Keawa Rozet

Github: <https://github.com/CSC-413-SFSU-02/csc413_02_p1-krozet>

9/17/17

Csc413-p1

Introduction

This is a functioning calculator using two separate stacks to complete each equation. Through the EvaluatorUI, the user inputs an equation using the buttons on the calculator and an answer is outputted in return when the “=” button is pressed. While using the EvaluatorTester, the user inputs many different equations into the main function and an answer for each equation is outputted.

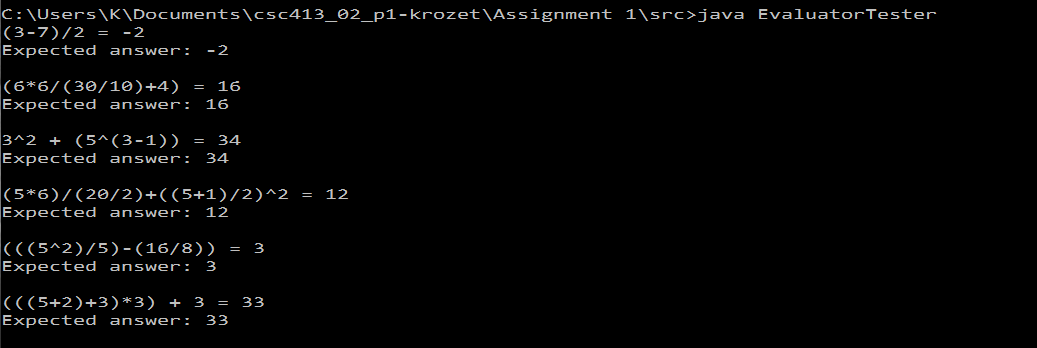
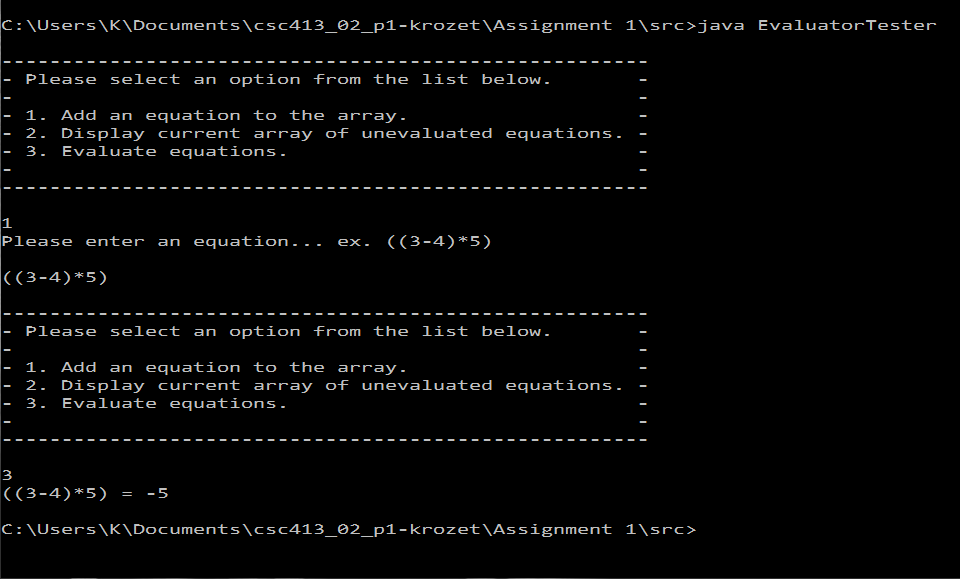
The calculator works by taking in a string and passing it to an Evaluator object. The eval method in the Evaluator object works by tokenizing the string using operators as delimiters. Each token is then evaluated to be either an operand or an operator. If it is an operand, it is passed into the operandStack. If it is an operator, it goes through a series of checks to see how it should be handled. If the operator is a “(“, then the evalParenthesis method is called. This method operates very similarly to the eval method by using two separate temporary stacks to hold the operands and operators. Since the tokenizer is a global variable, the evalParenthesis uses the same tokenizer as eval and picks up right where eval left off. As soon as the “)” operator appears, the values in the temporary stacks are evaluated and a final integer is pushed to the original operandStack. The evalParenthesis method has the ability to make a recursive call in order to handle imbedded parenthesis.

Compilation

The EvaluatorTester class has two different methods to test the Evaluator class. The method that is initially setup for use, called userInterface, uses a UI at runtime to plug in equations. This method gives the user the options to either add an equation to an ArrayList, to display the contents of the ArrayList, or to evaluate the equations stored in the ArrayList.

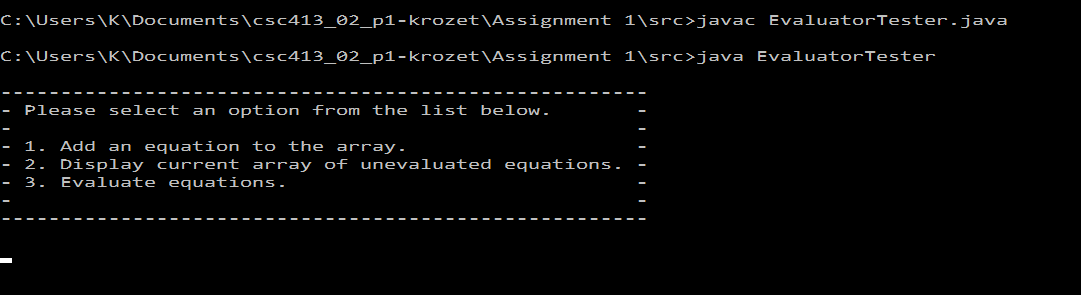
The other method is the manualInput method. This involves going into the method starting on line 53 and inputting the equations before compiling the program. String[] args is used to store the equations and String[] answers is an optional component to display the expected answer for easier comparison.

Like stated above, userInterface has been initially selected for testing purposes. If you wish to select the manualInput method, simply comment out line 15 and uncomment line 14, then alter the values starting on line 53.

The left is an example execution of the userInterface method and the right is an example execution of the manualInput method.

Command Line Instructions

Normal java command line compilation and execution will work on this project. Go to the directory, compile using javac and execute using java. This applies to both EvaluatorTester and EvaluatorUI.

However, adding args to the main function call using “src>java EvaluatorTester arg1 arg2 …” will not produce anything. The args string in the main function isn’t utilized as it previous was when first assigned.

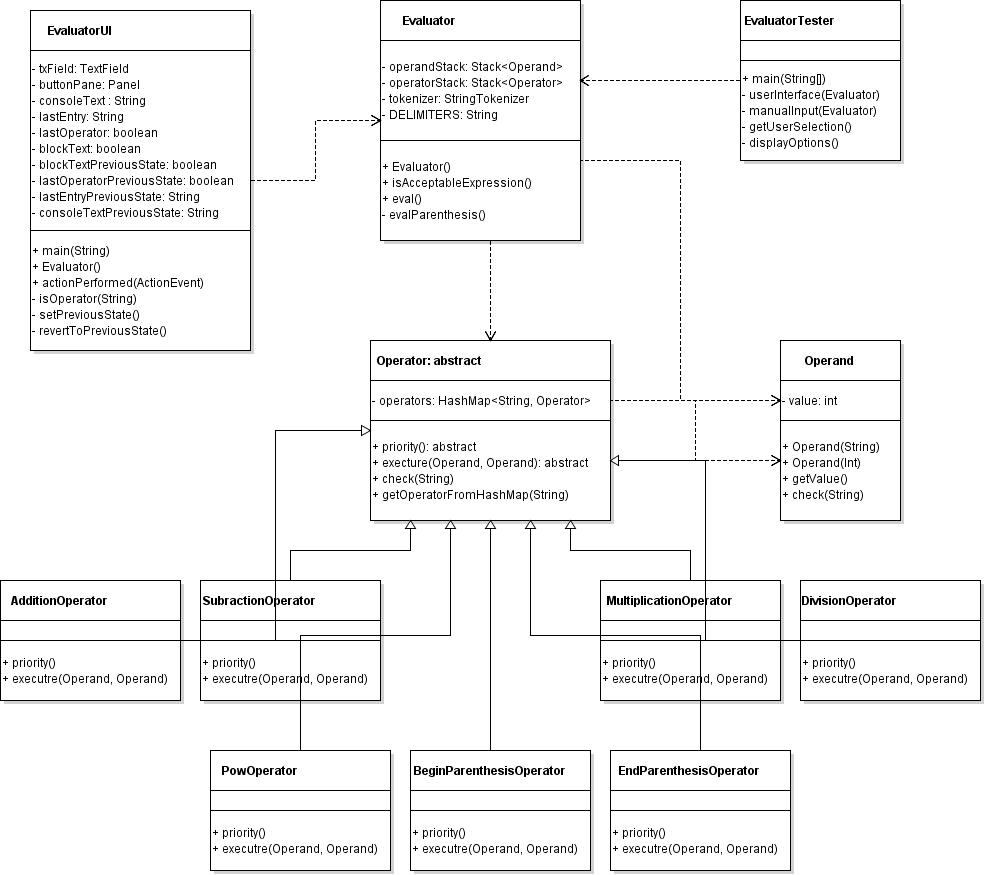
Assumptions

An assumption held was that most of the base code given was to act as a frame work for how the program would work. If not for this, I might have considered a different approach to implementing the calculator. I also held the assumption that accessing a HashMap within an abstract class would only be possible if there were an instance of that abstract class. This obviously is not the case if you make the HashMap static.

Implementation

The biggest decision I was forced to make was how I would handle the parenthesis. There were a few different options, but I thought that the parenthesis should each be handled as a mini equation. Thus, the evalParenthesis method largely mimicks the eval method. As far as time complexity goes, its worst case is equal to the eval method, which means it does not negatively contribute to the overall speed of the program.

I opted out of using the “#” and “!” operators because of how my eval method works. It compares incoming operators to the last operators stored and evaluates if needed, and then at the end it compares the remaining operators to finish the evaluation. If my algorithm pushed all the operators and operands, and then evaluated them at the end, then I would have been able to find use for the “#” and “!” operators to speed up my program.

The code isn’t as clean as I would have liked it to be, and that is due to the fact that I tried to implement a lot of safety nets and edge cases. These edge cases are not finished, though, because I found out ¾ of the way through that we did not have to worry about implementing any of these. I will go back through later and add the rest of them in, so I didn’t remove them from the code – thus making it messy at parts.

Conclusion

This project was my first time working with a GUI through java and it was simple enough for me to gain a decent understanding on how it works. In the near future I would like to beef up this calculator by add floats, possibly trig functions, a nicer GUI, and edge cases so that my calculator won’t break.

My biggest challenge was working with the GUI. Since it was something entirely brand new, I had to do a lot of outside research in order to overcome the obstacle of not knowing how to approach this aspect of the program.