AP Computer Science with Data Structures

Lab: Trees part 2

Collaboration rules apply as per the course policies.

Exercise: The shape of things to come

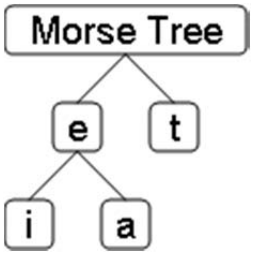
Given a binary tree `t`, a copy of `t` is a new tree with all new `TreeNode` objects pointing to the same values as `t` (in the same order, shape, etc.) Two trees have the same shape if they have `TreeNode` objects in the same locations relative to the root.

Complete and test the methods `copy` and `sameShape` in `TreeUtilities`. These methods must be implemented *correctly* to receive credit. Converting the tree to a list in order to copy it or determine its shape will not receive credit.

```
public static TreeNode copy(TreeNode t)
public static boolean sameShape(TreeNode t1, TreeNode t2)
```

Exercise: Coding Theory

Write a program to decode Morse code messages into plain text. The decoding is implemented with the help of a binary "decoding" tree. Morse code for each letter represents a path from the root of the tree to some node: a dot means "go left", while a dash means "go right". The node at the end of the path contains the symbol corresponding to the code. Try decoding the Morse code message
"`.  .-  -`" using the following decoding tree. (Data compression algorithms make use of such decoding trees.)



The following program decodes this message.

```
TreeNode decodingTree = createDecodingTree(display);
System.out.println(decodeMorse(decodingTree, ". .- -", display));
```

Although you've been given the `createDecodingTree` method, you'll need to write the `insertMorse` method it calls to add a `letter` to the appropriate position in the `decodingTree` (as determined by the letter's `code`), lighting up the `display` as it walks down the `decodingTree`.

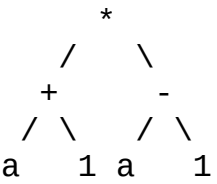
```
private static void insertMorse(TreeNode decodingTree,
          String letter, String code, TreeDisplay display)
```

You'll then need to write the method `decodeMorse`, which uses the `decodingTree` you built to decipher a `cipherText` Morse code message (such as "`. .- -`"), again lighting up the `display` as it walks down the `decodingTree`.

```
public static String decodeMorse(TreeNode decodingTree,
          String cipherText, TreeDisplay display)
```

Exercise: Expression Trees

An algebraic expression with parentheses and defined precedence of operators can be represented by a binary tree, called an **expression tree**. For example, we can represent the expression "(*a* + 1)(*a* – 1)" with the following tree.



Draw an expression tree for the expression "2 / (1/*x* + 1/*y*)". The drawing must be in your notebook in order to receive any credit for this part of the lab.

Now write the recursive method `eval`, which evaluates the expression represented by `expTree`. Assume that the nodes contain `Integer` operands and the `String` operators "`+`" and "`*`".

```
public static int eval(TreeNode expTree)
```

For example, the following code prints the value of the expression "4 * (2 + 3)".

```
tree = new TreeNode("*",
     new TreeNode(new Integer(4)),
     new TreeNode("+",
          new TreeNode(new Integer(2)),
          new TreeNode(new Integer(3))));
System.out.println(eval(tree));
```

If you finish early

Implement the method `createExpressionTree`, which takes in a string expression (consisting of integers, "`+`", "`*`", and parentheses) and returns an expression tree that represents it (again consisting of `Integer` objects and `String` operators).

```
public static TreeNode createExpressionTree(String exp)
```

For example, the following code creates an expression tree and evaluates it.

```
TreeNode tree = createExpressionTree("(5+6)*(4*(2+3))");
display.displayTree(tree);
System.out.println(eval(tree));
```

The `createExpressionTree` method is fairly difficult to implement, and there are many ways to write it. One way is to sweep across the string from left to right, keeping track of the number of parentheses still open (open parentheses that have not yet been matched by a closing parenthesis). If there are no parentheses still open when you come to an operator, you can recursively create expression trees for the substrings to the left and right of the operator, and join these trees with a new `TreeNode`. If you don't find such an operator, then you've either got an integer (which you'll parse into an `Integer` using the constructor that takes in a string) or an expression in parentheses (which you'll need to extract from the parentheses and recursively create an expression tree from it).