COMP8710 Assessment 2 – Calculator using JavaFX

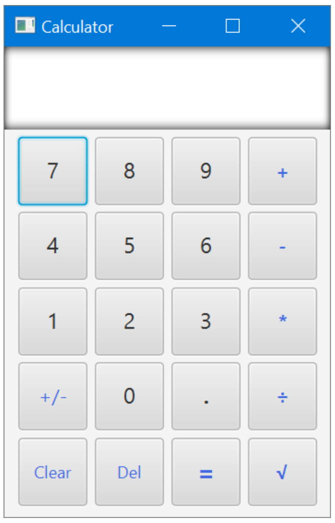
Introduction

You are expected to complete the implementation of a given JavaFX project. It simulates a calculator with four binary operations, i.e. addition, subtraction, multiplication and division, and two unary operations, i.e. the square root and negation of an input number.

You should download the project file, `A2-Calculator.zip`, from the module Moodle page and unzip the file. And then open the project in IntelliJ Idea and run the main method. It should open the calculator GUI window as shown below.

You should study the given classes in the project, i.e. `Main.java`, `Model.java` and `Controller.java`, and the `GUI.fxml` file as well, to make sure that you understand well what they are and how they work. *Please do ask me if you have any queries.*

Please note:

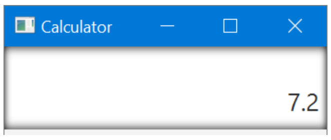


- For this assessment you are required to use the given `Model` class. It has three fields for you to store user inputs: `previousOperand`, `currentOperand` and `operator`. It also provides methods for you to do binary and unary calculations. *You should not make any changes in this class.*
- You should not make any changes in `Main.java` and `GUI.fxml`.
- You should write your Java code in the `Controller` class to replace `//TODO` in each of the event handling methods. You may define your own private helper methods to eliminate code duplication and make your code easy to understand and maintain. *You should not change the given code in this class.*
- Rather than implement all tasks in one go, you should try to do one task at a time and test it thoroughly to make sure it works as expected.
- Your mark will be capped at 60% if your code has syntax errors and doesn't compile with the given `Main.java`, `Model.java` and `GUI.fxml`.

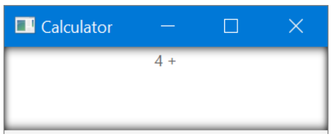
Submission

You should upload the file `Controller.java` using the submission link on the module Moodle page before the submission deadline. *A mark of zero is normally awarded to late or non-submission.*

Basic implementation requirements (42%)

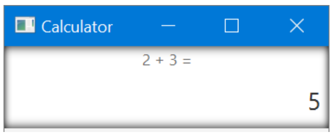


- When clicking on each of the digit 0-9, and the dot "." buttons, its value should be stored in the `currentOperand` field of `Model`. It should also be displayed in the label named lblDisplay. e.g. clicking on the buttons "7", "." and then "2", it should display "7.2" in lblDisplay. *Note: lblDisplay is bund with the field display.*



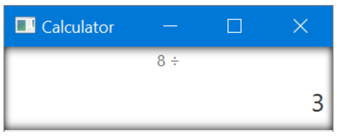
- When clicking on a binary operation button after the first input number, it should update the fields `previousOperand` and `operator` of `Model` accordingly. The field `currentOperand` should be set empty. It should also show the formula in the label named lblFormula. The label lblDisplay should be cleared and ready for user to enter the second input. e.g. clicking on the buttons "4" and "+", it should show "4 +" in lblFormula, but nothing in lblDisplay. *Note: lblFormula is bund with the field formula.*

- When clicking on the "=" button after two inputs and a binary operator, it should do the calculation and store the result in `currentOperand` of `Model`. Both `operator` and `previousOperand` should be set empty. It should also update the formula and display the result. e.g. clicking on the buttons "2", "+", "3" and then "=", it should show "2 + 3 =" in lblFormula and the result "5" in lblDisplay.



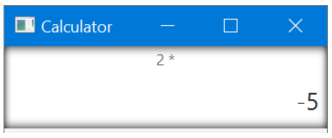
```
2 + 3 =
5
```

- Clicking on the "Del" button should empty the label lblDisplay. The `currentOperand` of `Model` should be updated accordingly.
- Clicking on the "Clear" button should empty both lblDisplay and lblFormula and reset the fields of `Model`.
- Clicking on the button "√" (square root) after an input should calculate and display the result in lblDisplay. The field `currentOperand` of `Model` should be updated accordingly. e.g. clicking on the buttons "8", "÷", "9" and then "√", it should show the result "3" in lblDisplay.



```
8 ÷
3
```

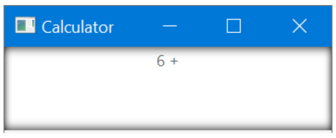
- Clicking on the button "+/-" should negate the input in lblDisplay and update the relevant field of `Model` accordingly. e.g. after clicking on the buttons "2", "*", "5" and then "+/-" it should show the result "-5 " in lblDisplay.



```
2 *
-5
```

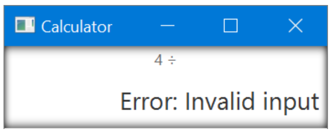
Further implementation requirements (50%)

- User input should be displayed as a valid numerical value and it should look like a real number. e.g. an input with only a dot should be displayed as "0." instead of ".". There are other cases to be checked as well.
- Users should be able to perform more complex calculations. e.g. clicking on the buttons "2", "*", "3", and then "+", it should show in lblFormula "6 +", i.e. the result of the first operation (i.e. 2*3) and the second operator "+". Also lblDisplay should be empty for user to enter the next input. The fields of `Model` should be updated accordingly.



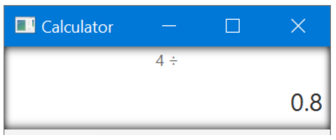
```
6 +
```

- You should make your calculator more robust. You should test it thoroughly and address any issues that may cause a runtime crash. e.g. it should do nothing when clicking on the button "+" without the first input number.
- It should show the message "Error: Invalid input" if the result is not a valid number. e.g. clicking on the buttons "4", "÷", "0", and then "=", it should show the error message in lblDisplay. The `currentOperand` of `Model` should be updated appropriately.



```
4 ÷
Error: Invalid input
```

  Subsequent clicking on any binary/unary operation buttons should do nothing with the error message in display, but clicking on a number or the dot button it should replace the error message with the number. e.g. clicking on the buttons "." and then "8" now, the error meesgae should be replaced by "0.8" in lblDisplay.



```
4 ÷
0.8
```

Coding quality (8%)

- Your code should compile without any syntax error.
- Your code should be well indentied with appropriate comments.
- You should elminate code duplications if possible. Your code should be easy for others to understand, maintain, and reuse.