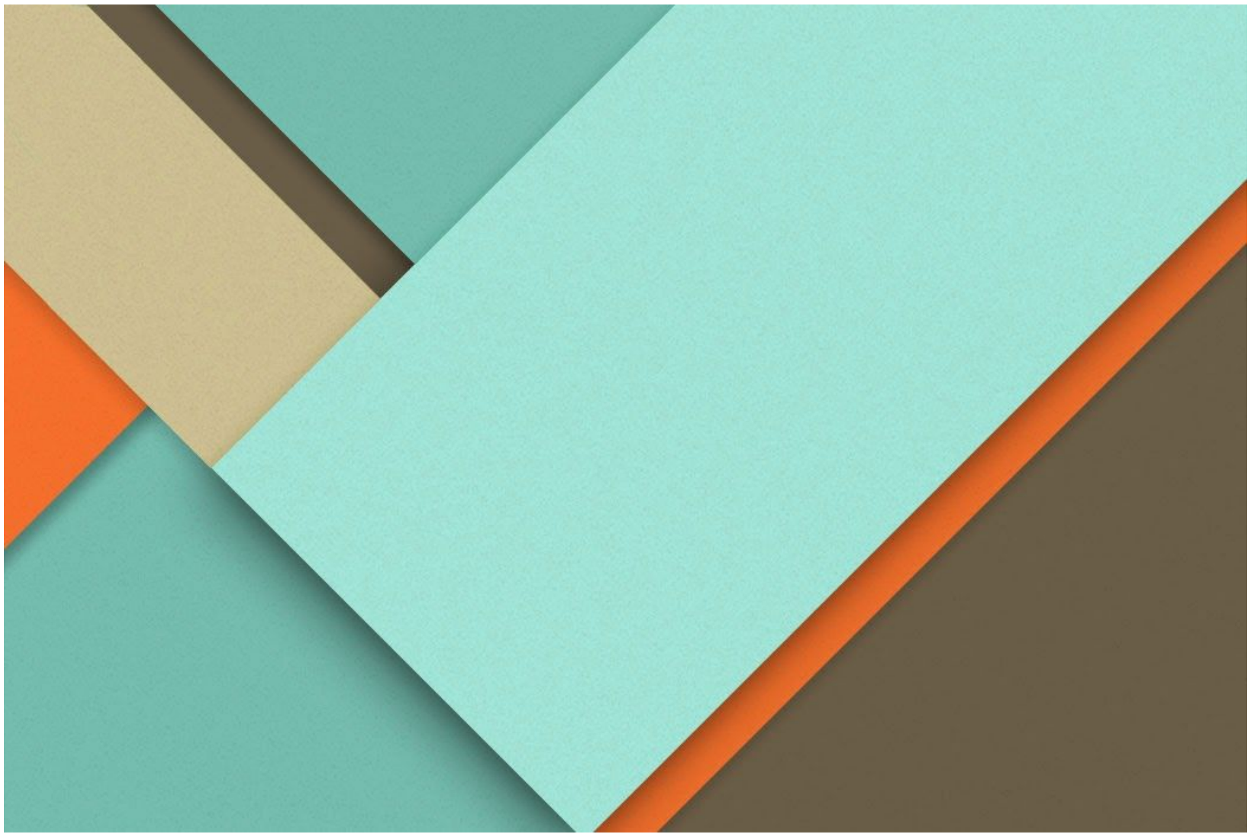


Team Project 2

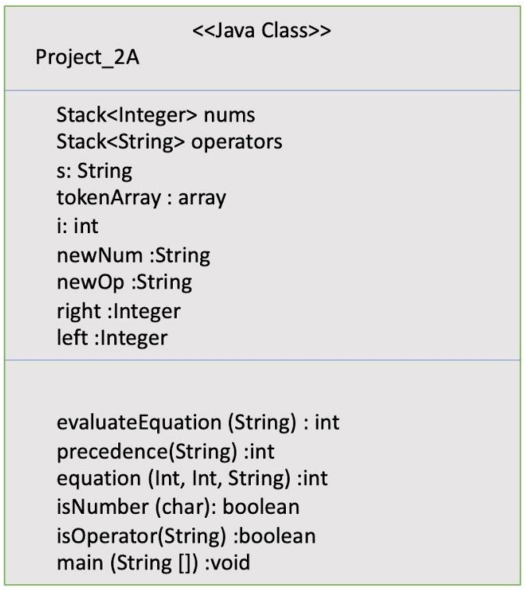
02.16.2021

Dalton Vining
Samuel Maynard
Shane Callaway

How we designed the system:

The system was designed in a way that combines the methods to convert an infix expression to a postfix expression, and the method to solve a postfix expression. The main change that was made is that instead of adding the expression to a string as you process the infix expression numbers are added to a stack and operators are added to a stack so that as precedence and parentheses based requirements are met calculations can be performed with the proper numbers. Having the system function in this way removes the need to first convert the infix expression to a postfix expression and then solve it, which means this method should be much more efficient, as each item is only added and popped from a stack at most once, and only one string must be gone through, whereas performing each method individually would require you to process both the infix and postfix strings.

UML Class diagram:


<<Java Class>>
Project_2A
Stack<Integer> nums
Stack<String> operators
s: String
tokenArray : array
i: int
newNum :String
newOp :String
right :Integer
left :Integer
evaluateEquation (String) : int
precedence(String) :int
equation (Int, Int, String) :int
isNumber (char): boolean
isOperator(String) :boolean
main (String []) :void

Two test cases:

The .txt file associated with this project contains several expressions to evaluate. However I will use just two of those expressions here to more precisely show the input and output of our project.

Test case 1 will be "1 + 3 > 2"

Expected result: 1 (boolean true)

Actual result: 1 (boolean true)

Test case 2 will be "(1+2)*3"

Expected result: 9 (int)

Actual result: 9 (int)

These expressions are first read in main with each expression separated by a ";"char. Spaces within the expression are ignored. So, for example, with test case 1, the spaces between each operand and operator are ignored so the expression can be evaluated properly. In test case two, which has no spaces, the expression can be read as is. Since test case 1 has a comparison operator, the expected result will be a1(true), while the expected result from test case 2 will be an int value of 9. In test case 1, the left side of the comparison operator is evaluated first, since "+" has higher precedence than ">". After the addition is complete, the comparison operator is used, and since 4 > 2, boolean true is returned as the int value of 1.

For test case 2, since it contains parentheses, the expression within the parenthesis will be evaluated first. Then the result of that evaluation is then multiplied by the "outside" value of 3. While multiplication has higher precedence than addition, because of the parentheses, order of operations dictates that whatever is within the parentheses be evaluated first; in a way overwriting precedence in this particular instance. Test case two becomes 3*3, which of course equals 9, and proper evaluation has been done.

How we contributed:

Shane worked on the main algorithm starting with the two methods for infix to postfix and solving postfix covered in class and modifying them so that they could be used as a single method. This includes creating the two stack structures and laying the groundwork for the overall evaluation method, needed branches for each time of operator or number that could be found, and the definition for each needed support. Once this was laid out both Samuel and Dalton helped to make modifications to this method both as errors were found and as ways to improve the system were discussed. All members had a hand in writing the supporting methods, with Dalton writing the majority of the method to perform the equation designated by each operator, and Samuel writing the method to establish precedence for each operator, as well as implementing the needed code to read information from a file.

Improvements:

A simple improvement could be the addition of more possible operators. You could also add in a method to perform the same calculation with prefix equations. A GUI could be added in to make the process more streamlined for a user, thus not requiring them to see the source code.