# CS102 – Algorithms and Programming II
# Lab Programming Assignment 4
# Fall 2021

**ATTENTION:**

1. Compress all of the Java program source files (.java) files into a single zip file.
2. The name of the zip file should follow the below convention:
   **CS102_Sec1_Asgn4_YourSurname_YourName.zip**
3. Replace the variables "YourSurname" and "YourName" with your actual surname and name.
4. You may ask questions on Moodle.
5. Complete **all** of the assignment before the lab session and upload the above zip file to Moodle by the deadline (otherwise, significant points will be taken off). You will get a chance to update and improve your solution by consulting the TA during the lab. You will resubmit your code once you demo your work to the TA.
6. The deadline is at **16 November 2021, Tuesday 23:55**.
7. You have to come to the lab session on 17 November 2021, Wednesday to demonstrate your work.

**The work must be done individually. Codesharing is strictly forbidden. We are using sophisticated tools to check the code similarities. The Honor Code specifies what you can and cannot do. Breaking the rules will result in disciplinary action.**

In this lab, we are going to walk you through implementing a calculator app. This will be an exercise for you to practice a few of the Object-Oriented Principles, including inheritance. Importantly, you are not allowed to use the **instanceof** operator at any place. It is most of the time a bad practice to use it and defies the purpose of inheritance.

## Overview

To begin with, the final edition of the software will look like the below image:

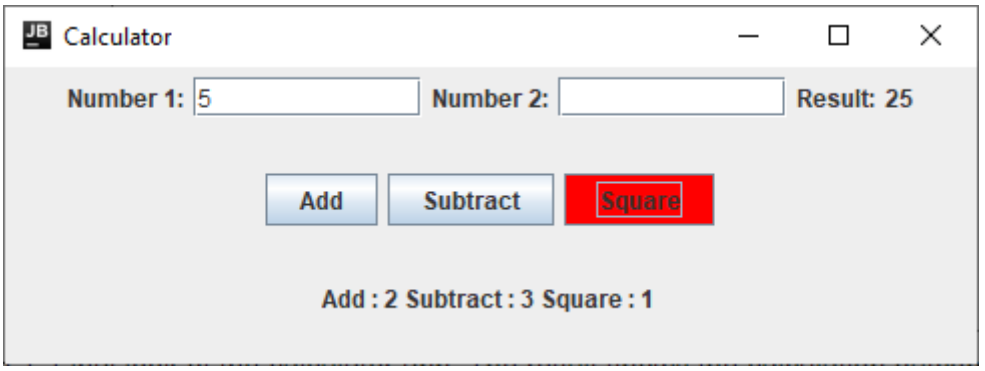


Figure 1: Final look of the calculator app. The result shows the outcome of square operation with the given input.

- It has two text fields for two inputs of the calculator that are **Number 1** and **Number 2**. Next to the fields is a label showing the final result of the calculation. There are three operations defined in the example, but we expect you to create more operations and add them here.
- It should support two types of operations: Binary and Unary. Binary operations (i.e., addition, subtraction) accept two inputs while unary operations (i.e., square, square root) accept a single input. You will assume that for the unary operation, the user will only use the **Number 1** field. So, in that case, you should ignore the **Number 2** field.
- At the bottom of the screen, you will show users the number of times each operation is **successfully** executed.

With the introduction being done, let's move on to the implementation details of this software.

# Implementing Math Operations

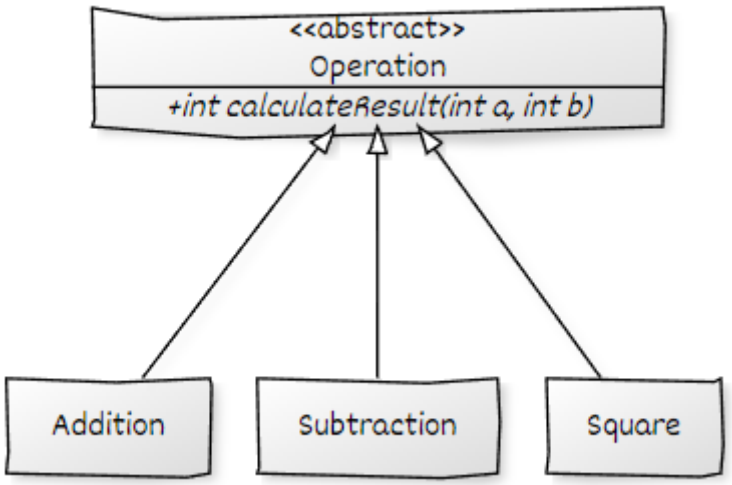


Figure 2: Class Diagram for operation classes. Variables and non-abstract methods are omitted in the diagram.

First, you will need to create a class hierarchy, as shown in the above class diagram. If you are not familiar with such diagrams it says the following:

- You have an abstract class called Operation, which has an abstract public method called calculateResult.
- You have non-abstract classes called Addition, Subtraction, and Square extending Operation class.

You can add additional methods to **only** Operation class if you want.

**Operation** class should have at least these 3 variables. Those variables should not be public. You may create constructors, setters, and getters if needed:

- an integer keeping track of how many times this operation is called
- a boolean specifying if this operation is unary or binary
- a String corresponding to the name of the method (i.e., for Addition, it should be Add)

You may add extra variables to the **Operation** class if you need them. Do **NOT** add any additional variables to sub-classes.

You will need to do the following things:

- Create the aforementioned class hierarchy, and for each subclass (Addition, Subtraction, Square, and more) implement the required method. Implement at least **8** operations, including Addition, Subtraction, and Square. Specifically, have at least **4** binary and **4** unary operations eventually.
- **calculateResult** method will calculate the outcome of the operation and return it. Here, you may also consider updating the variable counting number of times an operation is executed.
- At this point, you may find it redundant to create a separate class for each operation (add, subtract, etc.). You may think of creating a single class and creating a method for each operation. However, the latter approach violates the “Open–closed principle” of Object-Oriented Programming. The principle imposes that each class should be closed for modification but open to being extended (or inherited). This is why we are encouraging you to create such a hierarchy of classes.

## Implementing Button Logic

You have completed the core logic of the software. If you realized, while implementing this, you did not consider the user interface at all. Let’s jump into it then.

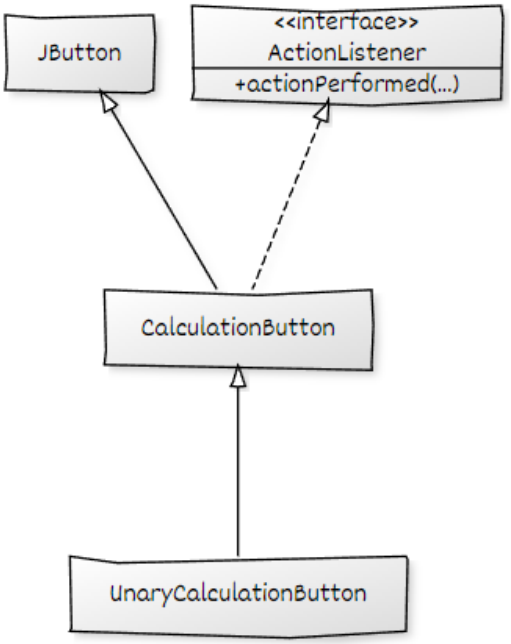


Figure 2: Class Diagram for button classes. Variables and the methods except for the ones in the listener are omitted.

This time, you will focus on the above class diagram. The diagram says the following:

- You have JButton class (it is Java’s class, you will directly use it)
- You have a CalculationButton class that extends JButton class and implements the ActionListener interface. (ActionListener interface is provided by Java itself, you should directly use it)
- You have a UnaryCalculationButton class that extends the Calculation Button. Likewise, this class will automatically be an implementor for the ActionListener.

Here, the CalculationButton corresponds to binary operations, while UnaryCalculationButton to unary operations.

Again, begin with creating the class hierarchy above.

**CalculationButton** should at least have the following variables. Those variables should not be public. You may create constructors, setters, and getters if needed:

- a reference to an Operation object
- a reference to the **Number 1** field (JTextField object)
- a reference to the **Number 2** field (JTextField object)
- a reference to the **result** label (JLabel object)
- a reference to **CountInformer** object. That class is detailed in the next section.

You may add extra variables if you need them. Do **NOT** add any additional variables to the subclass of **CalculationButton**.

You will need to do the following things:

- As you may know, the **actionPerformed** method of ActionListener class will be called when this button is pressed (Hint: You may need to call addActionListener(...) method at some point). That is, by implementing that method, you are specifying what this method will do when it is clicked.
- The **actionPerformed** method will get the values from the text fields (**Number 1** and **Number 2)** and call the **calculateResult**(...) method of the operation. Note that you should implement different **actionPerformed** methods for CalculationButton and UnaryCalculationButton.
- You may wonder why we need the **informer** object. The motivation is that in this design, we give the responsibility of adjusting some parts of the UI to the panel class (which we will introduce shortly). Once a button is clicked, this class will inform the panel classes that an operation button is clicked. That way, the panel class will be noticed and update the counts of each operation.
- The UnaryCalculationButton class will extend the CalculationButton and make necessary changes to support unary operations.

## Combining Math & Button Logic

Let's move on to the last component of the project. Until now, you have designed core logic (Operation) and the button logic. You will need to combine these two sets of classes. To this end, you will use the panel classes. You are already familiar with the panels (i.e., JPanel) from the previous lab assignment. You will work on them here.

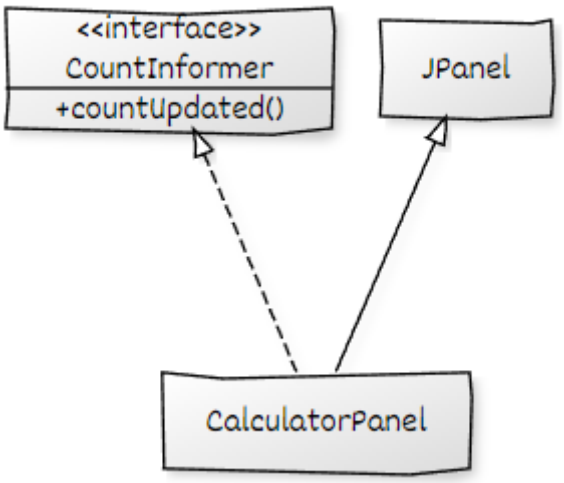


Figure 3: Class Diagram for panel classes. Variables and the methods except for the ones in the listener are omitted.

The above figure shows the class diagram of the panel classes. As you know, the JPanel is from Java itself. You will extend it and create a CalculatorPanel class. CalculatorPanel will also implement the CountInformer interface.

In the **CalculatorPanel** class, you will need to create at least the following member variable:

- an array of Operation objects (i.e. Operation[] x;)

Again, you may add additional variables here.

In this class, you will do the following things:

- Create all of the UI elements (i.e., buttons, text fields, labels)
- Create necessary Operation class instances (i.e., Operation x = new Addition()) and hold all of them in an array you defined as a member variable.
- It will have a **calculateAndUpdateCountMessage()** method. The method will go over the array of operations, get the **numberOfTimesCalled** variables from each object and update the corresponding UI element. That is, it will call the **setText** button of the label showing the counts, which is at the bottom of Figure 1
- Implement the **countUpdated**() method of the CountInformer interface. This method will call the method in the above bullet point. As we mentioned in the previous section, the button classes will inform this interface object once a button is clicked. In that case, it will update the count values in the user interface using the **calculateAndUpdateCountMessage()** method.

# Final Remarks

Lastly, please ensure your software makes the following:

- Showing appropriate error message when a text field is left empty.
- When a unary operation is executed, it should automatically clear the **Number 2** text field.
- Operation count is incremented only if the operation is successfully executed. That is, all fields are not empty, and the result is calculated.
- Buttons corresponding to unary operations should be colored red.

You have now finished all of the necessary sections of the assignment. With this lab, we wanted to introduce you to some of the Object-Oriented Design concepts, including inheritance, interface, and abstract classes/methods. We also showed you the Open-closed rule, one of the 5 core principles, called S.O.L.I.D. principles. You can search the web to learn more about them if you are curious.

**IMPORTANT NOTES:**

1. Please comment your code according to the documentation and commenting conventions used in the textbook.