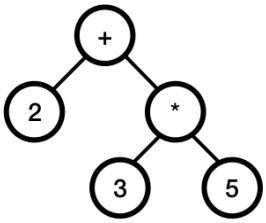
ExpressionTree.java

Objective: To build and evaluate an expression tree.

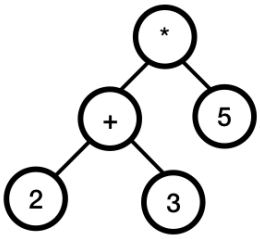
Background:
The prerequisite for this project is to complete both the **EvaluateTree** project and the **ToPostfix** project. **ExpressionTree** needs working versions of methods from those projects.

Earlier this year we made a simple calculator using stacks (SimpleCalc). The user would enter the expression as a **String** and the program would evaluate it using a stack.

In this project, binary trees will be used to store and evaluate an expression. The advantage of using a binary tree is the structure of the tree determines the precedence of operations. For example, the expression "2 + 3 * 5" would be stored in the tree as such:



The expression would be evaluated using postorder and a stack. If the addition happens first, like "(2 + 3) * 5", then the tree would look like this:



Notice the tree stores no parentheses. The tree's structure and postorder allow for proper evaluation. Also, the only operators in this project will be the binary operators "+", "-", "*", "/", "%", and "^". No unary operators will be used in the expressions. This will insure that every parent node in the tree has exactly two children.

Assignment:
Download the file **ExpressionTree.zip** from Mr Greenstein's web site. Unzip the file and it will create the directory "**ExpressionTree**" containing the file **ExpressionTree.java**.

1. Prior to this you completed the **ToPostfix** project. Move the following files from the **ToPostfix** directory to the **ExpressionTree** directory: **ArrayStack.java**, **ExprUtils.java**, **Stack.java**, and **TreeNode.java**. Using Geany, copy your **Prompt.java** file into the directory too. You will need to prompt the user for input.

2. Copy the following expression tree methods from **BinaryTree.java** into **ExpressionTree.java**. As you copy them, test each method and be sure they work:
   void printInorder(), **void printPreorder()**, and **void printPostorder()**.

3. Copy method **double evaluateTree()** from **EvaluateTree.java** into **ExpressionTree.java**.

4. Write the **void treeMakerInterface()** method in **ExpressionTree.java**. It should implement the features listed in the **printMenu()** method, specifically
   - i - input new expression
   - pre - print the tree in prefix notation
   - in - print the tree in infix notation
   - post - print the tree in postfix notation

- e - evaluate the expression
- p - print the expression tree
- q - quit

5. Write the **TreeNode<String> buildTree()** method in **ExpressionTree.java**. **buildTree()** reads an array (or **ArrayList**) of tokens in postfix order and builds the tree. Work out an algorithm before you write the code. (Hint: Use a TreeNode stack.)

A sample run (user input in **bold**):

```
% java ExpressionTree

Welcome to ExpressionTree!!!

Current expression:

Choose:
  (i) input new expression
  (pre) print prefix notation
  (in) print infix notation
  (post) print postfix notation
  (e) evaluate expression
  (p) print tree
  (q) quit
 -> i
expression -> 2 * 3 + 4

Current expression: 2 * 3 + 4

Choose:
  (i) input new expression
  (pre) print prefix notation
  (in) print infix notation
  (post) print postfix notation
  (e) evaluate expression
  (p) print tree
  (q) quit
 -> p

Print tree

   4
+
      3
   *
      2

Current expression: 2 * 3 + 4

Choose:
  (i) input new expression
  (pre) print prefix notation
  (in) print infix notation
  (post) print postfix notation
  (e) evaluate expression
  (p) print tree
  (q) quit
 -> pre

Prefix order
+ * 2 3 4


Current expression: 2 * 3 + 4

Choose:
  (i) input new expression
```

```
  (pre) print prefix notation
  (in) print infix notation
  (post) print postfix notation
  (e) evaluate expression
  (p) print tree
  (q) quit
 -> in

Infix order
2 * 3 + 4


Current expression: 2 * 3 + 4

Choose:
  (i) input new expression
  (pre) print prefix notation
  (in) print infix notation
  (post) print postfix notation
  (e) evaluate expression
  (p) print tree
  (q) quit
 -> post

Postfix order
2 3 * 4 +


Current expression: 2 * 3 + 4

Choose:
  (i) input new expression
  (pre) print prefix notation
  (in) print infix notation
  (post) print postfix notation
  (e) evaluate expression
  (p) print tree
  (q) quit
 -> e

Answer: 10.0

Current expression: 2 * 3 + 4

Choose:
  (i) input new expression
  (pre) print prefix notation
  (in) print infix notation
  (post) print postfix notation
  (e) evaluate expression
  (p) print tree
  (q) quit
 -> i
expression -> 96 + 2.8 * 61.1 - 45.2

Current expression: 96 + 2.8 * 61.1 - 45.2

Choose:
  (i) input new expression
  (pre) print prefix notation
  (in) print infix notation
  (post) print postfix notation
  (e) evaluate expression
  (p) print tree
  (q) quit
 -> p

Print tree

   45.2
-
```

```
         61.1
      *
         2.8
   +
      96

Current expression: 96 + 2.8 * 61.1 - 45.2

Choose:
  (i) input new expression
  (pre) print prefix notation
  (in) print infix notation
  (post) print postfix notation
  (e) evaluate expression
  (p) print tree
  (q) quit
 -> e

Answer: 221.88

Current expression: 96 + 2.8 * 61.1 - 45.2

Choose:
  (i) input new expression
  (pre) print prefix notation
  (in) print infix notation
  (post) print postfix notation
  (e) evaluate expression
  (p) print tree
  (q) quit
 -> i
expression -> 8 / 4 + (2.1 * (5 + 3.3) % (6 - 1))

Current expression: 8 / 4 + (2.1 * (5 + 3.3) % (6 - 1))

Choose:
  (i) input new expression
  (pre) print prefix notation
  (in) print infix notation
  (post) print postfix notation
  (e) evaluate expression
  (p) print tree
  (q) quit
 -> p

Print tree

         1
      -
         6
   %
            3.3
         +
            5
      *
         2.1
+
      4
   /
      8

Current expression: 8 / 4 + (2.1 * (5 + 3.3) % (6 - 1))

Choose:
  (i) input new expression
  (pre) print prefix notation
  (in) print infix notation
  (post) print postfix notation
  (e) evaluate expression
  (p) print tree
  (q) quit
```

```
 -> e

Answer: 4.430000000000003

Current expression: 8 / 4 + (2.1 * (5 + 3.3) % (6 - 1))

Choose:
  (i) input new expression
  (pre) print prefix notation
  (in) print infix notation
  (post) print postfix notation
  (e) evaluate expression
  (p) print tree
  (q) quit
 -> q

Thanks for using ExpressionTree! Goodbye.
```