CSC 413 Project Documentation

Fall 2018

Name: Inez Wibowo

Student ID: 905957954

Class, Semester: CSC 413.02, Fall 2018

A link to the repositories:

https://github.com/csc413-02-fa18/csc413-p1-inezwibs

Table of Contents

1 Introduction

1.1 Project Overview

This project is about testing our ability to create a well-designed program, which means the program is sustainable in the longer term, allowing for other programmers to take the program and modify a specific file containing the feature they want to add/edit/delete without having to re-do the entire code. The program produces a calculator that calculates simple math expressions like addition, subtraction, multiplication, powers, division, and parentheses. The mathematical expression can be as long or short as the user desires and it will display the result.

1.2 Technical Overview

This calculator project is an opportunity for us to practice object oriented designed program to create a calculator that evaluates a string of mathematical expression and presents an option to experience the calculator using Java GUI. The calculator contains simple mathematical operations: Addition, Subtraction, Powers, Multiplication, Division, and Parenthesis. Each of these operators are assigned priority to determine which operation will be processed first. Any numbers inside parentheses are evaluated first, afterwards powers are next, then follows multiplication and division, and addition and subtraction are last in priority. To use the calculator, one can enter an expression via the command line, the IDE console, or the GUI.

1.3 Summary of Work Completed

1.3.1 Created the following classes and methods:

1.3.1.1 In the "operators" package:

- Created AddOperator, SubtractOperator, DivideOperator, MultiplyOperator, PowerOperator that extends the Operator abstract class and each of the child Operator class extends the abstract Operator class and includes concrete implementation based on the operator predefined priority and the mathematical operation requirements

Operator	Priority	
AddOperator and SubtractOperator	1	
DivideOperator and MultiplyOperator	2	
PowerOperator	3	

- Additionally, created LParensOperator for opening parenthesis and RParensOperator for closing parenthesis classes to implement the priority levels of each.

Operator	Priority
LParensOperator	5

RParensOperator	4

- As instructed, I also declared an instance of Hashmap in the abstract Operator class. I used "static" instantiation so the Hashmap is ran when the class is loaded. The declaration of the Hashmap is placed outside the instantiation so that it lives after static function is completed.

- The boolean "check" function is used to check and return - if it is true - that an object of operators that contains the key value equals to that operator token that is passed to the check method.

1.3.1.2 In evaluator package:

- Created new Hashmap objects in the "Evaluator" method to support the 7 operators: addition, subtraction, multiplication, division, powers, left and right parenthesis.
- Added a space in the private variable "DELIMITERS" to tokenize white spaces separately. This additional whitespace will be ignored in the eval method.
- In the "eval" method, I added the following if/else loops to check the new operator:
    - If/else to check if operatorStack is empty, and if so to push new operator to the operator stack
    - If/else to check if the operator is the left parenthesis and to push the new operator on the operator stack if true
    - Otherwise, if the token is a close parenthesis, then I wanted the process to start running, which is to pop up the operator at the top of the stack, pop the last 2 of the operandStack, and push the result onto the operandStack. All of this will run until we see an open parenthesis next in the stack. Finally, once the left parens is found, the left parenthesis will be popped out
    - If token is not a close parenthesis, but the operator priority in the stack is higher than the new operator priority, which means it is either an open parenthesis or a non-parenthesis with a higher priority. If the former, then it should simply be pushed to the stack and continue down the if/else statement, and if the latter then it should be processed, which means the top of the operatorStack will be popped and executed against the 2 operands at the top of the stack. Result will be pushed onto the operand stack afterwards, and so will the new operator whose priority is being compared to the operatorStack priority at the beginning of the loop.
    - Otherwise the operator should simply be pushed onto the stack, in the case where the new operator is lower in priority than the operator on the top of the stack.
    - Once there are no more tokens to review in the string tokenizer, we have reached the end of the string, then I added a while loop, to complete the last few steps
        - If the operatorStack has a left parenthesis, then we can simply pop it off the operator Stack
        - If other than we can process the operator with the 2 operands at the top of the stack. Push the result onto the operandStack and repeat until the operatorStack is empty.
        - Once it is empty, then the return value, does not return 0 but returns the operand value that is last on the stack.

1.3.1.3 *Inside Operand class*

- I edited the “getValue ()” function to return the value, instead of return 0
- I also changed the check function to include a try-catch loop that will return an integer if the token passed through is indeed an Operand, and if not, this will throw a Number Format Exception.

1.3.1.4 *Inside EvaluatorUI class*

- I added code to the “actionPerformed” function to set the text in the text field as corresponding to the button that was pressed
- For the equals button, this indicates calling the “eval” method of the Evaluator class, so an object is created when button in index 14 is pressed, which is the equals button, to produce an integer result

2 Development Environment

Version of Java used: JDK 1.8

IDE used: IntelliJ IDEA 2018.2.2 (Ultimate Edition)

Windows 10 10.0

3 How to Build/Import and Run your Project

1. First, copy the remote repository from Git to the local repository, using “git clone [repository link] “ using https or using SSH keys (only after setting up your SSH keys following the instructions on this link. For ex., you can copy to your local directory by entering the following command on your local computer folder where you’d like files to be copied in:

   git clone [git ssh url]

2. The last folder in the path we choose to provide into Intellij to import from will automatically become the root. Since we want the “calculator” folder to be the root, after cloning to your local desktop, navigate to the folder by using file browser (or cd the file name) and then copy and note the path to the “calculator” folder
3. In the welcome screen for Intellij, select “Import project”
4. Paste folder path to “calculator”
5. Then tell IntelliJ, where the Java JDK file is. If you have not downloaded the Java file, you can do so now. Once downloaded, provide intelliJ with the path to the JDK directory
6. Once completed the files should appear in IntelliJ
7. Select the Build Tab, and click on “Build” to the build the Evaluator Driver file, or you can also click on “Run” and it will build and run for you

8. If running the Evaluator Driver, type in the expression in the IDE console that which you want to be processed. If running the EvaluatorUI, use the GUI to select the buttons on the calculator interface shown to display the mathematical expression to be processed, marked with a "=" equals sign.

4 Assumption Made

1) Assumed that the entered operations would only have infix integers expression
2) Assumed that the numbers inside the parenthesis had the highest priority, then the powers, then the multiplication & division (of equal priority), and the addition and subtraction of the last/lowest priority
3) All numbers entered and produced are integers
4) Assumed that we are only processing the math operations for which we have classes for
5) CE will clear the last digit of the expression and can be used repetitively until it clears the entire expression string while C will clear the whole string
6) Results can be continuously used in the following expression in the GUI, for ex if the result of an operation is 5, then 5 can be used as the beginning of the next mathematical expression, unless cleared. However, results in the console cannot be used continuously and one would have to start again at the prompt "Enter an Expression" once an operation is completed.

5 Implementation Discussion

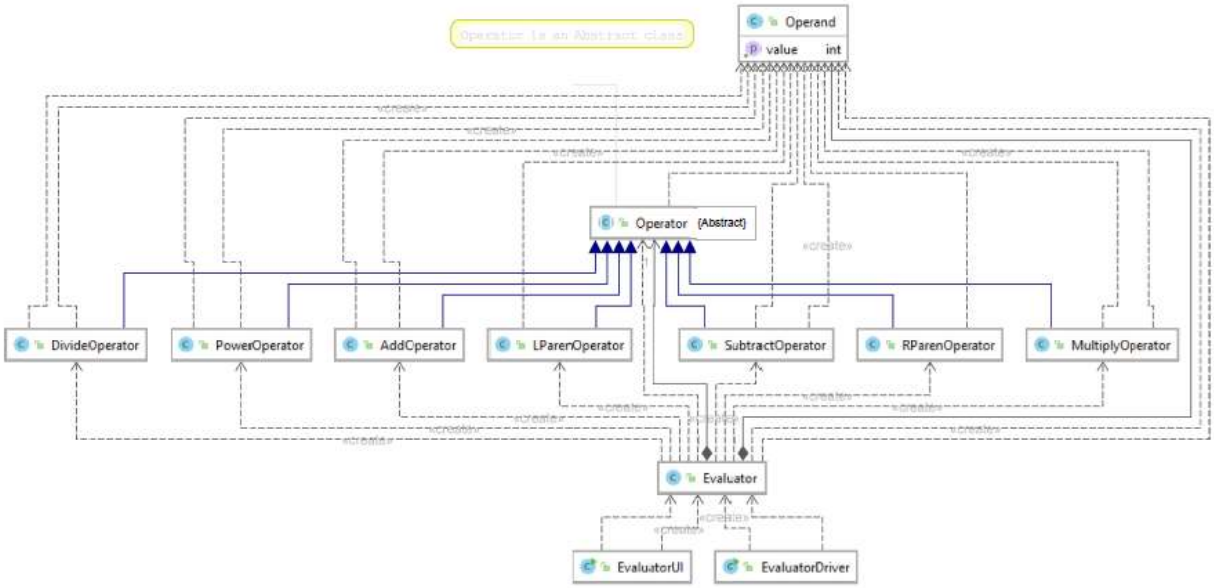
The implementation of the Operator classes is placed in separate classes, under the "operators" package so that they are independent, and another developer can add on additional classes as they like to add new mathematical operations in the future.

Implementation of the if/else loop in the "eval" function within the evaluator package, in "Evaluator" class, was done iteratively. First, I started with lower level mathematical expressions, such as addition and subtraction, and continued to multiplication & division, powers, parenthesis, and lastly nested parenthesis with alternating orders of priorities such as 2*3+(3-5) *2-((2^3*4) +5/2) to test the hierarchies of the operation as well as the effectiveness of the loops.

Using two stacks are effective because you can sustainably add more complex Operands and Operators this way.

The following diagram was generated through the UML Diagram tool through IntelliJ. It displays the hierarchy of the classes and the relationships between them. Additionally the Abstract class, such as "Operator" is also marked with a note "{Abstract}" in the diagram.

5.1 Class Diagram



6 Project Reflection

In the future, if I had more time, I would have liked to test the sustainability of the Calculator by adding on additional operations, and operands. I suppose this would then force me to review the "eval" function in the "Evaluator" class more closely, as currently there are some areas that are 'hard-coded' such as trying to identify where the parenthesis is. I could not get the .equals method to work to identify the parenthesis token that was passed, and tried to work with instanceof to no avail. So, I ended up short-cutting and changed one of the priorities in the "LParensOperator" classes to 5, which would have originally been 4, the same as the RParensOperator.

7 Project Conclusion/Results

Working with this project felt like peeling an onion. At least , that was my approach to the resolving the idea of what we wanted to accomplish and deciding how I could make this work in the time allotted. In the very beginning I simply mapped out what packages were imported where, which classes used which methods, what each method intended to accomplish, and I tried to understand the existence of the code that was already provided; the purposes they represented and so on. Then I started tackling the easiest first, such as the Operator classes, as suggested by the Professor. I started with the Addition, Subtraction, Multiplication, Division, and Powers. I didn't add on the parenthesis classes until later. I started with the Addition class, and tested only addition operators, then slowly I layered on more operations. Until finally, I added the parenthesis.

When I got stuck on a problem or issue, such as removing white spaces in the string, or looping until a left parenthesis is found, I try to take different approaches and trying to understand how I can leverage what is already provided. With removing white spaces, I realized, all I needed to do was add a white space in the DELIMITERS, which then will be ignored by the, not …. equals(" ") method , on the tokenizer at the top of the eval method. This realization happened several days after wrestling with the

issue. The left parenthesis issue I felt was shortcut-ed and I was concerned about keeping the structure of the program the way it is rather than modifying it. It was har to make sense of the ">=" comparison of what was on the operator stack and the new operator, when it came to the parenthesis, and I don't know if that is just a block in the way I've wrapped my head around it but I think I could have made it work differently if provided more time.

Additionally, the CE, option of the GUI, it would have liked it better if I could delete to the last operator, versus the last digit, however, again, it takes me a while to wrap my head around the idea, and how I need to code it, so I chose the easy way out which is to delete the last digit. At the least this will allow the user to have a reasonable user experience in which the user is not asked to do more work or exposed to frustration than need be.

I've learned a lot in this project about how to organize classes, how methods can work together to be sustainable. It might look more complex in the beginning, but it is actually structured for growth. I appreciated the amount of time and discussion on slack, and in the classes, to help us prioritize the classes and the assignment to tackle. In the end, I believe I produced a code that is acceptable for a first release, is reasonably bug-free considering the assumptions, and provides good user experience within the time limitations and parameters of the requirements.