fxml file and delete its contents.
- Open the SliderController file and delete its contents.
- Right-Click the ComboBox-view.fxml file and select Open In SceneBuilder.
- Add an AnchorePane container to the scene.
  - Set the "**Pref Width**" field in the Inspector Panel Layout to **600**
  - Set the "**Pref Height**" field in the Inspector Panel Layout to **400**
- Add four label controls as shown in the Figure below:


Fahrenheit:
Celsius:
This is a label with empty text and -fx-border-color style set to black.

- Add a Slider control to the AnchorePane container with the following size.
    - Set the "**Pref Width**" field in the Inspector Panel Layout to **500**
    - Set the "**Pref Height**" field in the Inspector Panel Layout to **50**
- Set the properties of the Slider control as shown below:



AnchorPane
- Label Fahrenheit:
- Label
- Label Celsius:
- Label
- Slider

Min	0
Max	100
Value	0
Block Increment	10
Show Tick Marks	✓
Show Tick Labels	✓
Major Tick Unit	25
Minor Tick Count	3
Snap To Ticks	✓
Label Formatter	
Disable	
Opacity	1
Orientation	HORIZONTAL
Node Orientation	INHERIT
Visible	✓
Focus Traversable	✓
Cache Shape	✓
Center Shape	✓
Scale Shape	✓
Opaque Insets	0 0 0 0

- Create a listener method by which you will get the value of the slider, set it into the Celsius label box and then convert it to Fahrenheit and display it in the Fahrenheit label box.
- Use this conversion equation:

```
    double fahrenheit = (9.0 / 5.0) * celsius + 32;
```

- Associate this listener method to the following events in the Code section of the Inspector panel.
    - onDragDetected
    - onInputMethodTextChanged
    - onMouseClicked
    - onMouseDragged
- Complete your code as described, run, and test the application, make sure the output is correct as shown in the solution video "Lab2-Exercise3.mp4".
- Once you finished, select File → Export to Zip File…, the Save as pop-up window appears, click OK to save your project as Exercise3.zip file.

5 Hand In

- Group the following zip files into one zip file and name it *yourUwoId*_ Lab2.zip. For example, if your email address is aouda@uwo.ca, then name the archive file as aouda_Lab2.zip.
    – Exercise1-1.zip, Exercise1-2.zip, Exercise1-3.zip, Exercise1-4.zip, Exercise2.zip, and Exercise3.zip
- Submit your final zip file through OWL on the due date mentioned above, to be graded out of 20.