Deepinder Bhuller

911770878

CSC 413.03, Spring 2019

https://github.com/csc415-03-spring2019/csc413-p1-rogue-7

Project 1:
Expression Evaluator and Calculator GUI

Introduction

Project Overview

Project 1 consists of many classes that come together and work as a basic calculator. The project was given to us incomplete and the goal is to develop the working logic of a calculator. The goal of the program was to evaluate expressions such as 2 + 3 * 4 should = 14. Expressions consisted of digits such as “3” and “12”, using operators +, -, *, /, and (,). Each operator had a priority to determine the order it was to be executed. The method that achieved this was the eval() method. Our job was to complete the implementation of the algorithm. If an Operand token is scanned, an instance of that object is pushed to a stack. The same follows for an instance of an operator, ie: a digit. Inside the stack, the operators are organized by priority, the + and - Operators having the highest priority, followed by the * and /. The operator with the least priority was the ^ operator. Our requirements were to implement the Evaluator class, implement the Operator class hierarchy, test the Evaluator implementation using the unit tests provided, use the Evaluator implementation in the calculator GUI.

Technical Overview

Part 1 of this project was the develop and implement an abstract class Operator. I realized very early that for this project, a good approach to fixing the code would be to implement subclasses of the class Operator. The classes would be named after the mathematical operation performed by the operator, for example the “+” operator class would be called AddOperator. Other subclasses implemented are AddOperator, MultiplyOperator, DivideOperator, Subtract Operator. Another requirement of the project was to use an operatorStack and a operandStack. These two stacks would keep track of the operators and operands in the String expression, which is passed into the eval() method. The passed in string is then parsed by a String Tokenizer. If the token is a “(“, it is pushed to the operand stack with priority 0.

Operands such as “2” or “23” are pushed to the operand stack. If the priority of a new Operator, meaning if the next token parsed is an Operator, and its priority is less than or equal to the priority of the top of the operand stack, the top element in the stack is popped out, and the top two operands are popped out of the Operand stack and the execute() method is called on the two operands with the corresponding operand. Calling this method executes a method implemented in each sub cass of each operator which performs the appropriate action on each operand with the proper operator. This method returns an operand object and is pushed to the stack. If a “)” operator is reached by the tokenizer, a method reachedClosePar() is called which stores the top element of the operator stack in a temporary variable temp. Then the top two operands of the operand stack are popped. Then the temporary operator’s execute method is called with the two

operands popped as parameters then the returned operand is pushed to the operand stack. The main logic of the program is written with if statements checking if a token is a valid operator. If an invalid token is passed in, the program throws a RuntimeException with the message “*******invalid token******”. The majority of the algorithm uses a while loop to check if a token of the expression passed into the algorithm is a valid operator or operand and if it is push it to the appropriate stack. If the token is an operator, the priority is determined upon the implementation of the subclass of the Operator class. Finally, the last operand is popped from the stack after all the whole expression is tokenized and processed in the eval() method. This value is returned by the eval() method.

The second part of this project was to implement a GUI linked to the eval() method when the “=” button is pushed. The GUI is implemented using an array of strings of each button type. The String objects then create an array of Button type objects the same length as the string array of buttons. The initial layout of the the JFrame object is set using the NORTH and CENTER positions of the object. A grid layout is implemented in the CENTER of the object. Buttons from the button array are added to the panel using a for loop and then the basics of the JFrame are set. Next, the actionPerformed() method, with the argument ActionEvent arg0. Inside this function, the action performed by the argument passed into the method is dependant on the button pressed, corresponding to the string in the original array. Certain events corresponding to the elements of the array trigger and activate certain things happening in the text field of the object. For Example, if the CE button is entered, the whole expression is changed and therefore not evaluated.

Each of the classes created in the operators package are implemented with the two abstract methods of the Operator class. These classes aren’t required to be abstract. The way to test each class is to use the Unit Tests for each operator that can be evaluated. For example, the AddOperator is tested using the AddOperatorTest and the Evaluator class is tested using the EvaluatorTest test. The unit tests are written by testing the evaluator with a string from something simple to something complex.

Work Completed

Overall, the work I did for this project could be broken up into three stages. First, I started with the implementation for the subclasses of the Operator class. When I first looked at the code I realized that classes would be need to be implemented for each operator that had priority and used the execute() method. Each sub class had an overridden method that set the priority of the operator and implemented that specific operators execute method. Each operator was initialized in a hashMap that stored the key, a string literal of the operator and the value, a new instance of the operator

subclass. This part was straightforward because each subclass only needed to implement two abstract methods of the superclass.

The second part of this project involved completing the eval() method and having it tokenize and evaluate a string expression passed into it. This part involved completing the partially implemented algorithm that would evaluate an expression. First I looked at the algorithm online and wrote pseudo code to get a basic understanding of what the method did as a whole. I gathered that if the passed in string could be tokenized, each token would be checked weather it was a valid token or not. If it was a valid token it is to be pushed to either the operand or operator stack. If the token is a operand it is pushed to the operand stack, if the token is an operator, depending on the priority it is placed in order. The lower the priority, the operation is executed first. If the operand stack is empty, a new instance of an operator is pushed to the stack. Another part of the algorithm was processing what happens between a closed and open parentheses. If an Open Parenthesis is tokenized, and the operator stack is full, compare it to the priority of the next object in the stack and push accordingly. If a closed parenthesis is detected, then a method, named reachedCloseParenthesis is called and the sub expression inside the parentheses is executed. After the resulting Operand object is pushed to the operand Stack. This method achieves this though storing the top of the Operator stack in a temporary variable, then popping the next two operands and calling the execute method on them. At the end of the method, the operator is popped out of the stack and the parenthesis expression is evaluated. Within this class I added to the DELIMITERS string by adding a “)”, “(“, “ “, and “$” operator. The space and parentheses operators were all parts of expressions and evaluating expressions, but the “$” operator was used as a BeginningExpressionOperator. This operator kept track of the end of the stack. This is important to implement because so the program can know when it is done parsing the string. If an instance of the operator is already in the stack when it is initialized, then the end of the stack can always be kept track of.

The third part of this project was figuring out how to connect the GUI to the logic of the program. At first this step seemed very difficult and I didnt know where to start but after googling the JFrame and ActionListner classes I was able to gain some insight to how the classes worked. I looked over the source code and the class names made it easy to look up and understand what each object, method and class did. I learned from the code that the text field of the GUI is defined by the TextField object and the button panel is defined by a Panel Object. An array is created of the same length as the string array of “buttons”. Each button object that is created from this string array was added to the button panel and then the method actionPerformed() was implemented. This method is the one that watches for input from the user based on mouse clicks or button clicks. To implement this method I used if statements to check if the button pressed was the ActionEvent variable passed into the method. If the argument is the button at the

index of the button array then the text field gets a corresponding call to set the text of the textfield to the string value of the String button array. An if statement was added for each index of the buttons array. At the 14th index of the Buttons array instead of setting the buttons textField to point to the String value of the bText array, we set the condition to execute the creation of a new Evaluator object and set the text field to the toString() value of the calc.eval method.This sets the 14th index of an array to call the eval() method and compute the result of the expression entered by the user.

Development Environment

Version of Java
Version 8.0
IDE
IntelliJ Ultimate 2018.3.4

Build & Import

To build and import my project in the IDE I used, first you must clone the repo so you have a copy of all the files in my project in your environment. After cloning the repo, you can open your IDE and click the “import from existing sources” button. After this step, you should select the appropriate folders containing the JUnit tests and the source folders from the selection window. It is suggested to make the source folder for this project to be the calculator folder for the project. After this the JUnit tests can be run and tested for each individual Operator subclass. They can be run by going to the test folder and left clicking and selecting “Run”. To access the the GUI, you run the EvaluatorUI class which builds and executes the GUI.

How To Run

To run this project, you need to download any IDE or intelliJ and after downloading the IDE’s and ensuring your program and environment are compatible, then you would go into IntelliJ and open up the project. After opening you would go to the EvaluatorUI class which would create and display the Calculator UI on the screen. To run the tests in the test file folder, you go go to edit configurations, set the configuration to run the test you want and then click the “run” button in the toolbar of the GUI. In the command line the program would be run with the javac command and then run with java + filename command.

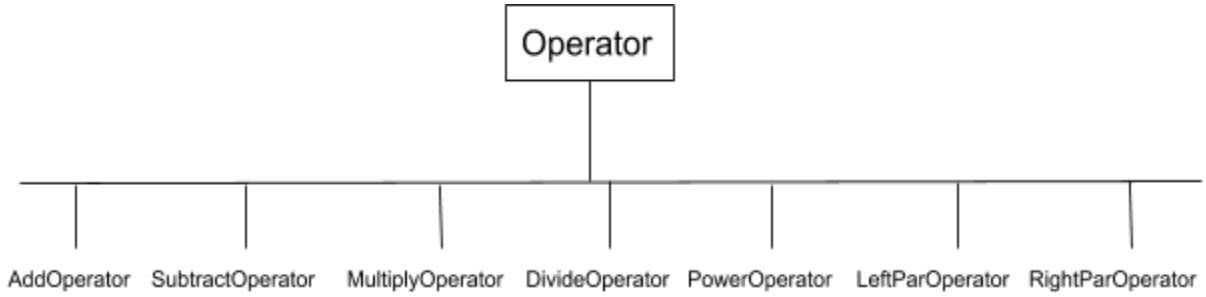
Assumptions Made

Assumptions made in this project include assuming that the calculator will be only solving balanced infix expressions meaning there are no extra parenthesis or operators with no operands. Another assumption to be made is that the program won't have to account for negative values because there is only one - sign which is the subtraction operator. We can assume most operators are binary operators, not unary. It was also assumed that there was no need to worry about values as ints or doubles because overall it would not affect the logic of the program.

Implementation Discussion

Overall the implementation of this project was very straight forward. To complete the project we needed to implement one class and define the subclasses of another class. The project is structured in a way where the operator and evaluator classes are in two different packages. The operator classes go in the /operators package and the evaluator class goes in the /evaluator package. The AddOperator, SubtractOperator, MultiplyOperator, PowerOperator and DivideOperator are the subclasses of the Operator class. Each subclass implements the abstract methods of the superclass. These methods are priority() and execute(Op1, Op2). The priority method I defined by setting the value of an int variable to the literal value of the priority(ie: 1, 2, 3, or 0). All these classes were essentially the same except for the difference in priority and how the execute() method functioned. The only major difference was with the PowerOperator because I defined a function that would take the power of the operand to the value specified by the other operand. These classes were initialized in a HashMap as the value to a String token of the operator(ie: AddOperator → “+”). The HashMap is initialized in the Operator class and stores the operators and the open and closed parenthesis. It also stores a (key, value) pair of an operator “$” to keep track of the end of the stack.

This is the hierarchy for the Operator class:



I tried to write readable and cohesive code. I used naming conventions to try and keep the classes/variables organized. I didn’t add much in terms of documentation above my

code other than "@Override" and some comments to help me keep track of what certain parts of the program are doing. Overall I tried to keep my code organized, readable and maintainable.

The evaluator package consists of the Evaluator, EvaluatorUI, EvaluatorDriver and Operand classes. The Operand class defines the behavior and attributes of the operand objects that make up the expression.

Project Reflection

Overall I found this project to be a good balance of challenge and review. The majority of this project involved working with multiple classes to evaluate a text expression like a calculator does. I spend the majority of my time in this project working on the eval() method. The algorithm was difficult for me to implement at first and I didn't understand where to begin. I went to a white board and started to write out how the program was working. I drew two stacks and "evaluated" an expression on the board. By doing this I was able to really understand how the eval() method works. The algorithm checks if a token is an operator or operand while there is another token to tokenize. If the token is an operand, it is pushed to the Operand stack. If the token is an operator, and if it is valid. When the operator stack is empty, the new operator derived from the token is pushed to the stack. When a "(" reached, push it to the stack. When the ")" is reached, then "process" the expression by calling the reachedClosedPar() method. I messed around with the code for a while because I kept getting empty stack and invalid token errors. Someone on the slack channel for the class mentioned adding things to the DELIMITERS string and that fixed some of the errors. I added the parenthesis into the String then added corresponding lines of code and I now passed some of the tests! I was very excited and motivated to push further and get the other tests to work. After this I went to office hours and got help with where I was stuck. My algorithm was not popping out the closing parenthesis after reaching the end of the expression. By popping the operator stack one more time at the end of the method, this ensures that there isn't something left over in the stack and the expression evaluates evenly.

The easiest parts of this project were to implement the hierarchy for the Operator class and finishing the implementation of the Operand class. This part of the project was straightforward and was almost like a review. I feel very comfortable defining subclasses and implementing abstract methods.

The GUI was another challenging portion of the project but that was because I had no prior experience with GUI's or visual elements in java. At first the EvaluatorUI class was completely foreign to me and I didnt know if I would be able to complete that part of the project. After looking at the code for a while and doing some reading/research on building a GUI, I quickly figured it was just a matter of mapping the buttons in the bText array (of strings) to the buttons array of type Button and putting them in the Panel object. I used if statements to check if the event(mouse click) is on

this button, then the corresponding operator is added to the text field. When the = button is clicked, the eval method is called and the expression entered in the calculator is evaluated using the other classes. I liked programming the gui, although it felt kind of tedious. If I had more time I would explore more into modifying the look and appearance of the GUI. I have experience with HTML, CSS, and JavaScript so i'm curious to learn more about how UIs and graphics are handled in Java.

If I could do anything differently for this project I would have spent more time working with and learning how to use the GUI. Aside from that I am confident that i've met the requirements for this project in terms of fixing the other classes and getting the logic of the program to work. I have implemented the subclasses of the Operator class and worked on the logic to where I passed all unit tests provided. This tells me that my Operator class and my Evaluator class are working properly.

Overall I am proud of the work i've completed for this project and Im confident my design is well done and correctly implemented. I know I could have written and organized my project better, and there are some edge test cases that I could have tested for. I think this is a good project to get reacquainted with java and start working on more complex projects.

Conclusion & Results

My program passed all unit tests and the buttons of the GUI calculator are functioning properly. I correctly implemented the classes and the algorithm. I did not test edge cases such as two parentheses next to each other like so, “)(“. This case would evaluate to multiply but this functionality wasn't added to the program. I used the supplied GUI and filled out the buttons and called the eval() method upon clicking the button with the mouse. I enjoyed this project and plan to update it and keep working on it to improve my skills.