CSULB
CECS277

Write a java code to have the user to select to perform calculations or to exit.

(Infix-to-Postfix Converter)

Stacks are used by compilers to help in the process of evaluating expressions and generating machine-language code. In this exercise, we investigate how compilers evaluate arithmetic expressions consisting only of constants, operators and parentheses.

Humans generally write expressions like 3 + 4 and 7 / 9 in which the operator (+ or / here) is written between its operands—this is called infix notation. Computers "prefer" postfix notation, in

which the operator is written to the right of its two operands. The preceding infix expressions would appear in postfix notation as 3 4 + and 7 9 /, respectively.

To evaluate a complex infix expression, a compiler would first convert the expression to post-fix notation and evaluate the postfix version. Each of these algorithms requires only a single left-to- right pass of the expression. Each algorithm uses a stack object in support of its operation, but each uses the stack for a different purpose.

Write class InfixToPostfixConverter to convert an ordinary infix arithmetic expression (**Check to make sure that a valid expression is entered**)

(6 + 2) * 5 - 8 / 4

to a postfix expression. The postfix version (no parentheses are needed) of the this infix expression is

62+5*84/-

The program should read the expression into StringBuffer infix and use a stack class implemented to help create the postfix expression in StringBuffer postfix. The algorithm for creating a postfix expression is as follows:

a) Push a left parenthesis '(' onto the stack.

b) Append a right parenthesis')'to the end of infix.

c) While the stack is not empty, read infix from left to right and do the following: If the current character in infix is a digit, append it to postfix. If the current character in infix is a left parenthesis, push it onto the stack. If the current character in infix is an operator: Pop

operators (if there are any) at the top of the stack while they have equal or higher precedence than the current operator, and append the popped operators to postfix. Push the current character in infix onto the stack. If the current character in infix is a right parenthesis: Pop operators from the top of the stack and append them to postfix until a left parenthesis is at the top of the stack. Pop (and discard) the left parenthesis from the stack.

The following arithmetic operations are allowed in an expression:

+ addition - subtraction * multiplication / division ^ exponentiation % remainder

The stack should be maintained with stack nodes that each contain an instance variable and a reference to the next stack node. Some methods you may want to provide are as follows:

a) Method convertToPostfix, which converts the infix expression to postfix notation.

b) Method isOperator, which determines whether c is an operator.

c) Method precedence, which determines whether the precedence of operator1 (from the infix expression) is less than, equal to or greater than that of operator2 (from the stack). The method returns true if operator1 has lower precedence than operator2. Other- wise, false is returned.

d) Method peek (this should be added to the stack class), which returns the top value of the stack without popping the stack.

(Postfix Evaluator)

Write class PostfixEvaluator that evaluates a postfix expression such as 62+5*84/-

The program should read a postfix expression consisting of digits and operators into a StringBuffer. Using the stack methods, the program should scan the expression and evaluate it (Check the validity of the expression). The algorithm is as follows:

a) Append a right parenthesis ')' to the end of the postfix expression. When the right-parenthesis character is encountered, no further processing is necessary.

b) Until the right parenthesis is encountered, read the expression from left to right. If the current character is a digit, do the following: Push its integer value onto the stack (the integer value of a digit character is its value in the Unicode character set minus the value of '0' in Unicode). Otherwise, if the current character is an operator: Pop the two top elements of the stack into variables x and y. Calculate y operator x. Push the result of the calculation onto the stack.

c) When the right parenthesis is encountered in the expression, pop the top value of the stack. This is the result of the postfix expression.

[Note: In b) above (based on the sample expression at the beginning of this exercise), if the operator is '/', the top of the stack is 4 and the next element in the stack is 40, then pop 4 into x, pop 40 into y, evaluate 40 / 4 and push the result, 10, back on the stack. This note also applies to operator '-'.] The arithmetic operations allowed in an expression are: + (addition), - (subtraction), * (multiplication), / (division), ^ (exponentiation) and % (remainder).

The stack should be maintained with a stack classes. You may want to provide the following methods:

a) Method evaluatePostfixExpression, which evaluates the postfix expression.

b) Method calculate, which evaluates the expression op1 operator op2.

The algorithms described above were explained using expressions with one digit. Your code should process integer operands larger than 9

If the expression entered is invalid, make sure that you handle it using an exception, by outputting an error message and allow the user to enter a valid expression or to return back to the main menu to choose to perform a task again, or exit.

Your code should be well documented (using java docs). Generate the api document and submit with your code, UML diagram, and multiple sample runs