Tze Yu Choi

P459200

AT 1.7

Assessment GitHub link: https://github.com/LetThinkAboutIt/AT1.7

**Content**

[Data Structures 2](#_Toc10637528)

[Nametype and purpose of each variable (in a table format) 2](#_Toc10637529)

[Algorithms 3](#_Toc10637530)

[Pseudo code for each method 3](#_Toc10637531)

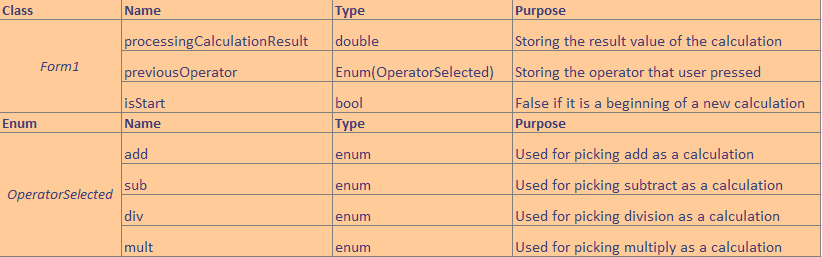
[Error handing techniques 5](#_Toc10637532)

[Recommended testing procedure (how should this software be tested before commercial release) 5](#_Toc10637533)

[Recommendations on upgrades and future enhancements 5](#_Toc10637534)

# Data Structures

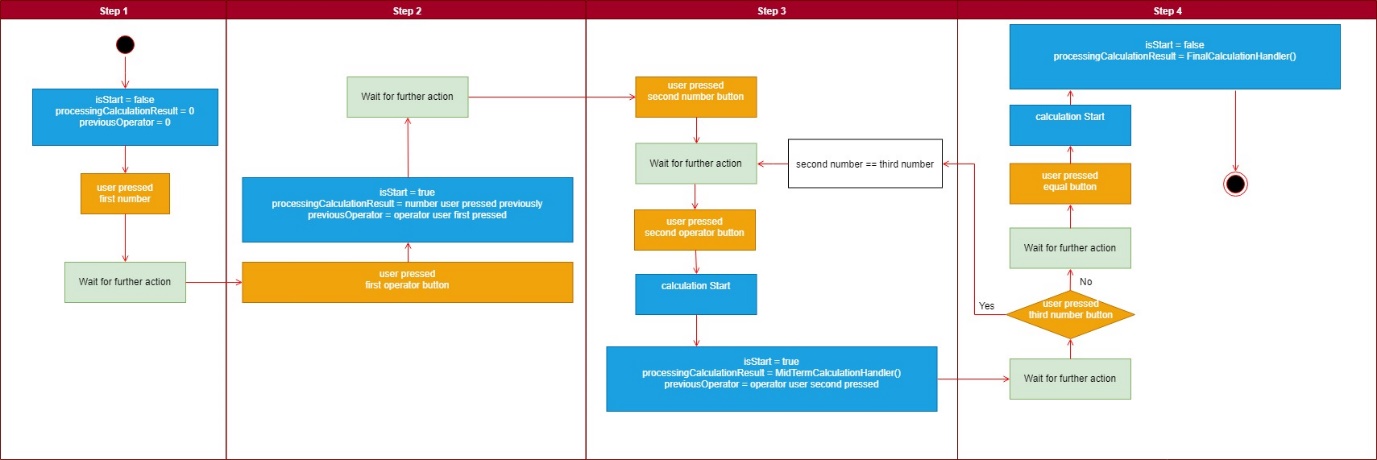
## Nametype and purpose of each variable (in a table format)



# Algorithms

## Pseudo code for each method

The main running flow of this application. ([Click here to have a big view](Action%20UML.jpg))



* Pseudo code for *MidTermCalculationHandler()*

MidTermCalculationHandler()

Is it new calculation?

If true

Get the number user input if no input then declares as 0

isStart = true

If false

Start Calculatie()

Clear output text box

Store the operator user last pressed

* Pseudo code for *FinalCalculationHandler()*

FinalCalculationHandler()

Is it new calculation?

If true

return

If false

Start Calculatie()

isStart = false

* Pseudo code for *Calculate(bool isEqualBtn)*

Calculate(bool isEqualBtn)

Get the number user input if no input then declare as 0

Case add

Call BasicMath.Arithmetic.Add(num1, num2)

If isEqualBtn is true

Output result to text box

Case sub

Call BasicMath.Arithmetic.Sub(num1, num2)

If isEqualBtn is true

Output result to text box

Case div

Call BasicMath.Arithmetic.Div(num1, num2)

If isEqualBtn is true

Output result to text box

Case mult

Call BasicMath.Arithmetic.Mult(num1, num2)

If isEqualBtn is true

Output result to text box

Default

return

## Error handing techniques

The rich text box contains a key press detector which allow the application check if the user is trying to input char that other than number, backspace or dot. Also, it will check if there is any dot already existed. If any of above is true, the application will deny entering the char and return.

Because the application has this protection so I can focus on handing the calculation problem like tan(90) is undefine, divided by 0, square root a negative number, etc.

For those problem I talk about above is handed by a if statement. Any user try to calculate the above function will get a pop-up message to inform them that is not able to calculate.

# Recommended testing procedure (how should this software be tested before commercial release)

The very first step of a program be tested is using a break point and Visual Studio built-in debug tool.

Programmer should try to input any button or input to see is the main variables in the right position where they need to be.

Then try to enter some input that expect not to be input. For example, a letter or double dots in a calculator and see does the exception handler working properly.

Finally, test every button and function of the application to make sure it is 99% bug free.

I want to say 100% but free at first but just haven’t see any 100% bug free application yet.

# Recommendations on upgrades and future enhancements

I recommend adding a TDD unit test for the application, so programmer doesn’t have to physically press all the button to test the application. In fact, it also stops programmer wasting time and it’s more efficient.

Also I may also recommend to make an inheritance. There will be one father class Calculator and we can have different calculator under this base class such as scientific calculator, programmer calculator, data calculator, etc.