CALIFORNIA STATE POLYTECHNIC UNIVERSITY
Computer Science Department
CS 2400-0x ***T. Nguyen/F22***

PROJECT: **3**
DUE DATE: March 19, 2022

Description:

We are used to writing arithmetic expressions with the operator between the two operands: a + b or c / d. If we write a + b * c, however, we must apply precedence orders to avoid ambiguous evaluation. This type of expression is called **infix** expression. There are two other types of different but equivalent ways of writing expressions.

Infix: X + Y: Operators are written in-between their operands. Infix expression needs extra information to make the order of evaluation of the operators clear: precedence and associativity, brackets ( ). For example, A * (B + C) / D.
Postfix: X Y +: Operators are written after their operands. The above infix expression can be written using postfix notation as A B C + * D /
Prefix: + X Y: Operators are written before their operands. The above infix expression can be written using prefix notation as / * A + B C D

More examples of the above three expressions are given below:

Infix	**Postfix**	**Prefix**
a + b + c	a b + c +	+ + a b c
a + b * c	a b c * +	+ a * b c
(a + b) * (c - d)	a b + cd - *	* + a b – c d

You are to implement two class methods that can convert an infix expression to its equivalent postfix and evaluate the postfix expression.
A term will be an integer literal. The parameter to your methods will be an array of Strings and so is the return value. Each element of the array can either be an integer literal or a binary operator + - * / ^ (where ^ is the power operator, x^y is x^y) The ^ operator is defined as:
x ^ y ^ z is: x ^ (y ^ z)

You are to implement the class *Expression* which includes the algorithms *convertToPostfix* and *evaluatePostfix*. They are class methods.

Class name: Expression
Conversion of Infix to Postfix:
Method name: **String[] convertToPostfix (String[] infixExpression)**
infixExpression – each element is a token that can be an operator or an integer literal. Operator can be +, -, *, /, or ^. This method will convert an infix to a postfix expression. Throw a *RuntimeException(msg)* if *infixExpression* is not well-formed or not an integer literal.
Conversion of Infix to Postfix:
Method name: **double evaluatePostfix (String[] posfixExpression)**
Throw *RuntimeException(msg)* when *postfixException* is not well-formed or not a number literal.

Write a Java application named **ExpressionTest** that uses the two methods in Expression, evaluates the infix expression specified in the command line to the infix expression and outputs the result. For example:

```
java ExpressionTest 1 + 10.5 / 2
6.25
```

You can assume number literals, operators, and parenthesis will be an arg in the command line. The literals can be integer or double.

You must implement your own generic Stack ADT and use it in this this project.

What to test?

Think of your own cases: correct input, without parentheses or with matched parentheses; and incorrect input (with unmatched parentheses). Your program should catch the error with unmatched parentheses and when one of the operands is missing. Make sure that you deal with all error conditions, for example, pop/top on an empty stack.

Project report: *(PDF format)*

- Page 1: Cover page with your name, class, project, and due date
- Page 2 to 3:
  - Section 1 (**Project specification**): Your ADT description. Description of data structures used and a description of how you implement the ADT
  - Section 2 (**Testing methodology**): Description of how you test your ADT, refer to your testing output. Explain why your test cases are rigorous and complete. Demonstrate that you test each method.
  - Section 3 (**Lessons learned**): Any other information you wish to include.

Turn in:
1. Project report submitted via Canvas.
2. There should be the following Java source files: Your Stack implementation java source files, Expression.java, ExpressionTest.java, Stack ADT. Compress these files into a single zip file and submit it with the following name:

   ```
   zip proj3 Expression.java ExpressionTest.java StackADT*
   scp p3.zip /user/tvnguyen7/cs2400-00#/bronconame-p3.zip
   ```
   where # is your class section, 1 or 2.

   NO package. You should check out your project on the CPP intranet using:
   ```
   unzip bronconame-p3
   javac ExpressionTest.java
   java ExpressionTest
   ExpressionTest ( 2 + 2 ) / 2
   ```

Grading Guide:
- 10%: Project report and project output.
- 80%: Program correctness
- 10%: Coding – efficiency, style, comments, formats

Notes:

The following information is required at the beginning of every source file.

```
//
//    Name:       Last, First
//    Project:    #
//    Due:        date
//    Course:     cs-2400-0x-f22
//
//    Description:
//                A brief description of the project.
//
```

x is either 1 or 2.