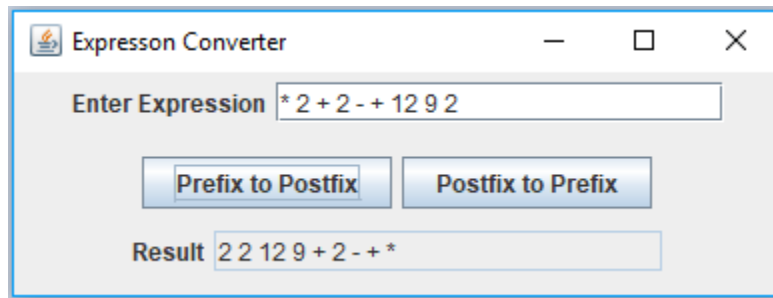


CMSC 350 Project 1

The first programming project involves writing a program that converts prefix expressions to postfix and postfix expressions to prefix. Customary infix expression place the operator between the two operands. In a prefix expression, the operator comes before the two operands. In a postfix expression it comes after them. One benefit of prefix and postfix expressions is that they never require parentheses nor rules regarding precedence or associativity.

The program for this project should consist of three classes. The main class should create a GUI that allows the user to input an expression in either prefix or postfix and convert it to postfix or prefix, respectively. The GUI should look as follows:



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

The second class should contain the code to perform the conversions. The pseudocode for prefix to postfix, which requires two stacks, is shown below:

```
tokenize the string containing the prefix expression
while there are more tokens
    push the token onto a reversal stack if it is not a space
while the reversal stack is not empty
    pop the next token from the reversal stack
    if it is an operand
        push it onto the operand stack
    else it is an operator
        pop two operands off of the operand stack
        create a string with the two operands followed
            the operator
        push that string onto the operand stack
pop the postfix expression off the stack
```

The pseudocode for the postfix to prefix conversion, which requires only one stack, is shown below:

```
tokenize the string containing the postfix expression
while there are more tokens
    get the next token
    if it is a space
        skip it
    else if it is an operand
        push it onto the operand stack
    else it is an operator
        pop two operands off of the operand stack
        create a string with the operator followed by
            two operands
```

```

        push that string onto the operand stack
    pop the prefix expression off the stack

```

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

The input expressions should not be required to contain spaces between tokens unless they are consecutive operands. The output expressions should always have a space between all symbols. A checked exception `SyntaxError` should be thrown by the methods that perform the conversions if an empty stack is ever popped or the stack is not empty once the conversion is complete. That exception should be caught in the main class, where a `JOptionPane` window should be displayed containing an error message.

You are to submit two files.

1. The first is a `.zip` file that contains all the source code for the project. The `.zip` file should contain only source code and nothing else, which means only the `.java` files. If you elect to use a package the `.java` files should be in a folder whose name is the package name. Every outer class should be in a separate `.java` file with the same name as the class name. Each file should include a comment block at the top containing your name, the project name, the date, and a short description of the class contained in that file.
2. The second is a Word document (PDF or RTF is also acceptable) that contains the documentation for the project, which should include the following:
 - a. 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
 - b. A test plan that includes test cases that you have created indicating what aspects of the program each one is testing
 - c. A short paragraph on lessons learned from the project

Grading Rubric

| Criteria | Meets | Does Not Meet |
|----------------------|---|---|
| | 20 points | 0 points |
| Design | GUI is hand coded and matches required design (4) | GUI is generated by a GUI generator or does not match required design (0) |
| | Supplied algorithm is used (5) | Supplied algorithm is not used (0) |
| | Code duplication is eliminated (3) | Contains duplicated code (0) |
| | Contains separate class for expression conversion (4) | Does not contain separate class for expression conversion (0) |
| | Contains checked exception class (4) | Does not contain checked exception class (0) |
| | | |
| Functionality | 60 points | 0 points |
| | | |
| | Correctly converts prefix to postfix (20) | Does not correctly convert prefix to postfix (0) |
| | Correctly converts postfix to prefix (20) | Does not correctly convert postfix to prefix (0) |

| | | |
|----------------------|---|---|
| | Correctly parses expressions without space delimiters (10) | Does not correctly parse expressions without space delimiters (0) |
| | SyntaxError exception is thrown on expression with invalid syntax (5) | SyntaxError exception is not thrown on expression with invalid syntax (0) |
| | SyntaxError is caught and a JOptionPane message is displayed (5) | SyntaxError is not caught or a JOptionPane message is not displayed (0) |
| Test Plan | 10 points | 0 points |
| | | |
| | All operators included in test cases (3) | Some operators not included in test cases (0) |
| | Test cases include expressions without spaces (3) | Test cases don't include expressions without spaces (0) |
| | Test cases include cases that pop an empty stack (2) | Test cases do not include cases that pop an empty stack (0) |
| | Test cases include cases which result in a non-empty stack (2) | Test cases do not include cases which result in a non-empty stack (0) |
| Documentation | 10 points | 0 points |
| | | |
| | Correct UML diagram included (3) | Correct UML diagram not included (0) |
| | Lessons learned included (4) | Lessons learned not included (0) |
| | Comment blocks included with each Java file (3) | Comment blocks not included with each Java file(0) |