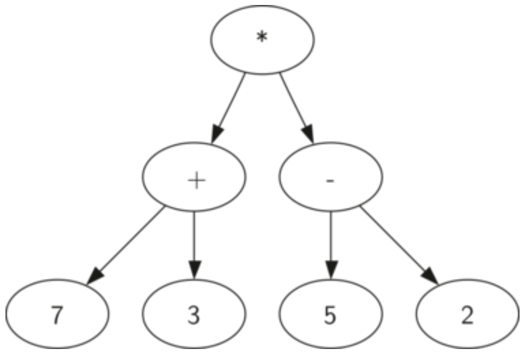
EvaluateTree.java

Objective: To create your own algorithm and implement the code for evaluating the result of a binary expression tree.

Background:
Binary expression trees hold algebraic expressions that can be easily evaluated for a result. For example, the following tree is created from the expression “(7 + 3) * (5 - 2)”.



Notice there are no parentheses in the tree, just operands and operators. The tree structure holds the priority of operations. The highest priority operations are at the bottom of the tree, while the lowest priority operation is at the root. By traversing the tree in a specific way, the result of this expression can be calculated.

Assignment:
Download the file **EvaluteTree.zip** from Mr Greenstein’s web site. Unzip the file and it will create the directory “EvaluationTree” with two files, **TreeNode.java** and **EvaluateTree.java**. **TreeNode.java** is the same file used in the **BinaryTree** project, and should not be changed.

You will create the **double evaluateTree()** method in the **EvaluateTree.java** file. It will return the result of evaluating the tree. A test main program is provided with two expression trees.

In this project, there are a few constraints to limit the scope.

1. The operations to be performed are addition (+), subtraction (-), multiplication (*), division (/), modulus (%), and exponent (^).
2. There will be <u>no unary operators</u> in the tree, only binary operators as shown in 1.
3. Operands will only be numbers. There will be no variables in the tree.

Important! It is wise to test your program on more than the two expressions provided. Create and evaluate a few more expression trees to insure your program works.

A sample run using the program in **EvaluateTree**:

```
% java EvaluateTree
```

```
Expression Tree
------------------------

Expression: 2 ^ 3 ^ 2

      2
   ^
      3
^
   2

Answer: 512.0

Expression: ( ( 2 + 3 ) * 5 / 6 ) + 7 * 8

      8
   *
      7
+
      6
   /
         5
      *
            3
         +
            2

Answer: 60.166666666666664

Expression: 1.2 * ( ( 3.4 - 2.1 ) / 1.3 * 8. )

      8.
   *
         1.3
      /
            2.1
         -
            3.4
*
   1.2

Answer: 9.599999999999998
```