Infix Expression Parser

Project 2A

16 February 2021 - Intro to Data Structures with Java

Github Link: https://github.com/rdmallinson7/Project_2_DS.git

ISSUED BY

Dayu Wang - St. Charles Community College

TEAM SIX

Rachael Mallinson

Sofia Nikas

Aleksis Martin

Design Explanation

Data Structures utilized: Stacks

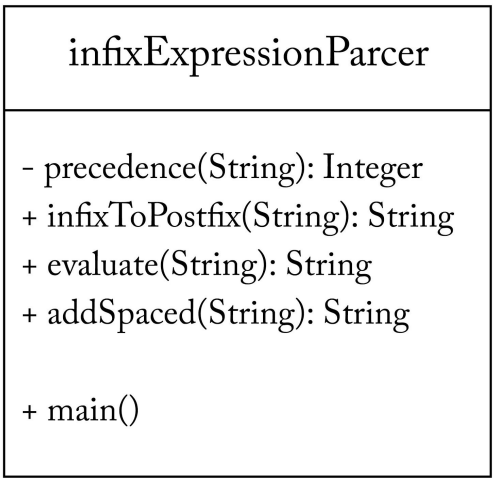
For our Infix Expression Parser, we decided to use three steps. First, we take in the input infix expression as a string labeled "infixExp." During this process, we take out all of the spaces because we aren't sure what kind of input we will receive. Then, we use another method (addSpaces) to add spaces into the "infixExp," separating each of our operands and operators.

Then we convert our infix expression (infixExp) to a postfix expression. In this method, we use a stack to store our operands and operators, until it's time to output them to the postfix expression in the correct order. In order to do this accurately, we set precedences for each of the operators. This helps us to tell the program the correct order that the operators or operands should be placed.

Lastly, we evaluate the postfix expression based on the order of operations made clear by the postfix expression and output it to the standard console.

UML Class Diagram

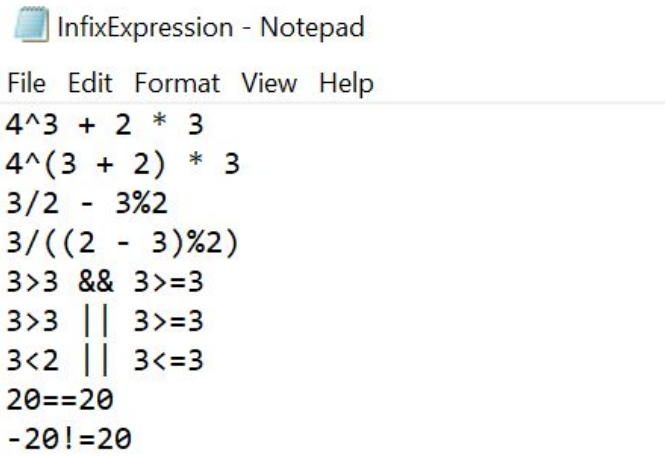
The following is a UML Diagram that outlines our Infix Expression Parser:



Test Cases

Test Case 1:

The first test case tests each operator and expressions with parentheses. All supported operators are represented in this test case. Some expressions are repeated with parentheses changing the order of operations.



Expected result:

1. 70
2. 3072
3. 0
4. -3
5. 0 (False)
6. 1 (True)
7. 1 (True)
8. 1 (True)
9. 1 (True)

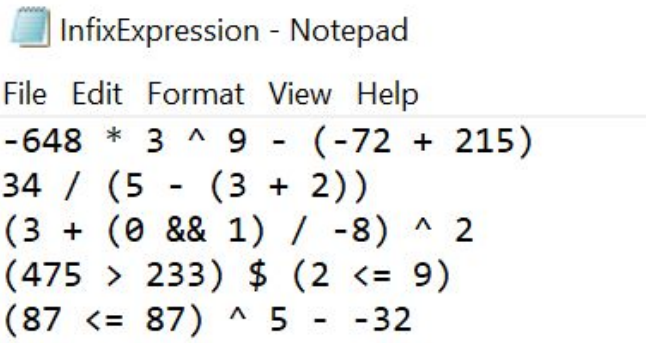
Actual result:

```
Infix Expression Parcer

Expression given: 4^3 + 2 * 3
Evaluated Expression: 70

Expression given: 4^(3 + 2) * 3
Evaluated Expression: 3072

Expression given: 3/2 - 3%2
Evaluated Expression: 0

Expression given: 3/((2 - 3)%2)
Evaluated Expression: -3

Expression given: 3>3 && 3>=3
Evaluated Expression: 0

Expression given: 3>3 || 3>=3
Evaluated Expression: 1

Expression given: 3<2 || 3<=3
Evaluated Expression: 1

Expression given: 20==20
Evaluated Expression: 1

Expression given: -20!=20
Evaluated Expression: 1
```

Conclusion: Expected result matches actual result.

Test Case 2:

The second test case tests larger numbers, negative numbers, nested parentheses, divide-by-zero error handling, and unsupported operator error handling.



Expected result:

1. -12754727
2. Divide-by-zero message, no program crash
3. 9
4. Unsupported operator message, no program crash
5. 33

Actual result:

```
Infix Expression Parcer

Expression given: -648 * 3 ^ 9 - (-72 + 215)
Evaluated Expression: -12754727

Expression given: 34 / (5 - (3 + 2))
Evaluated Expression: Divide-by-zero error, could not evaluate expression.

Expression given: (3 + (0 && 1) / -8) ^ 2
Evaluated Expression: 9

Expression given: (475 > 233) $ (2 <= 9)
Evaluated Expression: $ operator not supported.

Expression given: (87 <= 87) ^ 5 - -32
Evaluated Expression: 33
```

Conclusion: Expected result matches actual result.

Project Contributions

The contributions of each Team Member are as follows:

Rachael Mallinson

1. Additions to Precedence method
2. UML Diagram
3. Additions to Evaluate method

Sofia Nikas

1. AddSpaces method
2. Additions to Evaluate method
4. Test Cases

Aleksis Martin

1. Additions to Evaluate method
2. Ideas for improvement

Ideas for Future Improvement

The following are ideas for improvements we predict we could make to our Infix Expression Parser given an adequate amount of time.

1.) To avoid a program crash when the input file contains white space (tab, or empty lines) we wanted to implement a check of the inputFile in the main program.

```
// Eliminate any blank lines in the inputFile
while (infixExp.isBlank()) { infixExp = scanner.nextLine(); }
```

Aleksis wrote the above while loop using 2020-12 eclipse and the program skipped empty lines while reading in the inputFile. Racheal noticed the isBlank() method isn't supported on all types of Java so we ended up taking this out so it would work on all of our computers.

2.) Tweak the string outputs of the standard console so the input expressions and their evaluated answers in the standard console line up (match whitespace).

3.) Make improvements to catch any user-input errors such as non-closed parenthesis, two unsupported operands in a row (system crashes), or any situation which results in the stack becoming prematurely empty.

4.) Figure out why Eclipse says our scanner in the evaluate() method line 86 (we close it outside the while loop at line 141) isn't closed.

This concludes our Project 2A Report. Thank you for this opportunity and for your attention.