Lab 6: Calculator in GUI

Due: 18:00, 15 Mar 2012 (Fri) Full marks: 100

Introduction

In this lab, *set your NetBeans project name as* ***Calculator*** *and the class name of your* **JFrame** *form as* **CalculatorGUI**. You will write a GUI application that simulates a simple calculator. Figure 1 shows the desired user interface.

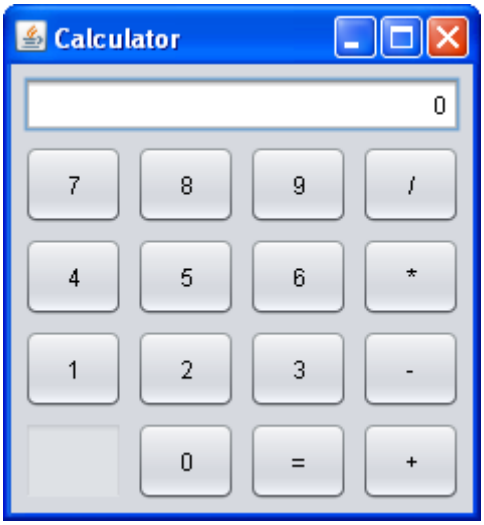

Figure 1: GUI of Calculator

- The *title* of the application frame is "Calculator".
- The calculator has *a text field* at the top for displaying calculation result, *10 buttons* for the digits 0–9, *five buttons* for the arithmetic operators + - * / =, and *one text field* at the bottom left for displaying the lastly pressed operator. For simplicity, decimal point input is *not* supported.
- The display text field is *non-editable*. The text within should be *horizontally aligned to the right* and initialized as "0".
- The operator text field at bottom left is *not enabled*. The text within should be *horizontally aligned to the center* and initialized as "" (empty).
- User inputs are by *pressing the buttons only*. Keyboard inputs are *not* needed.
- For simplicity, arithmetic operators are always evaluated from left to right. Operators * and / have *no* precedence over + and -. For example, if a user presses the keys "35+17*4/5=", then the calculation result is 41.6 instead of 48.6. Table 1 shows the states of the calculator for this input. Note that although decimal point input is not supported, floating point numbers can still occur due to the arithmetic operations.

- Figure 2 shows the state diagram of the calculator. For example, the key presses "35+17*4/5=" go through the states ABBCDDCDCDE.
- Once a calculation finishes after pressing '=' (state E), the user can start another calculation by pressing either another digit or operators + - * /. In the former case, the newly pressed digit is the new operand 1, while in the latter case; the previous calculation result is treated as operand 1.
- When a user presses some invalid inputs, a dialog should be prompted *immediately* to show a warning message. After closing the dialog, the calculator remains in the same state as before the invalid input is pressed. For example, "1++" goes through the states ABC but cannot proceed any more. The warning dialog is prompted once the second "+" is pressed. After closing the dialog, the calculator is still in state C, waiting for the input of operand 2.

Table 1: State of Calculator for Input "35+17*4/5="

Pressed key	**Next State**	**Operand 1**	**Last operator**	**Operand 2**	**Display**
	A				
3	B	3			3
5	B	35			35
+	C	35	+		35
1	D	35	+	1	1
7	D	35	+	17	17
*	C	52	*	17	52.0
4	D	52	*	4	4
/	C	208	/	4	208.0
5	D	208	/	5	5
=	E	41.6	=	5	41.6

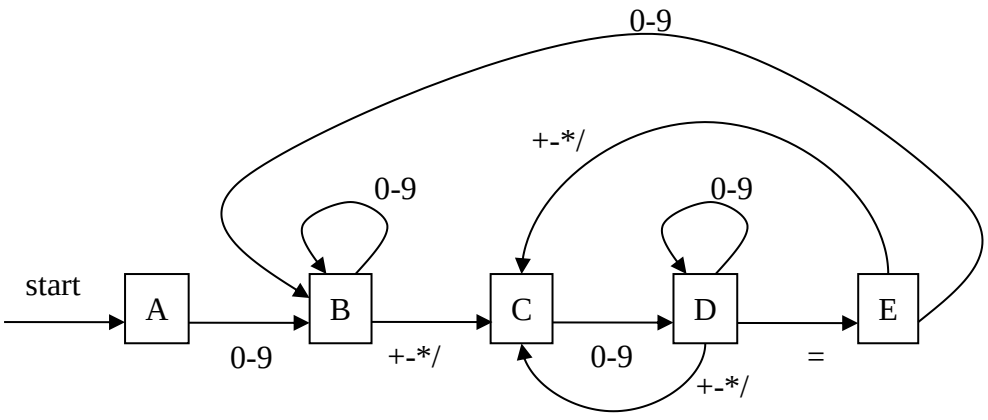

Figure 2: State Diagram of the Calculator

Program Specification

Class **CalculatorGUI**

This class represents both the GUI and the application logic of the calculator. Once your GUI is built, you should have several members defined in the class already. The following shows the *extra* members that you need for the application logic. You are *not allowed to add any more non-*`private` *contents*.

Private Instance Variables

- `private char state;`
  A character denoting the current state of the calculator in the state diagram. It should be either 'A', 'B', 'C', 'D', or 'E'.

- `private double operand1, operand2;`
  Operands 1 and 2 of the calculator. See Table 1 for their use.

- `private char operator;`
  A character denoting the last pressed operator of the calculator. It should be either '+', '-', '*', '/', or '='. See Table 1 for its use.

Constructor and Instance Methods

- `public CalculatorGUI()`
  This constructor is already provided to you by NetBeans once you built your GUI. You just need revise it to initialize the extra instance variables `state` as 'A', `operand1` and `operand2` as 0.0, and `operator` as '='.

- `private void digitPressed(int d)`
  This *private* method should be called by the event handling methods of the *digit buttons*. Parameter `d` is the digit on the button just pressed. You can assume that `d` *is always 0 to 9 (inclusive)*. In this method, depending on the current state, you should update (a) the state if needed, (b) operands 1 or 2, and (c) the display text field of the calculator accordingly. See Table 1 and Figure 2 for example.

- `private void operatorPressed(char op)`
  This *private* method should be called by the event handling methods of the

operator buttons. Parameter `op` is the operator on the button just pressed. You can assume that `op` *is always '+', '-', '-', '/', or '='*. In this method, depending on the current state, you should update (a) the state if needed, (b) operand 1, (c) the last pressed operator and its corresponding text field, and (d) the display text field of the calculation (i.e. format to one decimal place only) accordingly. In case of *invalid user input* (such as pressing the + button in state C), you should *prompt a warning dialog* using:

```
JOptionPane.showMessageDialog(this, "Invalid input!");
```

Note: the use of `this` instead of `null`. Again, see Table 1 and Figure 2 for example. There should be **no state change** and **no change on any text field** after invalid input.

Submission

Submit the source file `CalculatorGUI.java` to CU e-Learning System (i.e. the entry "**lab03 – Calculator in GUI**" under "**Lab Works**").

