San Francisco State University
CSC 413-03
Spring 2018

Victor Muñoz

https://github.com/csc413-03-sp18/csc413-p1-Vmunoz94

Introduction:

We were given an almost complete version of the Evaluator class (Evaluator.java), and our task was to complete the code in Evaluator.java, as well as the classes it uses such as Operand, Operator, and the GUI. Netbeans IDE was used to make all the necessary changes in the java files that we were given. The first thing that had to be fixed were the two simple classes; operand and operator, which handled operations such as addition and subtraction.

I first started with implementing the operator class, the very first thing we were told to do was to create a hash map with the keys being the operator expressions and the values being new subclasses that extend the abstract superclass. Since the parent class is abstract, all the subclasses that extend from the abstract parent class must override all abstract methods. The two abstract methods being the priority of the operator and executing the operation itself. Once the operator class was finished, I looked into the operand class, which simply returned the value and checked whether or not it is a valid expression or not.

Next, I worked on the Evaluator class and used the given test cases in order to check my program was running correctly. I changed the delimiters in order to accept parenthesis as valid tokens. When an open parenthesis was found, I would create a new operator and push it to the operator stack. When a closed parenthesis was found, then I started processing operators until an open parenthesis was found. Lastly, pop the open parenthesis. If no parenthesis is found, then process normal operations. Keep doing this until there is nothing more to operate.

Processing operation means:

* Pop the operator Stack
* Execute the Operator with the two Operands
* Push the result onto the operand Stack

This process happens a lot, so what I did was create a method that is called whenever in the program I need to process this operation. This removes redundancy.

Lastly, the GUI had to be worked on so that all the buttons did something and worked. All the buttons already had a value, I just made each button display at the very top. When the equal sign was pressed, the display at the top was turned to a string. That string was then passed to the evaluator and displayed the answer at the top. The "C" button removes the whole string, and the "CE" button only removes the last character in the display. The answer remains on the display unless the "C" button is pressed.

Instructions:

All compilation and execution were done on Netbeans.

Using the terminal, I only know how to compile and run the working GUI version. First you must make sure java is installed onto your terminal. Which is done by typing:

```
sudo apt-get install openjdk-7-jdk
```

onto your terminal. In order to compile and run the GUI, you must first find the EvaluatorUI.java file. To compile you type:

```
javac EvaluatorUI.java -Xlint
```

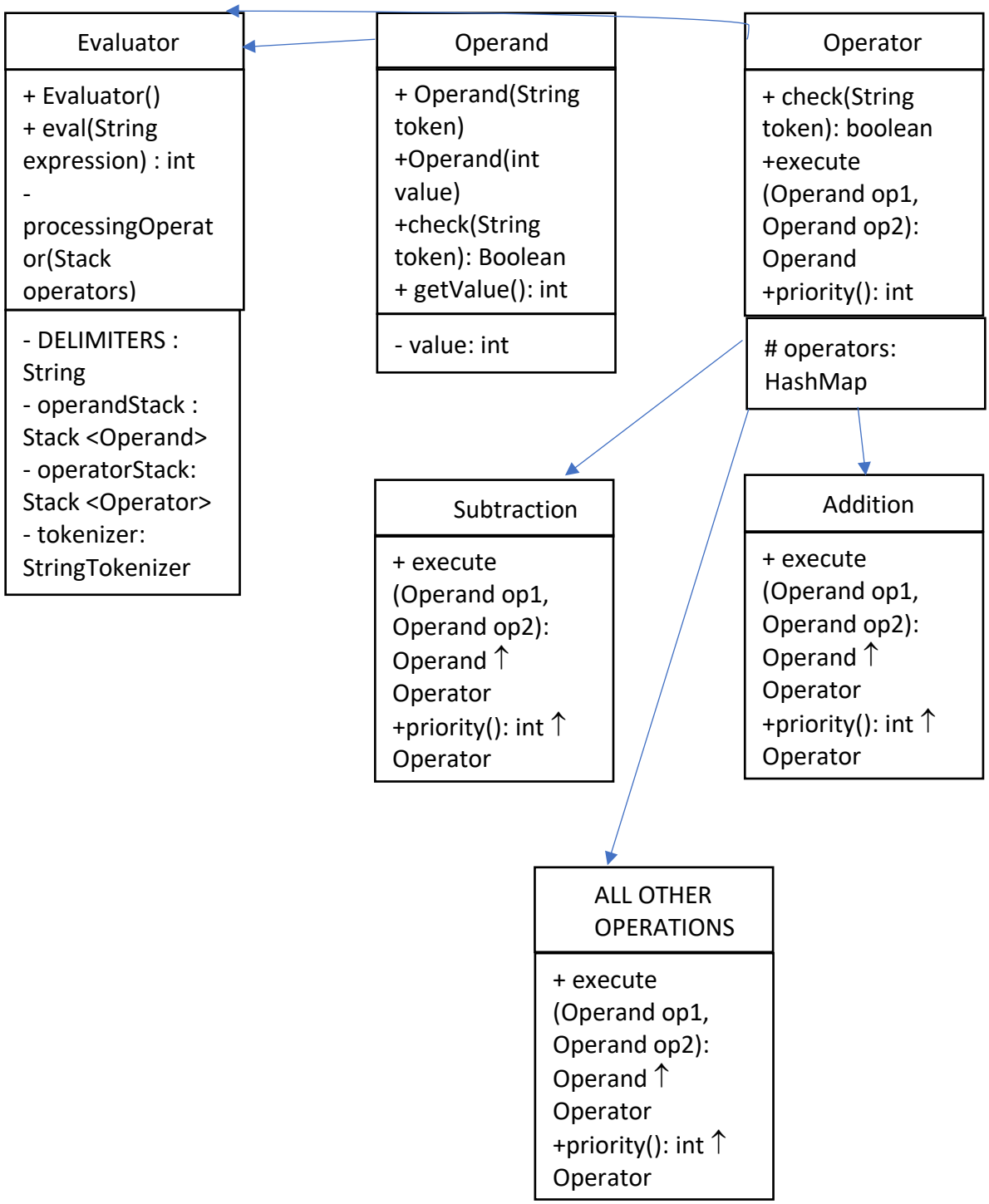
and to execute, you type:

```
java EvaluatorUI
```

Assumptions:

- Only integers will be used
- 2(2) is not a valid expression; however, 2*2 is a valid expression
- Division will not return floating numbers
- Only valid expressions will be used
- No division by 0

Implementation:



Result:

I had two major issues when completing this project. The first issue I encountered was that the tokenizer was not accepting parenthesis as valid tokens. For the first test case, the tokenizer would take ((1 as one single token, which is invalid. In order to accept parenthesis as valid tokens, I added them to the DELIMETERS. This solved the issue and now I was able to properly handle parenthesis. For example, when I see an open parenthesis I should create a new operator object and push it into the operator stack. On the other hand, when I see a closed parenthesis I should start processing the operators until I reach an open parenthesis, and lastly pop that open parenthesis out of the operator stack. This was my second issue. I completely forgot to pop the open parenthesis out of the operator stack. And this went unnoticed for a long time because my code would still pass all the tests except the hard test. It would give me an answer of 1220 for the hard test when it should be 1176. I couldn't find the problem until the next day when I looked at the code with a clear mind. Once fixed, the code passed the hard case.