

**Gebze Technical University
Computer Engineering**

CSE 222 - 2019 Spring

HOMEWORK 3 REPORT

**MUHAMMET BURAK ÖZÇELİK
151044050**

Course Assistant: Özgü GÖKSU

1 INTRODUCTION

1.1 Problem Definition

This program is used to analyze and return a given expression. How and under what circumstances the program analyzes the statement will be explained in the following section.

Input

The input will be provided through the commandline as a path to file containing a single infix expression (an example is provided at the moodle page: `infix_test_file_1.txt`). The expression's tokens will be space separated.

Output

The infix expression's value.

1.2 System Requirements

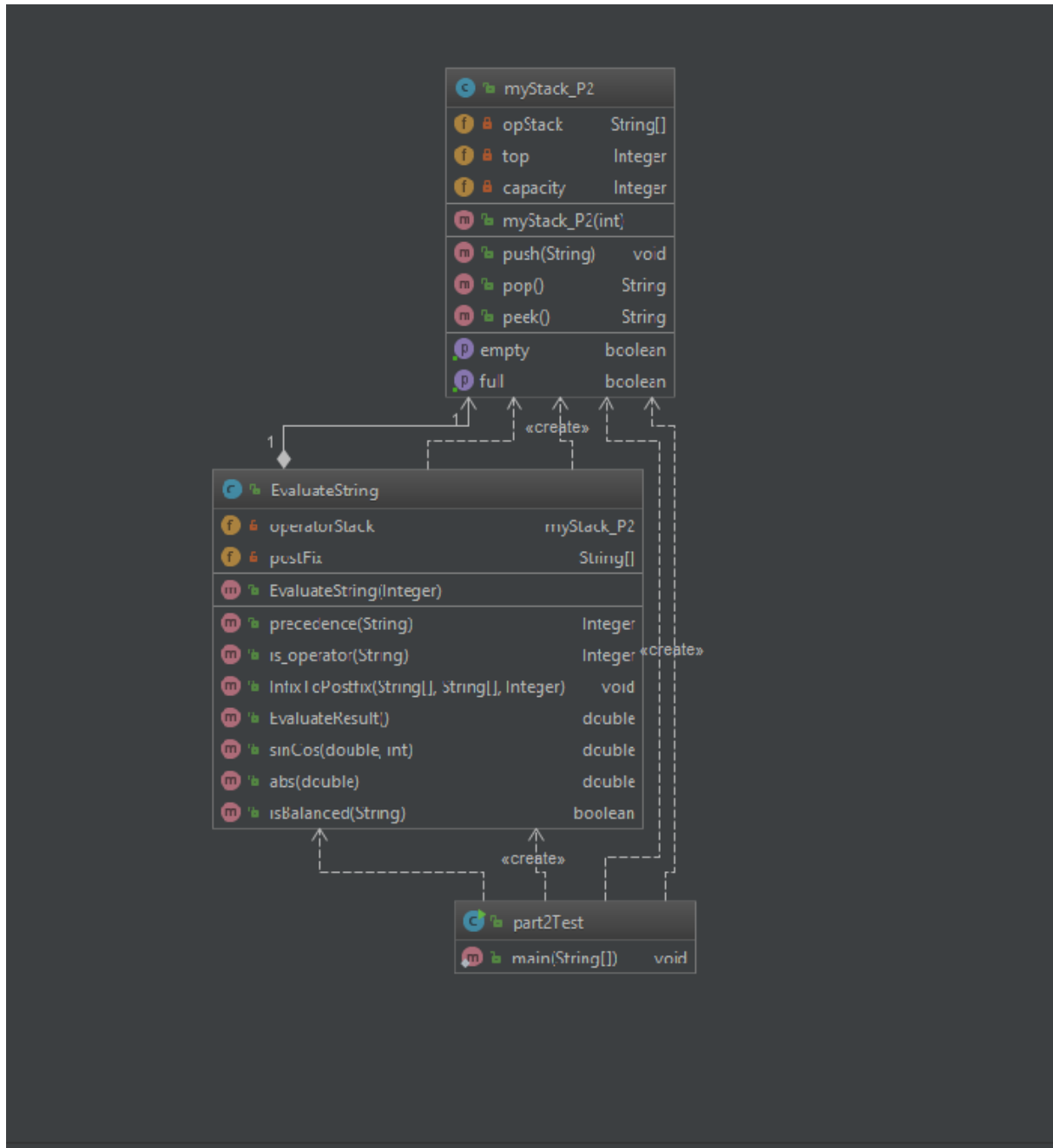
As system requirement, it is the stack class that we implement and a lot of String operation in this program.

In general, I used two classes to solve the problem. One of them is the `myStack` class and the `EvaluateString` class and a text file that we read the expression. We will read the expressions from a text file in the following format.

This program will run on IntelliJ and with Java 8 version on PC.

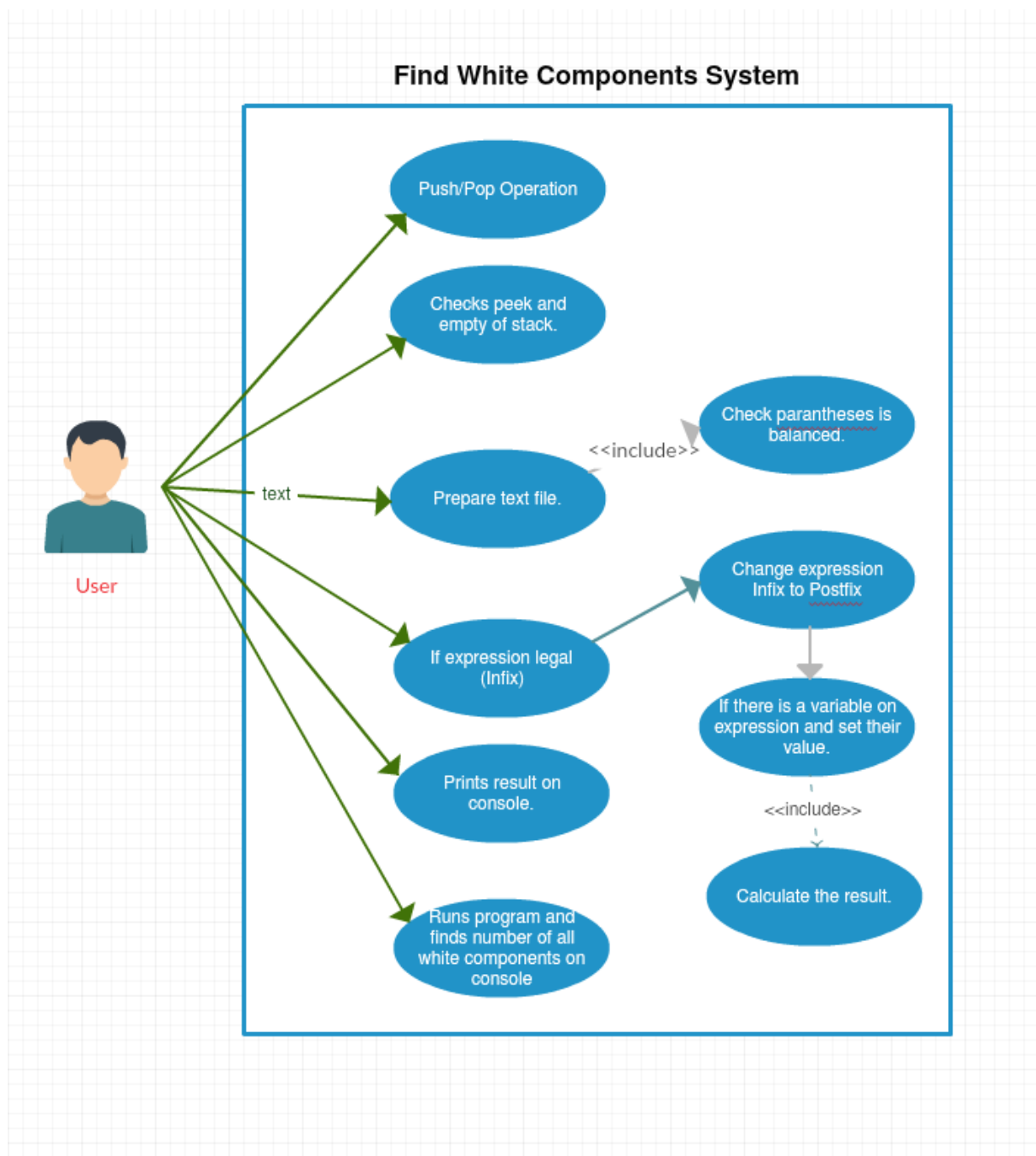
2 METHOD

2.1 Class Diagrams



Class diagrams of the program are shown above.

2.2 Use Case Diagrams



Since it is a single user as user, it will be able to access all public methods. The simple use - case diagram is shown above. The user will be able to prepare the text file and obtain the result of Expression in the file.

2.3 Problem Solution Approach

First I read the file. If there is a variable I break it into an list of variable names and values. If the line read is an expression, I separate it by space, and inside the sin (, cos (or abs (if any) next to '(' I add. In this way, the expression is converted to postfix sin (, cos (, abs (instead of putting other characters do not change the number of parenthesis of the expression.

Then I insert the infix expression in this list into a string[]. In the expression in this array, I substitute respectively '!', '?' and '#' instead of sin(, cos(and abs(. Now our infix expression is ready to convert to postfix.

I call the InfixToPostfix method in this class by creating the EvaluateString object. In the InfixToPostfix method, I'm pushing a '(' into the stack first, so I'm trying to determine it. And if the operator in the string is seen, I'm pushing the stack into the stack in order of priority. If the character ')' is seen, it is popped from the stack until the previous '(' character is seen and appended into the postfix array. If the incoming element is not an operator, it is appended directly into the postfix array. But if the element is the same as the variable name that we read from the file, its value is appended to the postfix instead of that element's name.

Now postfix array is ready to calculate. I call EvaluateResult() method.

In this method, the values in the array are pushed into the stack until an operator is seen in the postfix array. If an operator is seen and the operator is binary, the last two elements are popped and processed and pushed back into the stack. If an operator is seen and the operator is unary, the last element popped and processed and back into stack.

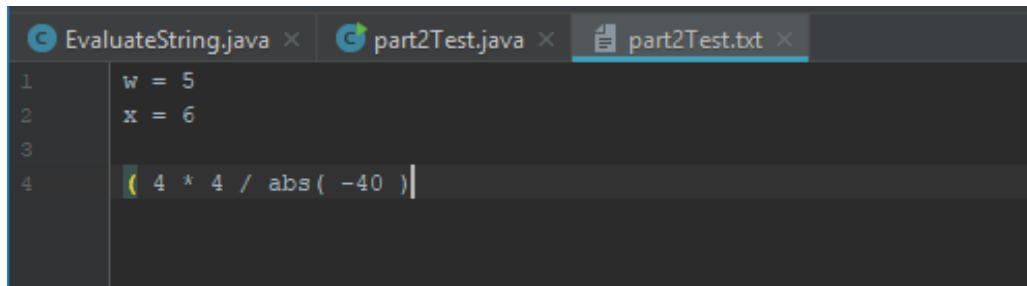
When the end of the array, the element in the stack is the result of the expression and is returned.(double).

Time complexity of this program is $\theta(n)$ because all program runs until the postfix or infix array's length.

3 RESULT

3.1 Test Cases

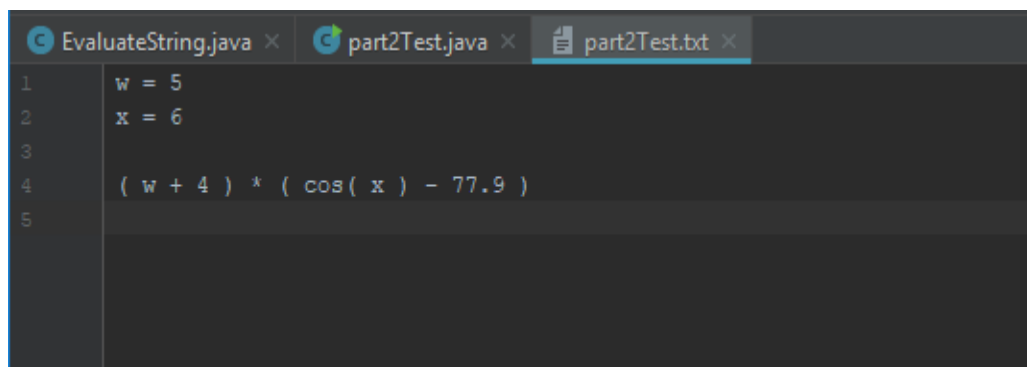
- I tested it when parantheses is not balanced.



```
EvaluateString.java x part2Test.java x part2Test.txt x
1 w = 5
2 x = 6
3
4 ( 4 * 4 / abs( -40 ) |
```

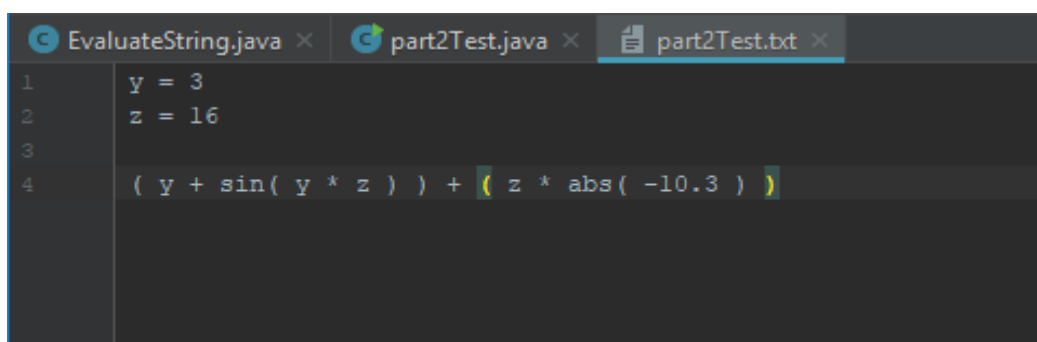
Figure 1

- I tested it with test file on moodle.



```
EvaluateString.java x part2Test.java x part2Test.txt x
1 w = 5
2 x = 6
3
4 ( w + 4 ) * ( cos( x ) - 77.9 )
5
```

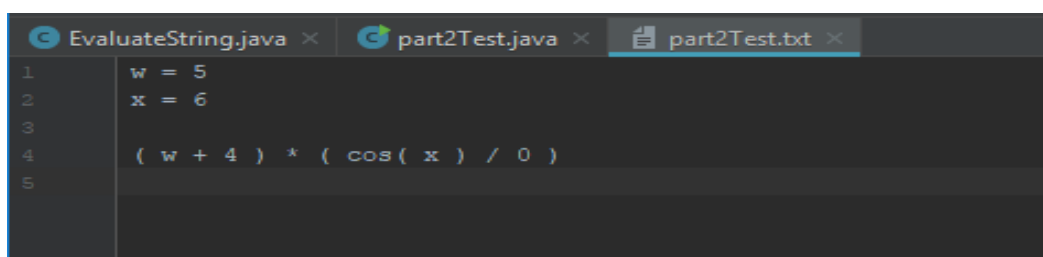
Figure 2



```
EvaluateString.java x part2Test.java x part2Test.txt x
1 y = 3
2 z = 16
3
4 ( y + sin( y * z ) ) + ( z * abs( -10.3 ) )
```

Figure 3

- I tested it with test file and I tried to divide an expression with '0' zero.



```
EvaluateString.java x part2Test.java x part2Test.txt x
1 w = 5
2 x = 6
3
4 ( w + 4 ) * ( cos( x ) / 0 )
5
```

3.2 Running Results

- The result of Figure 1.

```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...  
java.lang.Exception: Paratheses Not Balanced  
  
Process finished with exit code 0
```

- The result of Figure 2

```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...  
Infix ----> : (w+4)*(cos((x)-77.9) Replaced instead of sin( = '!((', cos( = '?(', abs( = '#(' on Postfix  
PostFix ---> : w4+x?(77.9-^  
Result      : -692,15  
Process finished with exit code 0
```

- The result of Figure 3.

```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...  
Infix ----> : (y+sin((y*z)))+(z*abs((-10.3)) Replaced instead of sin( = '!((', cos( = '?(', abs( = '#(' on Postfix  
PostFix ---> : yyz^!+(z-10.3#('^+  
Result      : 168,54  
Process finished with exit code 0
```

- The result of Figure 4.

```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...  
Infix ----> : (w+4)*(cos((x)/0) Replaced instead of sin( = '!((', cos( = '?(', abs( = '#(' on Postfix  
PostFix ---> : w4+x?(0/^  
java.lang.Exception: Expression can cot divide zero!  
  
Process finished with exit code 0
```

3.3 Algorithm Analysis

All operations are performed and shown in the Main test scenario. However, the sample outputs and algorithm analysis of the methods are given below.

- **InfixToPostfix Method()**

```
/**
 * Infix is used to convert the expression to postfix.
 * It pushes open brackets and operators into the stack. Operators push the stack by looking at the priorities
 * when disposing into the heap. Also variables append postfix array. When it sees closed brackets, it appends
 * the operators into the postfix array until he sees open brackets. Finally, it adds the values of the names
 * that are the same as the variables read from the file. Then, Postfix is ready to calculate!
 * @param infix The infix expression read from the file.
 * @param val Variable names and values read from the file.
 * @param capacity For stack to use allocate stack. (String length)
 */
public void InfixToPostfix(String [] infix, String [] val, Integer capacity)
{
    myStack_P2 newStack = new myStack_P2( capacity: capacity+1);
    newStack.push( "(" );
    String popOp;
    int counter = 0;

    for (int i = 0; i < infix.length; i++) {
        if(infix[i].equals("(")){
            newStack.push(infix[i]);
        }
        else if(is_operator(infix[i]) == 1 || is_operator(infix[i]) == 2){
            popOp = newStack.pop();

            while( (is_operator(popOp) == 1 || is_operator(popOp) == 2 ) &&
                (precedence(popOp) >= precedence(infix[i])) )
            {
                postFix[counter] = popOp;
                popOp = newStack.pop();
                counter++;
            }
            newStack.push(popOp);
            newStack.push(infix[i]);
        }
        else if(infix[i].equals(")")){
            popOp = newStack.pop();
            while(!(popOp.equals("("))){
                postFix[counter] = popOp;
                popOp = newStack.pop();
                counter++;
            }
        }
        else if( is_operator(infix[i]) == 0){
            postFix[counter] = (infix[i]);
            counter++;
        }
    }

    while(!(newStack.peek().equals("("))){
        postFix[counter] = (newStack.pop());
        counter++;
    }

    System.out.print("PostFix --->\n");
    for (int i = 0; postFix[i] != null ; i++) {
        System.out.print(postFix[i]);
    }
    System.out.println();
    for (int i = 0; i < counter; i++) {
        for (int j = 0; j < val.length; j+=2) {
            if(postFix[i].equals(val[j])){
                postFix[i] = postFix[i].replaceAll(postFix[i],val[j+1]);
            }
        }
    }
}
```


Time complexity of InfixToPostfix Method is $\theta(n*m)$ because for loop repeated until infix is null and in this loop (n), there is a while loop is repeated if stack's last element's precedence is smaller than new operator. (m)

- **EvaluateResult() Method**

```
/**
 * The operator pushes the values into the stack until the operator arrives. When the operator arrives, if the
 * operator is binary, pops the last two from the stack else if the operator is unary pop the last element from
 * stack and calculate the expression. Then it pushes stack. When the expression is completed, the popped element
 * will be the result.
 * @return Result of infix to postfix expression. (Double)
 * @throws Exception Can not Divide Zero
 */
public double EvaluateResult() throws Exception {
    myStack_P2 resultStack = new myStack_P2( capacity: 100);

    for (int i = 0; postFix[i] != null ; i++) {
        if(is_operator(postFix[i]) == 0){
            resultStack.push(postFix[i]);
        }
        else if(is_operator(postFix[i] ) == 1){
            double vall = Double.parseDouble(resultStack.pop());
            double val2 = Double.parseDouble(resultStack.pop());

            if(postFix[i].equals("*")){
                resultStack.push(String.valueOf((val2*vall)));
            }
            else if(postFix[i].equals("/")){
                if((vall) == 0){
                    throw new Exception("Expression can not divide zero! ");
                }
                resultStack.push(String.valueOf((val2/vall)));
            }
            else if(postFix[i].equals("-")){
                resultStack.push(String.valueOf((val2-vall)));
            }
            else if(postFix[i].equals("+")){
                resultStack.push(String.valueOf((val2+vall)));
            }
        }else if(is_operator(postFix[i]) == 2){
            double vall = Double.parseDouble(resultStack.pop());
            if(postFix[i].equals("!")){
                resultStack.push(String.valueOf(sinCos(vall, control: 0)));
            }
            else if(postFix[i].equals("?")){
                resultStack.push(String.valueOf((sinCos(vall, control: 1))));
            }
            else if(postFix[i].equals("#")){
                resultStack.push(String.valueOf((abs(vall))));
            }
        }
    }

    return Double.parseDouble(resultStack.pop());
}
```

EvaluateResult method's complexity is $\theta(n)$ because there is a one for loop and it repeated when postfix array is null.

- **sinCos() Method**

```

/**
 * https://introcs.cs.princeton.edu/java/13flow/Sin.java.html
 * This solution help me for this issue. I edited for my problem.
 * @param degree x's value of cos(x)/sin(x).
 * @param control If control 1 this expression cos(x) else sin(x).
 * @return It returns cos or sin method's result.
 */
public double sinCos (double degree, int control){
    degree = degree % 360;
    //Sinus complement with -90;
    if (control == 1) {
        degree = 90 - degree;
        if(degree < 0){
            degree += 360;
        }
    }
    // degree * radian
    double x = 0.0174533 * degree;
    // convert x to an angle between -2 PI and 2 PI
    x = x % (2 * 3.1415);

    if(degree % 180 == 0 && control == 0){
        return 0.0;
    }
    if(degree % 180 == 0 && control == 1){
        return 0.0;
    }
    // compute the Taylor series approximation
    double term = 1.0;    // ith term = x^i / i!
    double sum = 0.0;    // sum of first i terms in taylor series

    for (int i = 1; term != 0.0; i++) {
        term *= (x / i);
        if (i % 4 == 1) sum += term;
        if (i % 4 == 3) sum -= term;
    }

    if(control == 0){
        return (sum);
    }
    if(control == 1){
        if(degree % 180 == 0 ){
            return 1.0;
        }else{
            return (sum);
        }
    }
    return 0;
}

```

sinCos method's complexity is $\theta(x)$. X beacuse for loop is repeated until incoming value of degree. It is not connected array's lentgh.

- **Abs() Method**

```

/**
 * Returns the absolute value of the number.
 * @param v Variable (double)
 * @return Positive value of v.
 */
public double abs (double v){
    if(v<0){
        return (v*(-1));
    }
    return v;
}

```

Abs Method's complexity is $\theta(1)$.

- **isBalanced() Method**

```
/**
 * It checks number of expression's parentheses and its valid. The brackets are correct
 * until the parentheses are closed. That's all.
 * @param text Infix expression. (String)
 * @return True or False.
 */
public boolean isBalanced (String text){
    if(text == null){
        return false;
    }
    char [] test;
    myStack_P2 testStack = new myStack_P2(text.length());
    test = text.toCharArray();
    for (int i = 0; i < test.length() ; i++) {
        if(test[i] == '('){
            testStack.push("(");
        }
        else if(test[i] == ')'){
            if( testStack.isEmpty()){
                return false;
            }else{
                testStack.pop();
            }
        }
    }
    return testStack.isEmpty();
}
```

isBalanced() Method's complexity is $\theta(n)$. Because for loop repeated infix expression's length.

- **isOperator() Method**

```
/**
 * Allows us to understand whether the incoming string is an operator or variable.
 * @param symbol
 * @return If operator returns 1 or 2, else returns 0.
 */
public Integer is_operator(String symbol)
{
    if(symbol.equals("*") || symbol.equals("/") || symbol.equals("+") || symbol.equals("-") )
    {
        return 1;
    }
    else if(symbol.equals("!") || symbol.equals("?") || symbol.equals("#"))
    {
        return 2;
    }
    else {
        return 0;
    }
}
```

isOperator() Method's complexity is $\theta(1)$. It is only checks incoming symbol's is equal or not.

- **precedence() Method**

```

/**
 * It allows you to understand which operator has priority to process. (Integer return)
 * @param symbol Incoming operator of Method. (String)
 * @return Returns precedence of operator.
 */
public Integer precedence(String symbol)
{
    if (symbol.equals("?(")) {
        return 3;
    }
    else if (symbol.equals("#(")) {
        return 3;
    }
    else if (symbol.equals("!(")) {
        return 3;
    }
    else if (symbol.equals("*") || symbol.equals("/")) {
        return (2);
    }
    else if (symbol.equals("+") || symbol.equals("-")) {
        return (1);
    }
    else {
        return (0);
    }
}

```

precedence() Method's complexity is $\theta(1)$. It is only checks incoming symbol's is equal and return it's precedence value of integer.

Bonus: Test Case and Explanation

For the program to work, spaces should be placed when writing variable names. For Ex: "x = 6" And sin cos or abs operators should be write on file like this "sin(" "cos(" "abs("

```

1 x = -6
2 y = -3
3 z = 10
4
5 z + 2 * ( cos( abs( z * x ) ) / sin( abs( y * z ) ) ) - ( y * -2 )

```

```

"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
Infix ----> : z+2*(cos((abs((z*x))/sin((abs((y*z)))-(y*-2)) Replaced instead of sin( = '!((', cos( = '?(', abs( = '#(' on Postfix
PostFix ----> : z2zx*#(? (yz*#(! (/^+y-2*-
Result      : 6,00
Process finished with exit code 0

```