/\*\*

\*

\* File: CMSC 350 project 2

\* Author: Bedemariam Degef

\* Date: November 15, 2019

\* Purpose: This program is to use a binary tree data \* structure in order to convert postfix expression into \* infix expression

\*

\*/

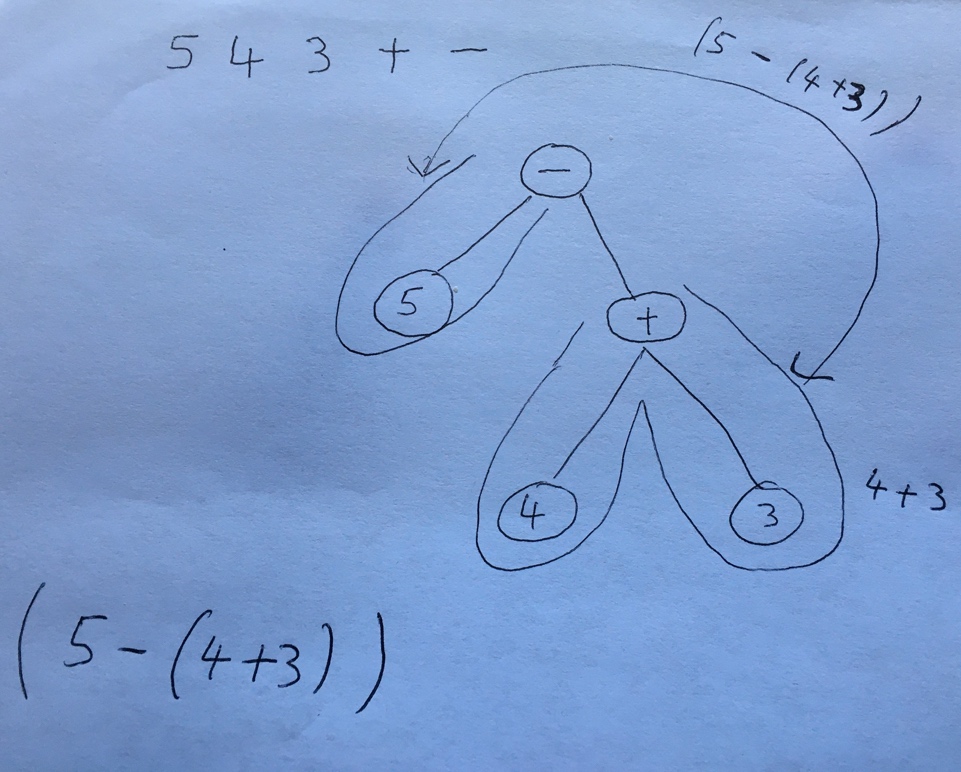
1. **Assumptions, main design decisions, error handling;**

My assumption after reading the instruction was that I needed to figure out a way to create a binary tree from a stack. Writing to the file portion might be hard to implement. Reading the assigned topics and other online resources have helped me understand what I needed to do.

There are going to be five classes and one interface. The first class is P2GUI which is going to design and implement the GUI. Exceptions are caught and handled in this class. The second class is OperandNode which is going to serve as an operator node. The constructor is going to initialize a single value which is going to be the operand value. The third class is going to be OperatorNode which is going to be the operator node. The constructor in this class will initialize operator node and the left and right children which will be an operands. The fourth class is PostfixEval which is going to take in the postfix expression, and by using the OperatorNode and OperandNode classes to construct a binary tree. There is also going to be a node interface to help use methods across multiple classes. The last class is a user designed InvalidInputException.

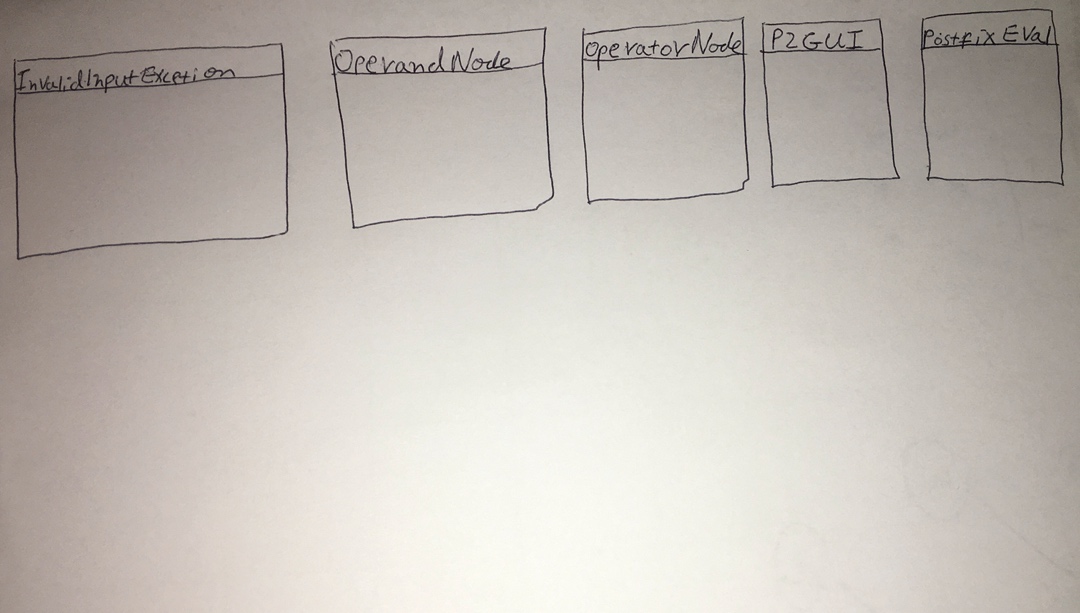
**The logic of the program is as follows**

**After the user enters the postfix expression and press the construct tree button, trim excess multiple white spaces from the user input and split the input into tokens after every single white space value. If it is a number create a new operand node and push it on stack, or if it’s \* - + / create operator node and add the previous two nodes as children and push to the stack. Write the address to the file. Finally, return the infix expression by traversing through the binary tree.**



There are going to be four exceptions used in this project. InvalidInputException is thrown and handled for any illegal inputs. If NullPointerException is thrown for an empty input value, it is going to be handled. Similarly, EmptyStackException is also handled when it is thrown for invalid postfix expression. IOException is also handled after it is thrown for an error when writing to the file.

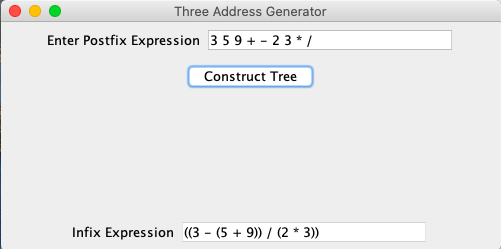
1. **A UML class diagram that includes all classes you wrote. Do not include predefined classes. You need only include the class name for each individual class, not the variables or methods;**

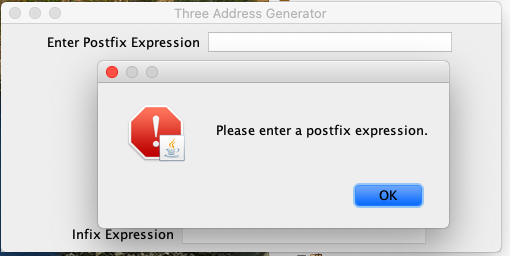
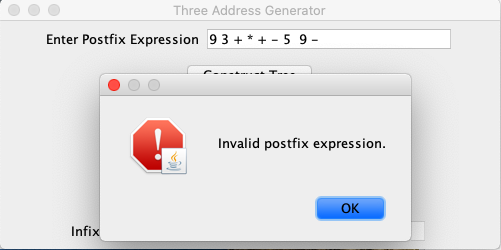
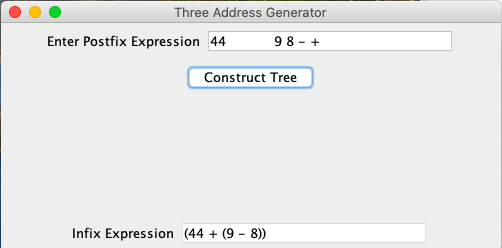
****

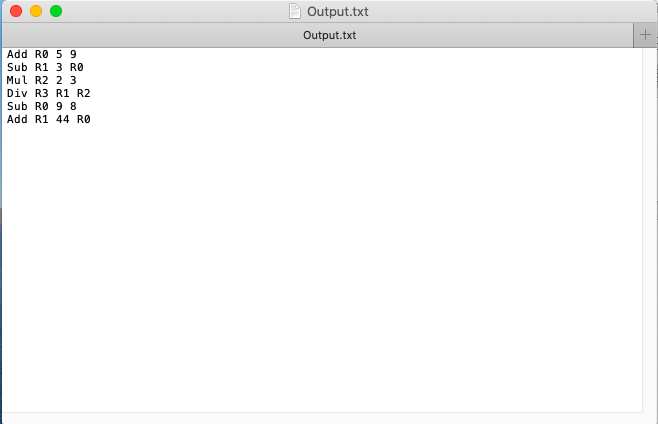
1. **A table of test cases including the test cases that you have created to test the program. The table should have 5 columns indicating (i) what aspect is tested, (ii) the input values, (iii) the expected output, (iv) the actual output and (v) if the test case passed or failed. Each test case will be defined in a table row.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **What aspect is tested** | **The input values** | **The expected output** | **The actual output** | **Pass or fail** |
| Testing all operators | 3 5 9 + - 2 3 \* / | ((3 - (5 + 9)) / (2 \* 3)) | ((3 - (5 + 9)) / (2 \* 3)) | Passed |
| Testing with spaces | 44 9 8 - + | (44 + (9 - 8)) | (44 + (9 - 8)) | Passed |
| Testing with invalid operators | 9 3 + \* + - 5 9 - | Invalid postfix expression | Invalid postfix expression | Passed |
| Testing with empty input |  | Enter postfix expression | Enter postfix expression | Passed |
| Invalid token beginning with a digit | 5 5 $ + - | Illegal token $ | Illegal token $ | Passed |
| Testing writing on a file | The above valid inputs | Add R0 5 9  Sub R1 3 R0  Mul R2 2 3  Div R3 R1 R2  Sub R0 9 8  Add R1 44 R0 | Add R0 5 9  Sub R1 3 R0  Mul R2 2 3  Div R3 R1 R2  Sub R0 9 8  Add R1 44 R0 | Passed |

1. **Relevant screenshots of program execution;**

****

****

****

**(5) Lessons learned from the project;**

I have learned about binary tree data structure and how they are implemented using a stack. Different ways of binary tree transversals. For example, Preoder, Inorder and Postoder. It got easier to understand once picturing what is going on in memory. Visualizing it and putting on paper and seeing how it works helped me learn it better and faster.