Gebze Technical University
Computer Engineering

CSE 222 - 2019 Spring

HOMEWORK 3 REPORT

GÖKHAN HAS
161044067

Course Assistant: Özgü GÖKSU

1 INTRODUCTION

1.1 Problem Definition

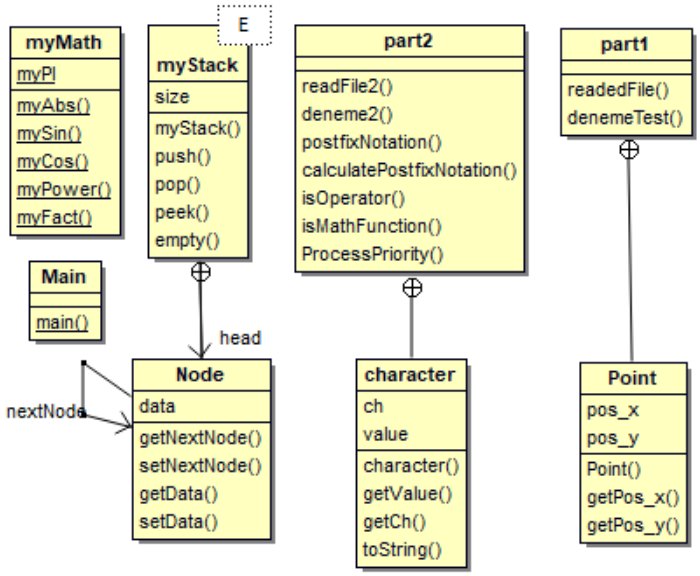
This assignment consists of two parts. In the first part, we are asked to calculate the number of whites from us. The second part gives us an infix statement. We need to convert this expression to postfix. In both parts, there is reading from the file. After reading the file needs to split in detail.

1.2 System Requirements

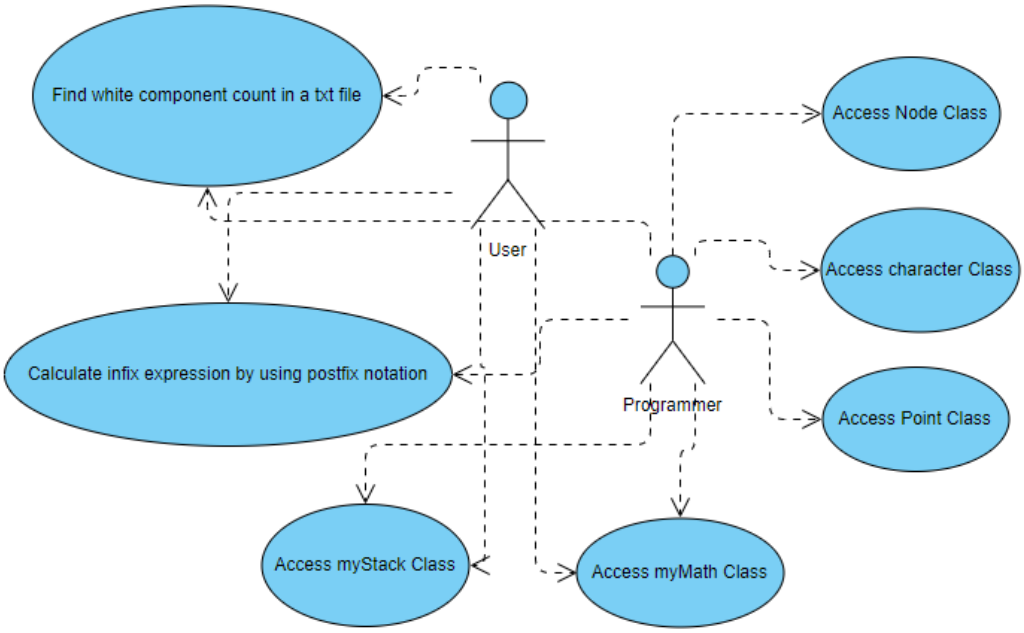
Java codes need JVM to work. For JVM, the Java Runtime Environment (JRE) must be installed. That is, the JAVA must be installed on the technological device. So these codes can work on any device with JAVA installed. As it is known, for JAVA codes to work, it is enough to move files with a class extension. The files with the class extension are 18.5 KB (18.992 bytes). Some smartphones may not support JAVA. (See: https://www.java.com/en/download/faq/java_mobile.xml) 128 KB of memory is not sufficient for program execution.

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagrams



2.3 Problem Solution Approach

This assignment consists of two different parts. So I started the assignment by defining two separate class structures, part1 and part2. As the data structure, we need to use Stack in this homework. I wrote my own class called myStack. Push, pop, peek and empty methods were written according to Stack methods. Node class was used as auxiliary class. The Stack also features LIFO (Last in First Out). Written in accordance with this feature. Time complexity was written the same as the normal Stack. Then you can look at the time complexity section.

Part1 was used as an auxiliary class in a private Point class. This class will be used to hold the location of any element in the matrix. Class structure can be seen in the class diagrams. I just wrote the methods I needed for homework. Then a method called readedFile was written. This method reads test_file_part1.txt in the src directory where the task is located, assigns it to the matrix and performs the desired operations. I read the values from the file by pushing the stack in my own. Then I pop the stack and assign it to the matrix. There is something to pay attention to here in the matrix I'm doing the reverse order. Then, I'm pushing myStack when it's equal to 1 by navigating each element of the matrix. myStack is taking object in Point type here. So on myStack, I'm holding the coordinates of that one. Then I look at the neighbors until myStack is empty. If the

neighbors are 1, I push them to myStack and do 0. When myStack is empty, I do count ++. So I finally keep the number of white components in count and write to the screen. (See Time complexity for time complexity.)

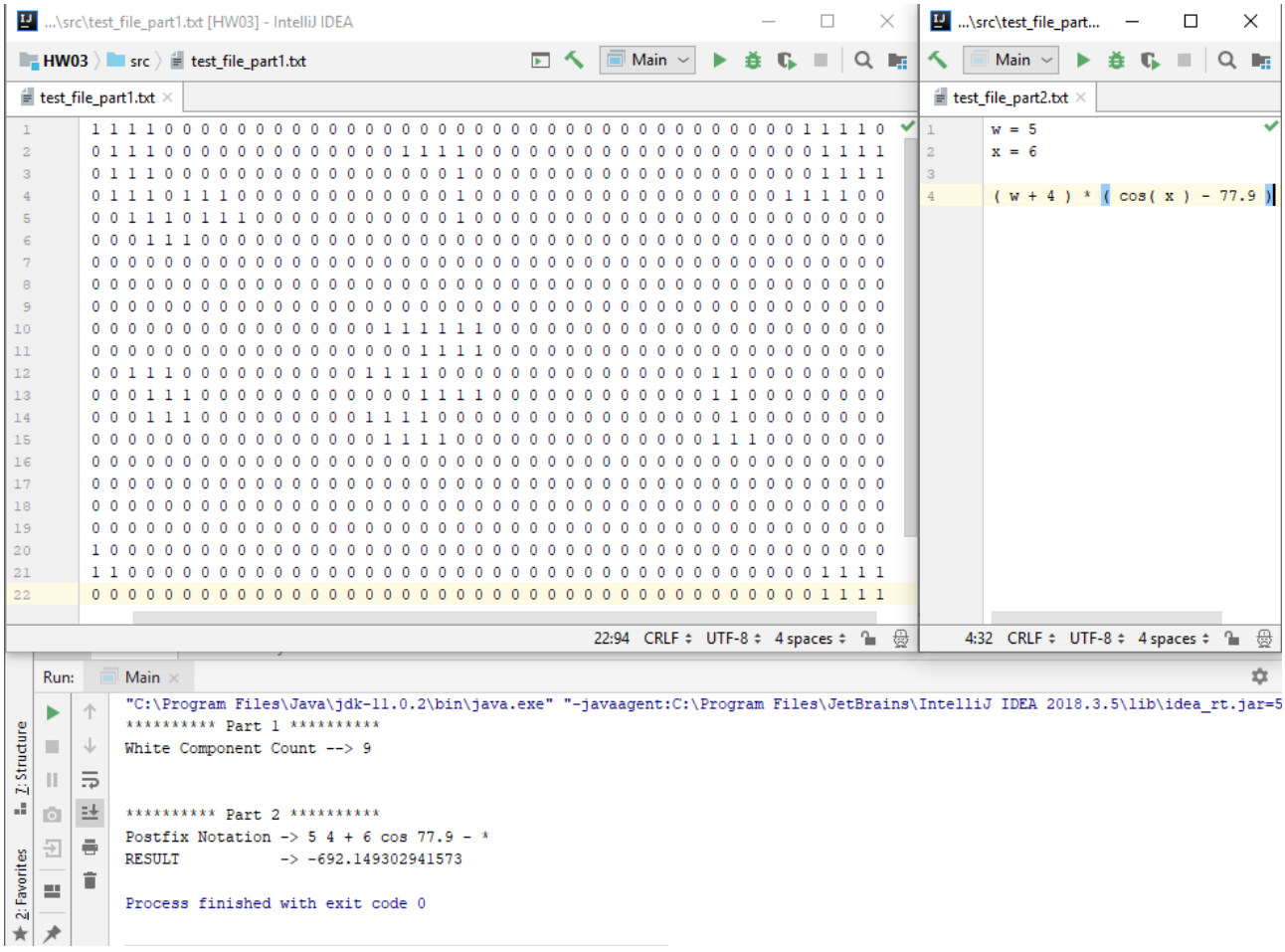
In Part2 I used an inner class and another class, myMath class, as an auxiliary class. sin, cos, abs methods were written in myMath class. While calculating these methods, they were also written for the factorial and pow methods. sin and cos method was written using Taylor Series. Character class is in class. This class is written to read the variabels in the file. (for expressions such as x = 5, w = 6). The character class also has a character type ch and the String type has a value variable. In the example above, ch will hold x and value will hold 5. I create a myStack of this character type and push the varieties in the file to this myStack. There is a blank line between expression and variables in the file. Completing the read operation according to that blank line. I'm assigned an expression to expressionString after it has arrived. Then I create an array of my character class and pop the variables in myStack. Then, if I'm browsing the elements in expressionString, if the array of character type is the same as the variable ch, then I assign the value variable to that value. Then I create a new String and I'm assigned it into expressionString. Then expressionString now has full numbers. I'm sending this string to the postfixNotation method. This method is converted to String by converting the infix notation sent using myStack to String type postfix notation. This expression is also printed on the screen. Then, this postfix string is sent to the calculatePostfixNotation method. Again in this method, using String type myStack, is calculated and returned as a double. This is is also printed on the screen.

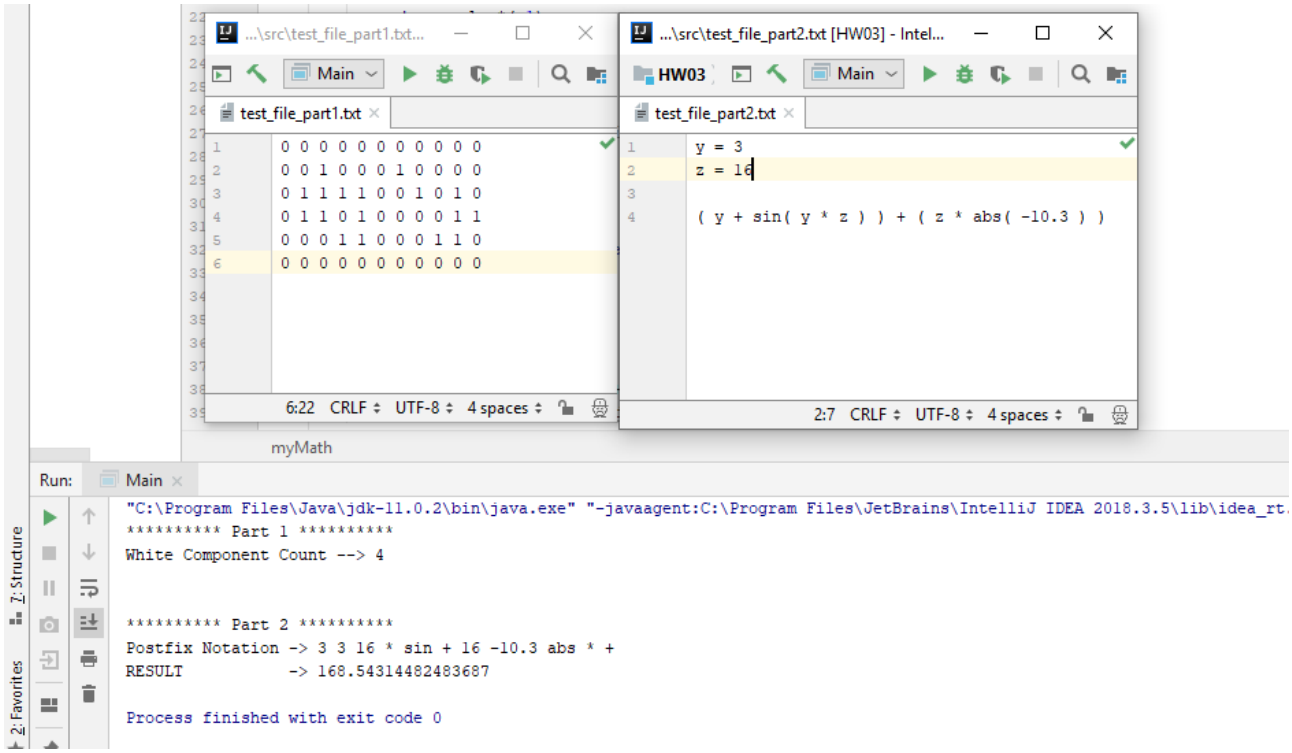
3 RESULT

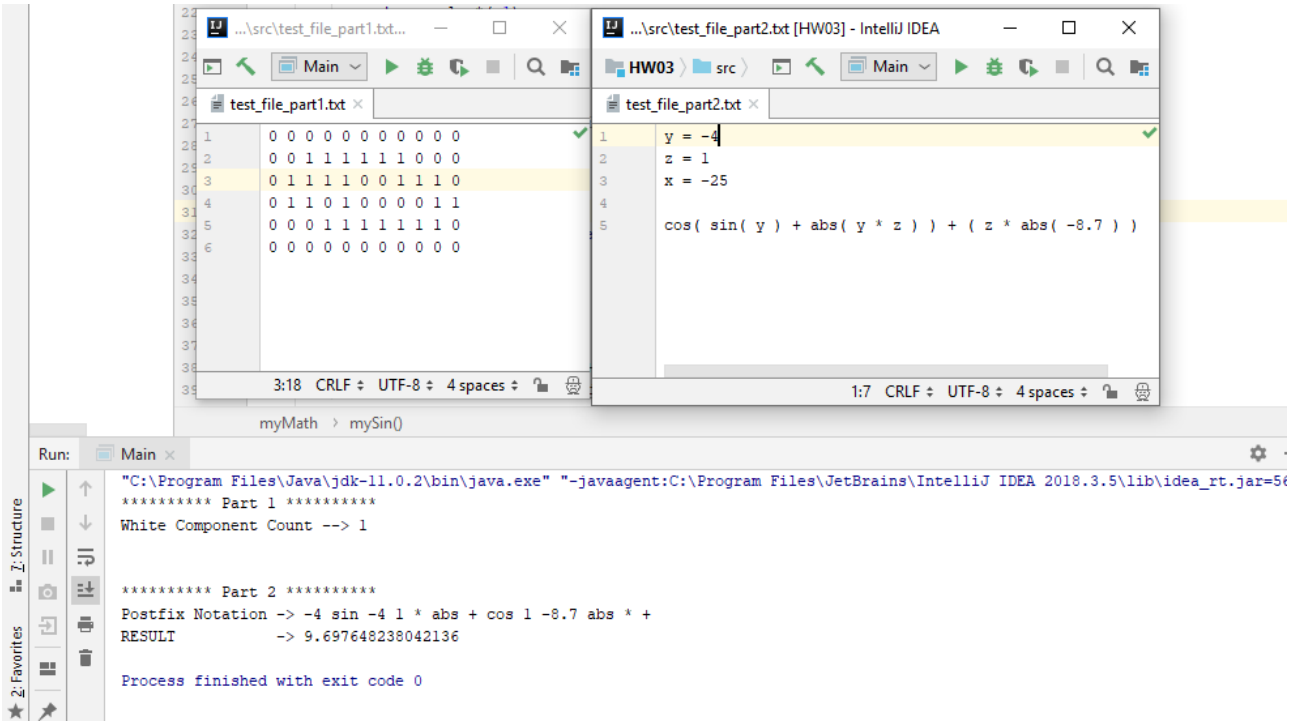
3.1 Test Cases

First I tested the files in the given files. Then I tested it using a lot of files. And I added them to the next running results.

3.2 Running Results







4 TIME COMPLEXITY

4.1 myStack Class

Node class methods → O(1)
myStack() → O(1) (Constructor)
push(E newObject) → O(1)
pop() → O(1)
peek() → O(1)
empty() → O(1)

Methods are O(1). Because I added the head node and removed head node. I never use loops.

4.2 part1 Class

Point class methods → O(1)
readedFile() method → O(n*m) n is matrix element and m is number of longest branched ones.
Test() method → Its also O(n*m).Because I called readedFile() method in this method.

4.3 part2 Class

character Class Methods → O(1)
postfixNotation() Method → O(expressionString's length – number of spaces)
→ O(n)

calculatePostfixNotation() Method() → O(postfixString's length – number of spaces)
→ O(n)

isOperator() Method → O(1)
isMathFunction() Method → O(1)
ProcessPriority() Method → O(1)
readFile2() Method → O(n) Because I called postfixNotation and calculatePostfixNotation method in this method. And different loop is file row's number.

4.4 myMath Class

myAbs method → O(1)
mySin method → O(n)
myCos method → O(n)
myPower method → O(n)
myFact method → O(n)