CIS351-Expression Tree

Submission Instructions

1. Submit **OperatorNode.java** through in Blackboard.
2. DO NOT forget to mention your full name in the Java class documentation.
3. submit a filled in **cover sheet** in Blackboard.

Introduction

In this Lab you will complete the implementation of a binary tree class that represents mathematical expressions. The class will provide functionality for evaluating expressions and formatting them in prefix, postfix or infix notation.

Download Materials

For this lab, you need to download the following starter code from Blackboard:

- Operator.java - Enumerated type representing the set of operators.
- ExpressionNode.java - Abstract superclass for expression nodes.
- OperandNode.java - Class representing operands (leaves) in an expression tree.
- ExpressionDriver.java - Simple test driver for executing expression tree methods.
- PrefixParser.java - Class for converting postfix expressions into expression trees.

Part 1 - Getting Started

For today's lab, there is two task:

1. Read the downloaded java classes to understand their functionality
2. Complete the OperatorNode.java file. This class representing operators (internal nodes) in an expression tree.

Part 2 - Instructions

Look at the following sample implementation of apre-order traversal:

```
static <E> void preorder(BinNode<E> rt) {
  if (rt == null) return; // Empty subtree - do nothing
  visit(rt);              // Process root node
  preorder(rt.left());    // Process all nodes in left
  preorder(rt.right());   // Process all nodes in right
}
```

In this code, the traversal has been implemented as a static method in some separate class that is passed a reference to a root node. As an alternative, it is possible to implement a tree as a recursive data structure without a separate class to handle the traversals. In this approach the node *is* the tree, and all of the functionality is implemented through methods of the node class. Our `ExpressionNode` class will be organized in this way. Under this approach, a preorder traversal might look like the following:

```
private void preorder() {
  visit();                // Process root node
  if (!isLeaf()) {
    left().preorder();    // Process all nodes in left
    right().preorder();   // Process all nodes in right
  }
}
```

It may seem odd to see a recursive method with no (apparent) arguments. In this case the argument is implicit. Since the recursive calls are executed on different `BinNode` objects, it is the object `this` that changes from one call to the next.

Note that the method above will only work for full binary trees: it assumes that every node is either a leaf, or contains two valid children. Our expression trees will necessarily be full because every operation must have exactly two operands. The methods for our `ExpressionNode` classes will be even simpler than the traversal above. Since leaves are stored in a a different node type, there is no need for an explicit `isLeaf` check .

Grading Criteria

Total points: 10 points

4 incomplete methods, each worth 2.5 pt

Farzana Rahman / frahman@syr.edu