About Lab 5

In Lab 5 we will be working with binary trees. Remember that binary trees are either empty or else they are nodes with data and two children (leftChild and rightChild) which are themselves binary trees. One way to implement binary trees is to use null for empty trees. That requires every tree method to have special cases to cover empty trees. You get nicer code with the following:

We will have 3 classes:

- ConsTree<T> (short for Constructed Tree) has data and two kids; both children are BinaryTrees.
- EmptyTree<T> has no data and no children.
- BinaryTree<T> is an abstract class that is extended by both EmptyTree<T> and ConsTree<T>.

Since BinaryTree<T> is abstract, it has no constructor.

Since EmptyTree<T> has no attributes, its constructor does nothing:

```
public EmptyTree() {
}
```

ConsTree needs data and two children. Its constructor is

```
public ConsTree(T data, BinaryTree left, BinaryTree right ) {
        this.data = data;
        this.leftChild = left;
        this.rightChild = right;
}
```

Lab 5 has a zip file of starter code. You need to expand that into a new folder before you start up Eclipse.

Build a Lab5 project in Eclipse using the folder of starter code.

The first thing you need to code for Lab 5 is the tree loading method of class TreeLoader. There is a stub for this already in the TreeLoader.java file:

```
BinaryTree<String> loadTreeFromFile(String fname) throws IOException {
        return new EmptyTree<String>();
}
```

Remove the return statement and replace it with code that implements the algorithm from the next slide.

The algorithm for reading a file makes use of a stack of BinaryTrees.

Start with an empty stack and a scanner for the file. At each step:

1. Get the next line of the file and separate into its data, left-bit and right-bit components.
2. If the **right-bit** is 1 pop the stack for the node's right child; otherwise make a new empty tree for the right child.
3. If the **left-bit** is 1 pop the stack for the node's left child; otherwise make a new empty tree for the node’s left child.
4. Make a new ConsTree node with the data and the two children.
5. Push this node onto the stack

When you reach the end of the file there should be 1 item on the stack --- the entire tree. If this is the case return the tree. If the stack is empty return an EmptyTree. If there is more than one tree on the stack throw an exception.

Once the loadTreeFromFile() method is written you can run the TreeApp program. This will give you a graphical interface that allows you to load and display trees in one window and it gives you buttons to run various methods on the trees.

The rest of the lab consists of writing those methods.

For each method the lab specifies you need to do the following steps:

a) We have already given an abstract version of the method to the BinaryTree class.
b) Update the stub for the concrete method with this name in the EmptyTree class. Since empty trees have no data and no children, this is usually one simple line of code.
c) Update the stub for the concrete method with this name in the ConsTree class. This is usually where you do actual work.

For example, one of the methods you are asked to write is nodeCount(). This is the number of nodes in the tree at and below this node. Remember that each node in one of our trees is the root of a tree, so every node has its own nodeCount(). Here are the steps for implementing nodeCount():

a. Make sure the abstract BinaryTree class has a declaration of

```
abstract public int nodeCount();
```

b. Empty trees have no nodes, so for the EmptyTree class this method is

```
public int nodeCount() {
        return 0;
}
```

c. The number of nodes in a constructed tree is the sum of the nodes in its two children, plus 1. Here is the code for the ConsTree class:

```
public int nodeCount() {
        return 1 + leftChild.nodeCount() + rightChild.nodeCount();
}
```

The "Count Nodes" button of the TreeApp program should now correctly count the nodes of the trees.

You should repeat these steps for each of the methods you need to implement.