# Objectives

Practice Stack, infix, postfix evaluation.

# Problem

Write a program that converts an infix expression to a postfix expression and then evaluates the postfix expression.

# Requirements

* This program must be implemented based on the given requirements and must include all the methods. Solutions off the internet will get zero points.
* Java pre-written Stack class cannot be used or else zero points will be assigned.
* All the required classes must be implemented.
* Your program must work with the given main method. I will use the main method to run your program and you will be graded based on the correctness of running the given main method
* Must submit one file containing all the classes
* All the methods header must be exactly as given. Feel free to change the code inside the method but not method header
* NO pre-written classes such as Stack, Queue, collection classes can be used or else zero points will be assigned.

# You must implement the following classes and interface

1. MyStack interface, the methods are provided in the given java file
2. Class Stack implements myStack.

|  |
| --- |
| Stack |
| -ArrayList<String> s //represents the stack |
| + Stack () // instantiate the ArrayList s  + void push (String token)// push the token to the top of the Stack  +String pop ();//pops and returns the element at the top of the stack  +String peek ();//returns the element at the top of the stack without popping it  +Boolean isEmpty();// returns true if the stack is empty, returns false otherwise  +String toString();//returns a string representing the elements in the stack |

1. Class Expression. See below for some details

|  |
| --- |
| Expression |
| -String exp; |
| +Expression (String e); // assigns e to exp  -int static precedence (String opr) ;  -int static calculate (int num1, int num2, String operation) //see below  +String getPostfix();// returns postfix of exp, see below  +int evalPostfix(); //evaluates the postfix expression, see below |

# Expression class Methods implementation

**Private int static precedence (String opr):** This is a private method since it will be only called in the Expression class.

if the opr is \* or / this method should return 3, if the opr is + or – this method should return 2. (values 2 and 3 are just numbers that I selected)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Private int static calculate (int num1, int num2, String operation):** This method is private since it will only be called in the Expression class**.** this method uses switch or if statements to calculate the expression. For example one of the if statement would be:

if(operation.equals(“\*”))

return num \* num2

you need to have conditional statements for the operations + , -, /

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public String getPostfix**(): This method does not get any parameter

Here is the pseudocode

declare a string to hold the postfix expression

create an object of the Stack class you just created

use the stringTokenizer class to tokenize the **exp** instance variable declared in the Expression class: StringTokenizer st = new StringTokenizer(exp," ");

while(st.hasMoreTokens())

{

String token = st.nextToken();

If the token is any of the operations +, -, /, \*

{

Get the precedence of the token by calling the method precedence

If the precedence is 3

{

while the stack is not empty and the precedence of the top of the stack is 3//the token is \* or /

{

Pop the element at the top of the stack

Concatenate it to the postfix expression

}//end while

}

else if the precedence of the token is 2 // meaning the token is + or -

{

While the stack is not empty and the precedence of the top of the stack is 2 or 3

{

Pop the element at the top of the stack

Concatenate it to the postfix expression

} //end while

}

Push the token to the stack

}

else // token is a number and must be concatenated to the postfix

{

concatenate the token to the postfix expression

}

}// end of while

While the stack is not empty

{

Pop the stack

Concatenate it to the postfix

}//end while

return postfix

//end of the method

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**int evalPostfix() :** this method does not get any parameter

Here is the pseudocode

String post = this.getPostfix() //cretes the postfix expression

Declare a Stack of the stack class that you created

int result = 0;

StringTokenizer st = new StringTokenizer(post," ");  
while (st.hasMoreTokens()) {   
 String token = st.nextToken();

If the token is not \* / + -

Push the token to the stack

else

{

String n1 = Pop the stack

String n2 = pop the stack

Int num1 = Integer.parseInt(n1) //convert string numbers to int. “12” 🡪 12

Int num2 = Integer.parseInt(n2);

Call the method calculate with num1, num2, and token

Push the result from the calculate method to the stack. (Since the calculate method returns an int and the stack is of type string you must concatenate the result with “” before pushing it to the stack. )

}

}//end while

Pop the stack and convert it to an integer and return the result

//end of the method

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Driver class

Complete driver class is provided.

# Sample output:

Infix: 2 + 3 + 7 \* 4 - 2 / 3, postfix: 2 3 + 7 4 \* + 2 3 / - = 33

Infix: 3 - 4 / 2 + 6 \* 3, postfix: 3 4 2 / - 6 3 \* + = 19

Infix: 5 \* 6 - 8 + 2 \* 10, postfix: 5 6 \* 8 - 2 10 \* + = 42

Infix: 4 + 8 \* 3 - 2 / 34, postfix: 4 8 3 \* + 2 34 / - = 28

Infix: 6 - 3 + 6 / 2 \* 4 - 8, postfix: 6 3 - 6 2 / 4 \* + 8 - = 7