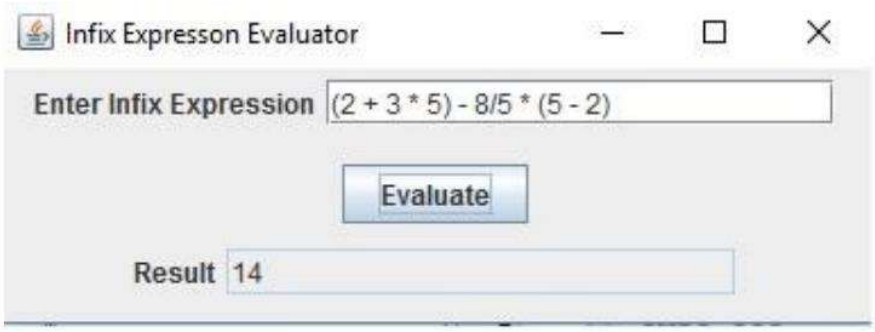
UMUC

CMSC 350 Project 1

1. Specification

Design, write and test a program that evaluates **infix expressions** of unsigned integers using two stacks. The main class will be **P1GUI.** It should create a Swing based GUI that allows the user to input an infix expression and displays the result. The GUI must be generated by code that you write. You may not use a drag-and-drop GUI generator. The GUI should look as follows:



Another class, **InfixEval,** should contain the code to perform the infix expression evaluation. For infix expression evaluation, two stacks will be used: a stack of operands and a stack of operators. The pseudocode for performing that evaluation is shown below:

```
tokenize the string containing the expression
while there are more tokens
  get the next token
  if it is an operand
    push it onto the operand stack
  else if it is a left parenthesis
    push it onto the operator stack
  else if it is a right parenthesis
    while top of the operator stack is not a left parenthesis
      pop two operands and an operator
      perform the calculation
      push the result onto the operand stack
    pop top of the operator stack and ignore it (i.e. it is a left parenthesis)
  else if it is an operator
    while the operator stack is not empty and
           the operator at the top of the stack has higher
           or the same precedence than the current operator
      pop two operands and perform the calculation
      push the result onto the operand stack
    push the current operator on the operators stack
while the operator stack is not empty
  pop two operands and an operator
  perform the calculation
  push the result onto the operand stack
the final result is at the top of the operand stack
```

Be sure to add any additional methods needed to eliminate any duplication of code.

Your program is only expected to perform correctly on syntactically correct infix expressions that contain integer operands and the four arithmetic operators + - * /. It should not, however, require spaces between tokens. The usual precedence rules apply. The division performed should be integer division. A check should be made for division by zero. Should the expression contain division by zero, a checked custom exception **DivideByZero** should be thrown by the method that performs the evaluation and caught in the main class, where a **JOptionPane** window should be displayed containing an error message.

Your program should compile without errors.

The Google recommended Java style guide (https://google.github.io/styleguide/javaguide.html) should be used to format and document your code. Specifically, the following style guide attributes should be addressed: header comments include filename, author, date and brief purpose of the program; in-line comments used to describe major functionality of the code; the meaning and the role of variables and constants are indicated as code comments; meaningful variable names and prompts applied; class names are written in UpperCamelCase; variable names are written in lowerCamelCase; constant names are in written in All Capitals; braces use K&R style.

In addition, the following design constraints should be followed: declare all instance variables private; avoid the duplication of code.

2. Submission requirements

Submit the following to the Project 1 assignment area no later than the due date listed in your LEO classroom.

1. All **.java** source files (**no other file types should be submitted**) and the **output file** generated by the program. The source code should use Java code conventions and appropriate code layout (white space management and indents) and comments. All submitted files may be included in a .zip file.

2. The solution description document **P1SolutionDescription** (.pdf or .doc / .docx) containing the following:
(1) Assumptions, main design and error handling;
(2) A UML class diagram that includes all classes you wrote. Do not include predefined classes. You need only include the class name for each individual class, not the variables or methods;
(3) A table of test cases including the test cases that you have created to test the program. The table should have 5 columns indicating (i) what aspect is tested, (ii) the input values, (iii) the expected output, (iv) the actual output and (v) if the test case passed or failed. Each test case will be defined in a table row.
(4) Relevant screenshots of program execution;
(5) Lessons learned from the project;

Grading Rubric:

Criteria	Meets	Does Not Meet
Design	**5 points**	**0 points**
	GUI is hand coded and matches required design	GUI is generated by a GUI generator or does not match required design
	Supplied algorithm is used	Supplied algorithm is not used
	Contains separate class for expression evaluation	Does not contain separate class for expression evaluation
	All instance variables are private	Instance variables are not private
	Contains checked exception class	Does not contain checked exception class
	Uses good object-oriented design practice regarding code efficiency, encapsulation and information hiding, class and code reuse, high cohesion of classes, avoiding code duplication.	Does not use good object-oriented design practice resulting in code efficiency, encapsulation and information hiding, class and code reuse, high cohesion of classes, avoiding code duplication.
	10 points	**0 points**
	Produces correct value for all operators	Does not produce correct value for some operators
	Correctly parses expressions without	Does not correctly parse expressions

Functionality	space delimiters	without space delimiters
	Correctly implements precedence	Does not correctly implement precedence
	Correctly evaluates parenthesized expressions	Does not correctly evaluate parenthesized expressions
	Detects division by zero	Does not detect division by zero
Test Cases	**5 points**	**0 points**
	Test cases table is defined and included in the P1SolutionDescription document	Test cases table is not defined and included in the P1SolutionDescription document
	All operators included in test cases	Some operators not included in test cases
	Test cases include expressions without spaces	Test cases don't include expressions without spaces
	Test cases include cases to test precedence	Test cases do not include cases to test precedence
	Test cases include cases with parentheses	Test cases do not include cases with parentheses
	Test cases include a case to test division by zero	Test cases do not include a case to test division by zero
Documentation	**5 points**	**0 points**
	Solution description document P1SolutionDescription includes all the required sections (appropriate titled).	No solution description document is included
	Source code follows Google recommendation Java style	Source code does not follow Google recommendation Java style
	Comment blocks with class description included with each class	Comment blocks with class description not included with each class
	Source code is commented and indented	Source code is not commented and indented
Overall Score	**Meets**	**Does not meet**
	16 or more	15 or less