Program 11 – Binary Expression Trees

100 points

IMPORTANT
You must write your code while carefully following the "IT 179 Program Grading Guidelines" described in the file **posted with Program 1**.

Set-Up
Create a new Java project named: **`P11`**. Next, inside the created Java project **`P11`**, create a Java package.

Objective
Practice with binary tree implementation and using stacks.

The Problem

You are to develop a Java program that requests and reads an arithmetic expression in postfix notation and creates a binary expression tree from that expression. The program will then print the expression out in prefix form and in properly parenthesized infix form. It will also evaluate the expression and print the result. You may assume spaces between elements of the expression.

To build the tree, you will need a Stack of binary tree nodes. Feel free to use the Java Stack class.

Sample input:

```
10 6 * 4 3 + -
```

Sample output:

```
Prefix: - * 10 6 + 4 3
Infix: ((10 * 6) – (4 + 3))
Result: 53
```

Required Classes

The `BinTreeNode` Class

You will have a `BinTreeNode` class. You will need to use this in a Stack, so it will be simpler to have it as a separate public class. It will have the following instance variables which may be public. Note that we could make this smaller, we're going with the simple approach.

- a boolean `isOperator`
- a char `symbol`
- an int `value`
- a `BinTreeNode left`
- a `BinTreeNode right`

You should also have appropriate constructors.

The `ExpressionTree` Class

This class will hold your binary tree. It will have a single private instance variable (unless you attempt the extra credit):

- a `root` of type `BinTreeNode`

It must have at least the following public methods:

- a constructor that takes a String that represents an expression in postfix notation
- `getPrefixNotation`, which will take no parameters and will return a String that contains the prefix version of the expression
- `getInfixNotation`, which will take no parameters and will return a String that contains the infix version of the expression
- `getValue`, which will take no parameters and will return an integer that is the result of evaluating the expression tree.
- You may find it useful to also provide a `getPostfixNotation`, but it is not required.

You should add additional public methods only if you attempt the extra credit.

The class must also have at least private recursive methods that do the work for `getPrefixNotation`, `getInfixNotation`, and `getValue`. You may add additional private methods that will help break up the work of the class.

You need a program class that will have **main** and handle interaction with your user.

Extra credit (up to 20 points):

You may earn up to 20 points of extra credit by handling expressions with variables. Read in the expression and store it in the tree. Write the prefix and infix. Ask the user for values for the

variables (only one time for each variable, even if a variable appears more than once in the expression). Evaluate the expression and display the result.

Submission

Zip your .java files and submit the P11.zip file to the **Program 11** assignment on ReggieNet.

Grading will be in accordance with the Program Grading Criteria provided on the Important Resources page on ReggieNet.