

**Gebze Technical University  
Computer Engineering**

**CSE 222 – 2019 Spring**

**HOMEWORK 3 REPORT**

**STUDENT NAME  
Can BEYAZNAR**

**STUDENT NUMBER  
161044038**

# **First Part of Homework**

## **1 INTRODUCTION**

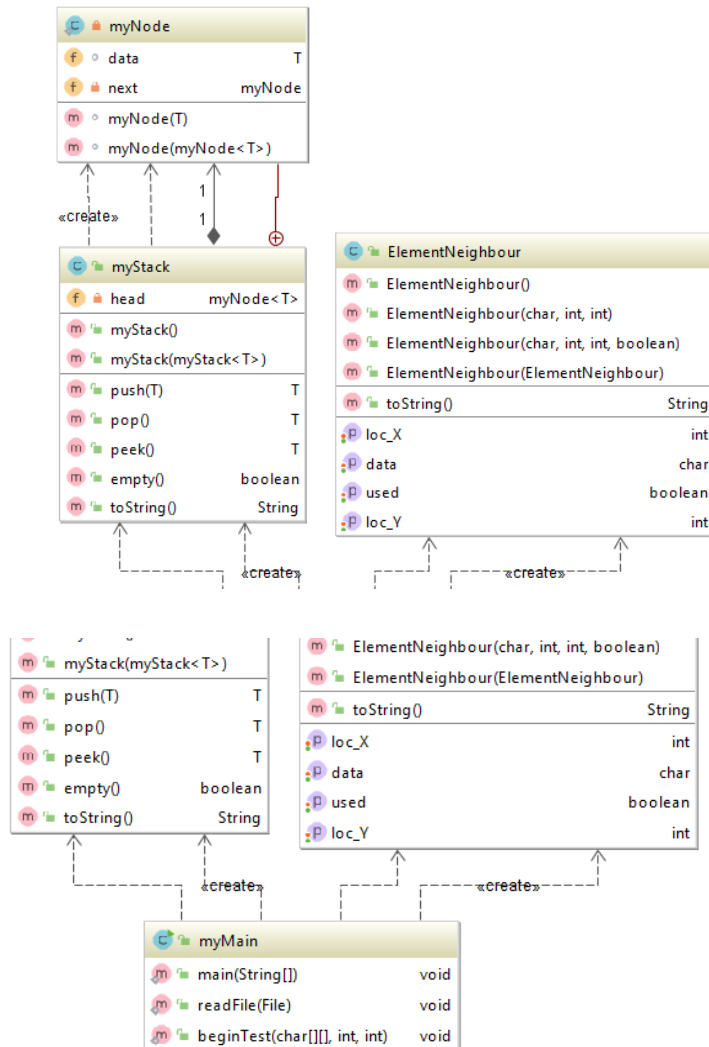
### **1.1 Problem Definition**

In this first part of homework, we want to count the neighbours in a two-dimensional array with using stack. The numbers we are looking for (which is '1') are connected to each other from right to left and from top to bottom. And we want to find how many areas we have that the numbers connected each other.

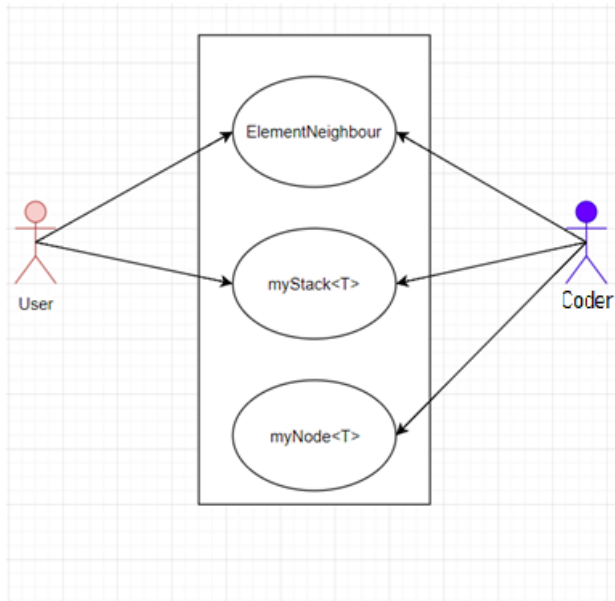
### **1.2 System Requirements**

## 2 METHOD

## 2.1 Class Diagrams



## 2.2 Use Case Diagrams



## 2.3 Problem Solution Approach

To solve this problem, firstly we have to make our stack class. Because we need to use the stack to the best solve this problem. When we create our stack class, one of the best way to link the elements is the linked list. Because our stack's methods are push, pop, peek, empty. And when we implement these methods, if we use arraylist or something else we have to know size of the elements or the capacity. But when we use linked-list we can easily create new element or delete an element in our list. To solve this solution efficiently, I made my class generic. also I created ElementNeighbour class to solve this problem in a good way. In this class we keep our data as "char" and we keep the location of elements and this element is used or not as boolean data type. After we make these classes we can solve the problem. Before assigning the data in the file to our two-dimensional array, we calculate how many lines and columns we have in our array. Next, a new array is created and assigned according to the number of rows and columns. After these operations are done, two nested loops are created and continue until the end of the array. In the meantime, if the number 1 in the array, the number of this element's elementNeighbour type parameter is sent to our stack. Then we look at the neighbors to see if there is a

number 1 next to our position. If there is a number 1, this neighbor element is added to our heap and the next process is done from it. After looking at all the neighbors of each element, this element is removed from the stack. When the stack is empty, the variable holding the number of zones is increased. These operations continue until the array is finished. And returns the number of zones.

## 2.4 Time Complexity

myStack<T> class' all the methods are  $O(n)$ . (outside the node class I use as an assistant. )

myNode<T> class' all the methods are  $O(1)$ .

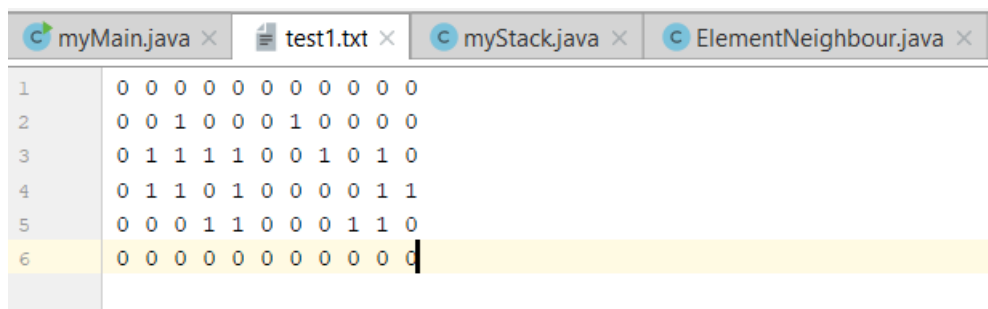
ElementNeighbour class ' all the methods are  $O(1)$ .

The time complexity of the readFile method is  $O(n)$ .

The time complexity of the beginTest method is  $O(n*x)$  ( X is the number of neighbours 1. )

## 3 RESULT

### 3.1 Test Cases



```

myMain.java × test1.txt × myStack.java × ElementNeighbour.java ×
1 0 0 0 0 0 0 0 0 0 0 0
2 0 0 1 0 0 0 1 0 0 0 0
3 0 1 1 1 1 0 0 1 0 1 0
4 0 1 1 0 1 0 0 0 0 1 1
5 0 0 0 1 1 0 0 0 1 1 0
6 0 0 0 0 0 0 0 0 0 0 0
  
```

[illegible]

### 3.2 Running Results

Test 1 :

Result: 4

—○—○—○—○—○—○—

Test 2 :

Result: 9

Process finished with exit code 0

# Second Part of Homework

## 1 INTRODUCTION

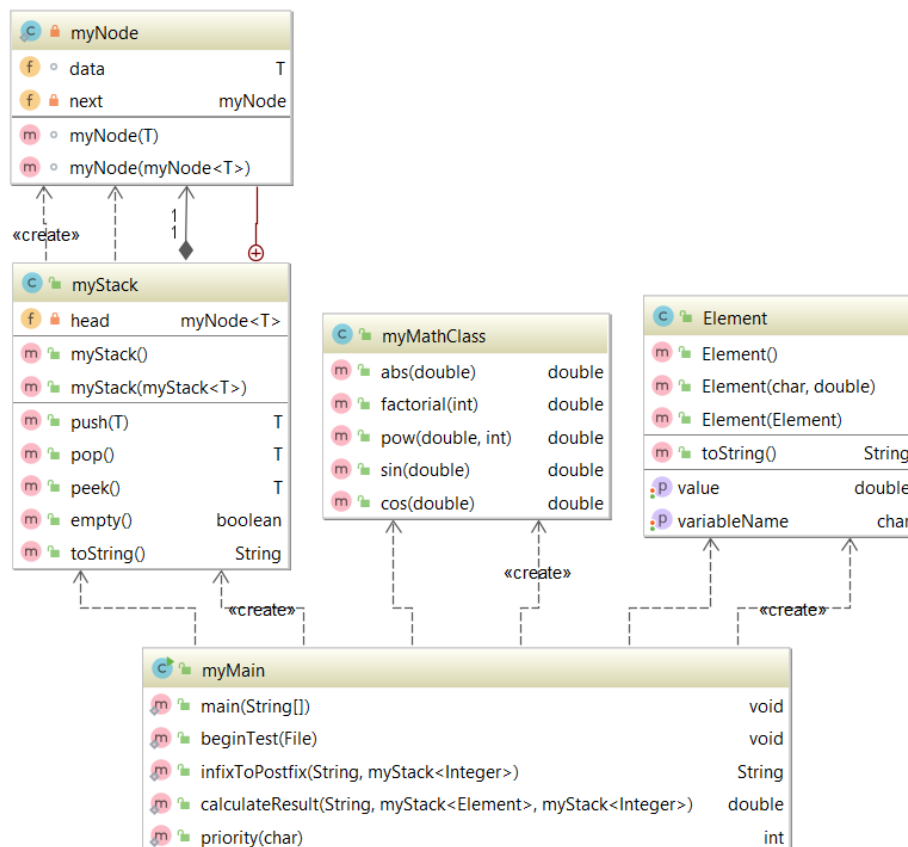
### 1.1 Problem Definition

In the second part of homework, we have to read the text file that has variables and an infix equations. We must convert the equation into postfix. Variables given after this conversion are requested to be put in place.

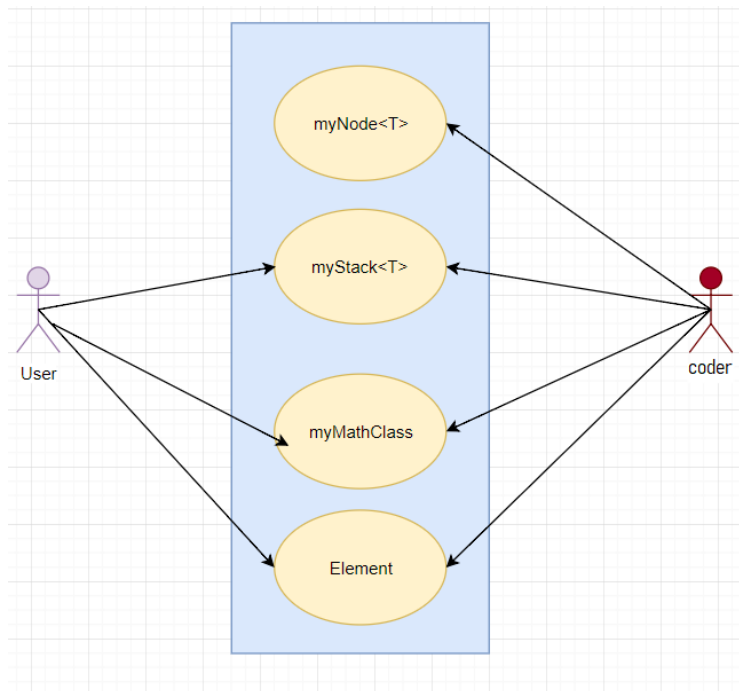
### 1.2 System Requirements

## 2 Method

### 2.1 Class Diagrams



## 2.2 Use Case Diagrams



## 2.3 Problem Solution Approach

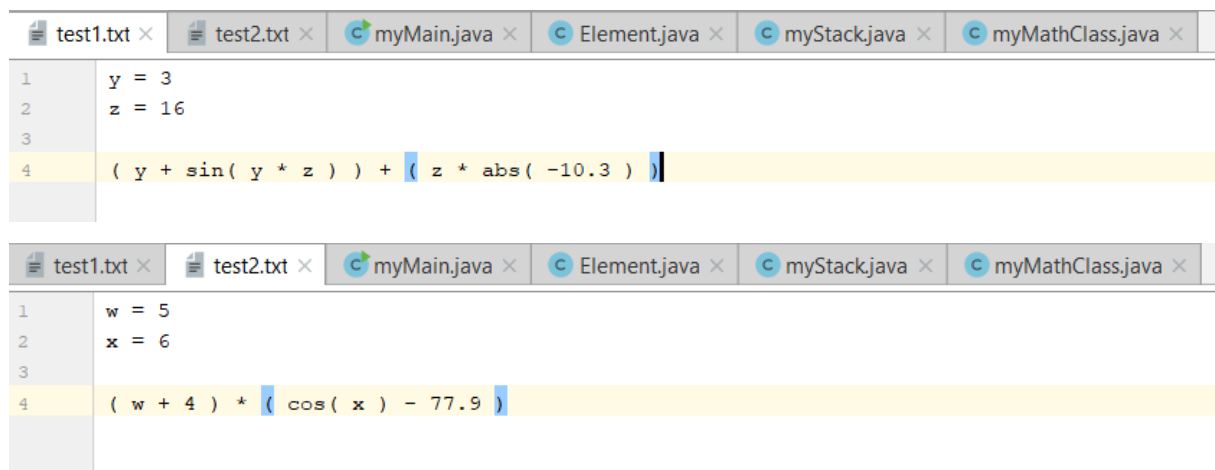
We need to create our own stack class as in the first part of the assignment before solving the problem. For special functions that will then be used in homework (like `sin()`, `cos()`, `abs()` ) we have to make our own math class. Because while we are solving the problem this class will be so useful. Then we need to create element class to keep every special element we read from the file. We need to keep the name and value of the variable in this class. After the making all the classes, we can solve our problem. First, our variables from the top of the equation are push to our stack. Then the equation in the last part is sent to an array and sent to the method that we will convert to postfix. This method creates a loop that runs along the length of our array. This cycle, according to the character we are in the array, entering certain conditions makes certain operations. In this process, if a letter is entered, it is discarded in our new directory. There are also special conditions for special functions such as `sin` `cos` and `abs`. Out of these conditions, if these functions are written in the array, the new array will add the position of these functions in a special stack. This process will be useful for us when calculating. After converting to



Postfix, we now go to the calculation method. In the calculation method, we need to know our unique function index's. In this method, a cycle like the previous method is created. And we operate according to the character we get in each position. if the character we obtain is a variable, the value of that variable is taken by looking at the heap of our variables and added to a new stack. If we see an operator, this operator is unary or binary by looking at the value of the variables we take a one or two values. According to these values we obtain, we return the result after the end of our loop.

## 3 Result

### 3.1 Test Cases



```
test1.txt × test2.txt × myMain.java × Element.java × myStack.java × myMathClass.java ×
1 y = 3
2 z = 16
3
4 ( y + sin( y * z ) ) + ( z * abs( -10.3 ) )

test1.txt × test2.txt × myMain.java × Element.java × myStack.java × myMathClass.java ×
1 w = 5
2 x = 6
3
4 ( w + 4 ) * ( cos( x ) - 77.9 )
```

## 3.2 Running Results

test1.txt :

y = 3.0

z = 16.0

( y + sin( y \* z ) ) + ( z \* abs( -10.3 ) )

Postfix String:

y y z \* S + z 0 10.3 - A \* +

Result: 168.54314435198438

-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-

test2.txt :

w = 5.0

x = 6.0

( w + 4 ) \* ( cos( x ) - 77.9 )

Postfix String:

w 4 + x C 77.9 - \*

Result: -692.1493028584729

-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-