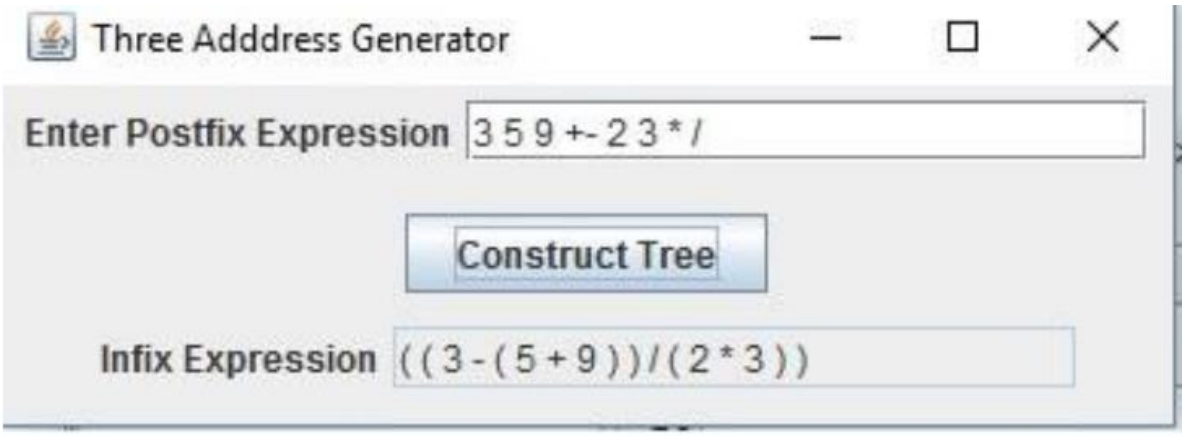
UMUC
CMSC 350 Project 2

1. Specification

Design, write and test a program that accepts an arithmetic expression of unsigned integers in postfix notation in which the tokens are separated by spaces and builds and processes the arithmetic expression tree that represents that expression.

The main class will be **P2GUI**. It should create a Swing based GUI shown below:



The GUI must be generated by code that you write. You may not use a drag-and-drop GUI generator.

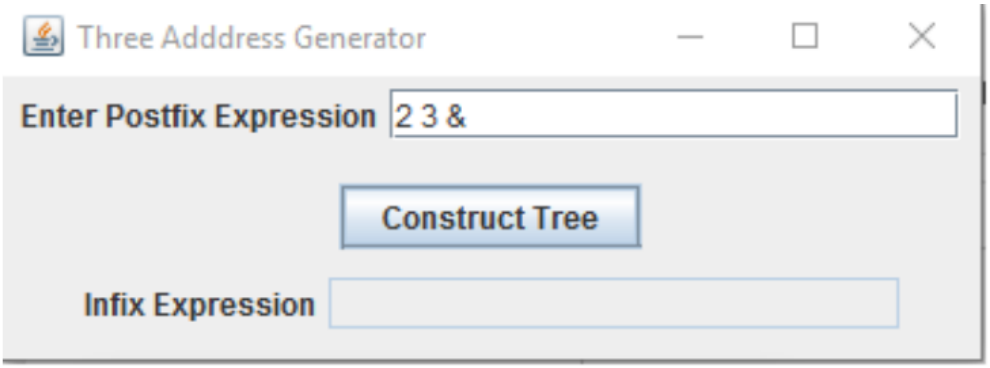
Pressing the *Construct Tree* button should cause the following: (i) the arithmetic expression tree that represents the entered postfix expression will be constructed, (ii) using that tree, the corresponding fully parenthesized infix expression should be generated and displayed in the GUI and finally (iii) a file should be generated that contains the 3-address format instructions corresponding to the arithmetic expression. These topics, including relevant classes for implementing and processing an expression tree, are discussed in the week 4 reading "Binary Trees, Expression Trees, BST (AVL) trees, and B-Trees" section II "Expression Trees".

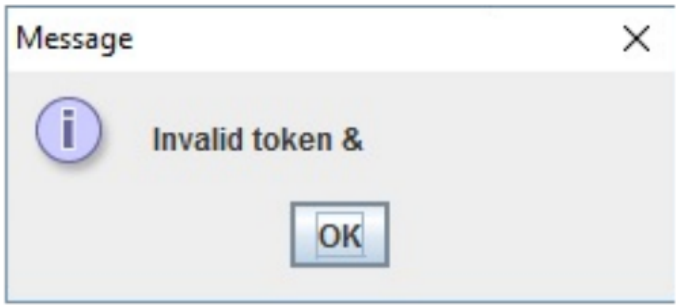
The above example should produce the following output file containing the 3-address instructions:

```
Add R0 5 9
Sub R1 3 R0
Mul R2 2 3
Div R3 R1 R2
```

It is not necessary to reuse registers within an expression as shown in the above mentioned reading and you can assume there are as many available registers as needed. Each new expression should, however, begin using registers starting at R0.

You may assume that the expression is syntactically correct with regard to the order of operators and operands, but you should check for invalid tokens, such as characters that are not valid operators or operands such as 2a, which are not valid integers. If an invalid token is detected, a checked custom exception `InvalidTokenException` should be thrown and caught by the main class and an appropriate error message should be displayed in an `JOptionPane`. Below is an example:





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 2 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 **P2SolutionDescription** (.pdf or .doc / .docx) containing the following:
(1) Assumptions, main design decisions, 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
	Other classes are used to support the implementation of the arithmetic expression tree	Does not use other classes to support the implementation of the arithmetic expression tree
	All instance data is private	Some instance data is not private
	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 regarding code efficiency, encapsulation and information hiding, class and code reuse, high cohesion of classes, avoiding code duplication.
Functionality	**10 points**	**0 points**
	Produces correct fully parenthesized infix expressions for all input	Does not produce correct fully parenthesized infix expressions for some input
	Produces correct three address file for all input	Does not produce correct three address file for some input
	Correctly parses expressions with space delimiters	Does not correctly parse expressions with space delimiters
	Registers restart at R0 on each new expression	Registers do not restart at R0 on each new expression
	Detects and handles invalid tokens	Does not detect and handle invalid tokens
Test Cases	**5 points**	**0 points**
	Test cases table is defined and included in the P2SolutionDescription document	Test cases table is not defined and included in the P2SolutionDescription document
	All operators included in test cases	Some operators not included in test cases
	Test cases include expressions with and without spaces	Test cases don't include expressions with and without spaces
	Test cases include a case to test invalid token beginning with a digit	Test cases do not include a case to test invalid token beginning with a digit
	Test cases include a case to test invalid operators	Test cases do not include a case to test invalid operators
Documentation	**5 points**	**0 points**
	Solution description document P2SolutionDescription includes project specific details for 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	**0-15**