Expression Tree Converter

Your assignment is build a data structure that represents a mathematical equation, and you'll use the Calculator that you built in Homework #10. You already have a method that converts user input into a postfix expression. This time, you will take the postfix expression and convert it into an expression tree. The exact implementation is left up to you, though you must read input from the console.

After you create the tree, you should do preorder, inorder, and postorder traversals in order to print prefix, infix (with appropriate parenthesis), and postfix expressions representing the input. Your application should be able to handle multiple inputs before exiting (ie via Ctrl-C); after printing the results, it should ask for a subsequent infix expression on the console.

Your console should do the following when the user types in an expression: convert to postfix, convert to expression tree, print traversals.

Expression Trees

An expression tree is a data structure used to represent an arithmetic expression. There are two types of nodes in an expression tree: operators and numbers. Number nodes are leaves and have no children. Operator nodes have two children that are also expression trees (each child can be an operator or a number). For example, the expression (4 - 2) will be represented by an expression tree with a root operator node. It'll store the fact that it's the subtraction operator, and it will have two children, each number nodes 4 and 2.

A more complicated expression will have more nodes associated with it. Consider the expression (9+6*2/(8-3)). This expression is the sum of two other expressions, so the root will be an operator node, with operator "+", and two child expressions. The left child will be the leaf node with number "9", but the right child is going to be an operator node with operator "/". It's children are also operator nodes: the left child is 6*2 and the right child is 8-3.

Tips for Getting Organized

The first step you should take is to figure out how to represent an expression tree in Java. Create a class that represents a node in an expression tree. Try creating some expression trees manually, and make sure that you can represent the full variety of

arithmetic expressions that can be handled by your original calculator.

Once you've done that, you'll need to build four methods in your expression tree class:

Required Methods
Note: You may use whatever method signatures you like.

convert(), which takes a postfix expression and creates an expression tree out of it
prefix(), a preorder traversal of the tree
infix(), an inorder traversal of the tree
postfix(), a postorder traversal of the tree

Once these methods are built, they will need to be called in your console so the results get printed on each input.

The Algorithm

The algorithm for converting a postfix expression into an expression tree is similar to some of the other conversion algorithms that you've done. Here are the steps:

1. Create an empty stack. Each element in the stack is going to be an expression tree, so set the generic type accordingly.
2. Loop through each token in the postfix expression created by your converter.
    a. If the token is a number, create a new expression tree node that represents just that number and push it onto the stack
    b. If the token is an operator, create a new operator expression. Pop two expressions off the top of the stack, and set them as children of this new node. Then push the node onto the stack
3. Once all the tokens in the postfix expression have been processed, the stack will contain only one expression tree. Pop it out, and return it.

Example Output

```
Type your expression:
3*4+5
Prefix: +*345
Infix: ((3*4)+5)
Postfix: 34*5+
```

Other Notes

Remember: postorder traversal of a tree prints the children of the node first, and then prints the data inside the node. Preorder traversal prints the node data first. When printing out these traversals, no parenthesis are needed.

Inorder traversal will print the left child, followed by the operator, followed by the right child. The result should look very similar to the original expression that was typed in by the user. This needs to be surrounded by parenthesis in order to specify the correct order of operations.

The recursive solution to coding the traversals is probably the simplest.

Submission

You will submit an expression tree class with the detailed methods. You can optionally separate the input console into a different class and submit that as well.

Grading Rubric

Criteria

Compiles successfully (1 point)

Project successfully creates and traverses an expression tree from an infix expression (8 points)

Good style demonstrated in the code; sensible formatting (1 point)