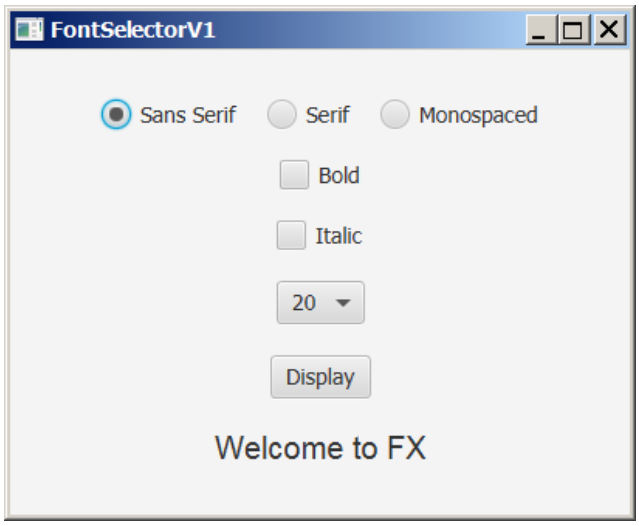
CSE3OAD/CSE4OAD – Lab 2

Java FX Layouts and Basic Controls

Question 1– Font Selector (Version 1)

Write a program, called FontSelectorV1, which when run, displays the initial screen shown below.



The screen allows us to select

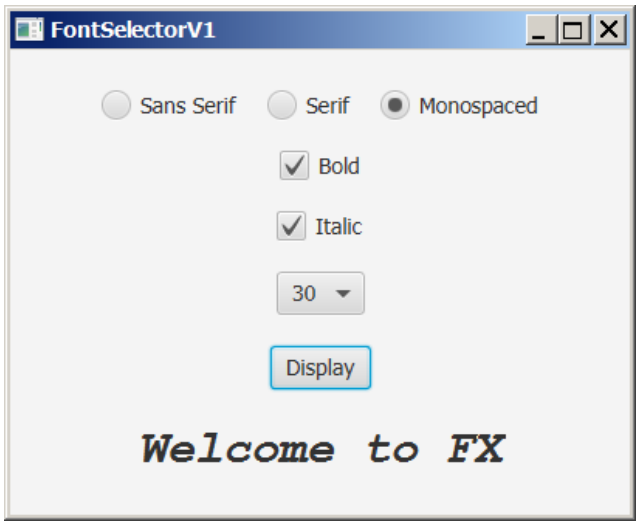
- The font family, using the three radio buttons on row 1
- The font weight, using the check box on row 2. If the check box is checked, the font weight is BOLD. Otherwise, it is NORMAL
- The font style, using the check box on row 3. If the check box is checked, the font posture is ITALIC. Otherwise, it is NORMAL
- The font size, using the choice box on row 4. The values go from 10 to 50 in step size of 5.

The last row displays a message (a label) in the font determined by the choices set by the buttons.

Initially, as can be seen, the font family is sans serif, font weight normal, font style normal and font size 20.

To display the message in a different font, the user makes the appropriate choices, and then press the Display button.

The screen below shows an example.



HINT: Use JavaFX CSS rules to set font family, font size, font style and font weight. Consult the JavaFX CSS Reference (available on the Web) whenever necessary.

Question 2– Font Selector (Version 2)

(More Challenging)

Modify the program so that we do not need the Display button any more. Each time we make a change to a check box, a radio button or the choice box, the message will be displayed in the new font.

Hint: A check box has what is known as the `selected` property (an instance of `Property`), a radio button has the `selected` property, and the choice box has the `value` property. To each of these properties, we can add an event listener (event handler), which is known as a change listener. The listener will listen and respond whenever a change of state is made to the event source. Consult the API or google search for the details.

Recommendation: After Question 1, you should move on to Question 3, and do this question later.

Question 3– Calculator (version 1)

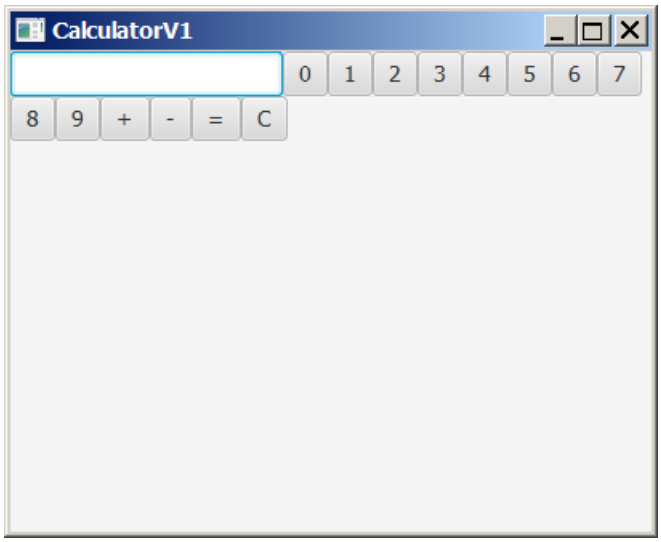
(Last question of Lab 1 revisited)

Write an application (call it "CalculatorV1") that displays a window containing the following:

- 10 buttons representing the numbers 0-9.
- Buttons with each of the following symbols: +, -, *, /, =, C.
- Label (empty to start).

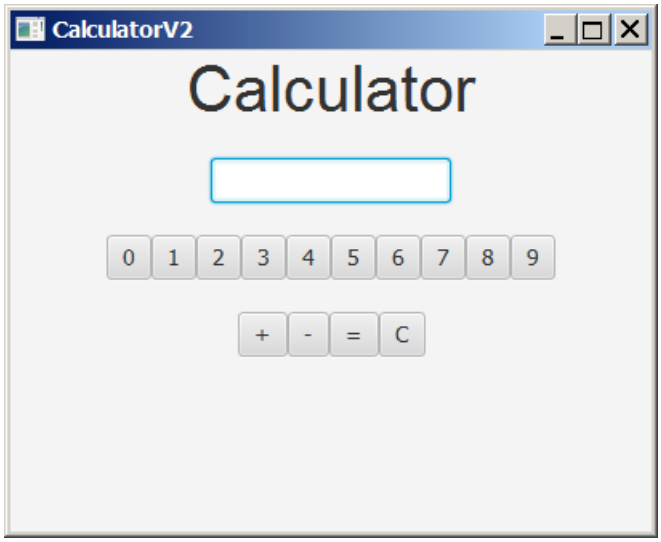
Implement appropriate events for each button to make the application behave as a calculator.

For part a), don't worry about the layout of the window elements. The displayed window may look like the diagram shown below.



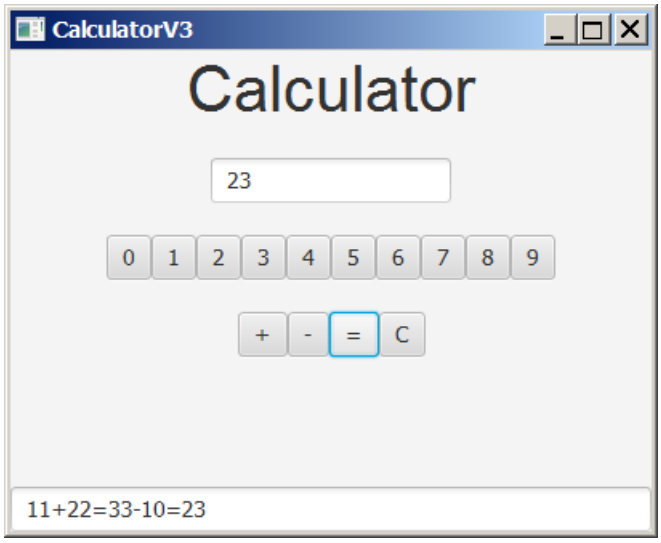
Question 4– Calculator (Version 2)

Make a copy of the previous program (call it "CalculatorV2") and modify it so that the user-interface controls are arranged as shown in the diagram below.



Question 5– Calculator (Version 3)

Make a copy of the previous program (call it "CalculatorV3"). Modify it so that the user-interface controls are arranged as shown in the diagram below.



Note that at the bottom of the screen, we have a text that captures each of the clicks, as well as the result of the calculations. This text field is cleared only when we click the "C" (clear) button.

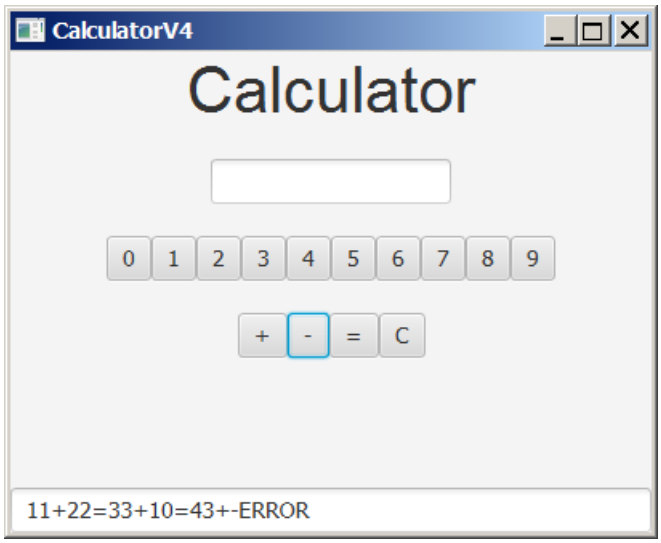
HINT: The following code shows how to place a control at the bottom of the screen. It creates a region and let this region grows to occupy the extra space.

```
public void addContents(Stage stage)
{
   // Define labels
   // and place label3 at the bottom of the vbox
   //
   Label label1 = new Label("LABEL 1");
   Label label2 = new Label("LABEL 2");
   Label label3 = new Label("LABEL 3");
   Region spacer = new Region();
   VBox box = new VBox(label1, label2, spacer, label3);
   box.setVgrow(spacer, Priority.ALWAYS);

   // Set scene and stage
   Scene scene = new Scene(box, 400, 300);
   stage.setScene(scene);
}
```

Question 6– Calculator (Version 4)

Make a copy of the previous program (call it "CalculatorV4"). Modify it so that the user-interface controls are arranged as shown in the diagram below.



In this version, when the user takes an invalid action, the message "ERROR" is displayed in the text field at the bottom. For example, for the screen shown above, an error results because user has pressed "+" and "-" in succession.

HINT: The following code shows how we can catch and exception and display a message about it on the console window.

```
public void addContents(Stage stage)
{
   ...

   // Set scene and stage
   Scene scene = new Scene(root, 400, 300);
   stage.setScene(scene);

   // Catch exceptions
   // Thread.currentThread is the application thread
   //
   Thread.currentThread().setUncaughtExceptionHandler((thread,
      exception) ->
     {
       System.out.println("ERROR: " + exception);
     });
}
```

■