CS-260	Lab #3: Binary Search Trees

General

For this lab you will further your studies of Abstract Data Types and by coding a pointer-based binary search tree ADT. The basic assignment for this lab is to implement a binary search tree based version of the spell checker. You are to do this with as minimal changes as possible to the existing code by replacing the use of the "TestList" class with your own implementation of a pointer based binary search tree class. This can be done with only changing the changing the dictionary field from "TestList" to your container class.

As with all of my labs, you can enhance this in any way (maybe add a GUI interface) as long as you implement the required components as defined in the lab. Doing additional features helps to improve your programming skills and knowledge.

Turn in you complete source, project files, directory structure, etc. of you ECLIPSE project directory (use the EXPORT tool in Eclipse to export the entire project directory). This zipped file should be turned in using the same method and done for previous labs.

General lab requirements:

At the beginning of the .java files your comments should include your name, creation date, Project #3, and purpose of this .java file.

Each public class and method should have a JavaDoc formatted comment specifying at least what the method will do. Methods that are implementations of interfaces that are already documented only need the @Override tag as they will get the JavaDoc from their interface definition. You should additionally have inline comments at every point in your program when the purpose is not obvious from the code itself.

Your indentation should be consistent. You should indent once for each block level (code between braces), and reduce the amount of indent used after the end of each code block. Use the Eclipse code formatter to clean up your code!

You should use EXCEPTION HANDLING where ever it makes sense to do so. For classes that implement standard java interfaces, use the javaDoc's for the interface as a guide for which exceptions should be handled for each interface method.

Concepts

This lab will introduce you to Java programming techniques for a BINARY SEARCH TREE ADT and gives you additional experience with parts of the Java Class Hiearchy. Take a look at the Java **TreeSet** class. You are to write a new class that behaves similarly to this class. We won't do the Ordered Set, just the **Set** interface.

Lab #3: Binary Search Trees

Background

Review the material in the online text. Be sure to complete the sections on **Overview of Tree ADTs** and **Binary Search Tree ADT** prior to doing this lab. I also suggest that you practice building BSTs using the TRAKLA2 Java applets (see the links on the class weekly outlines).

If you need to review or study recursion, here is a good tutorial:
http://wou.edu/las/cs/csclasses/JavaTutorial/cs151java.html
Start with #70 and go through #73. Please try to do the self-test quizzes.

Here's the Java class definition for a binary tree node should be:

```
class Node {
  // fields
    T item;
    int height;
    Node lChild;
    Node rChild;

  // methods

    // 3 constructors
    Node() {
        this(null, null, null);
    }

    Node(T item) {
        this(null, item, null);
    }

    Node(Node lChild, T item, Node rChild) {
        this.lChild = lChild;
        this.item = item;
        this.rChild = rChild;
        height = 0;
    }
}
```

The implementations for CS260 will always have the Node structures done as inner classes of the container class. Thus Nodes will NOT be known outside of the container.

Assignment

Lab 3a:

1. Write a new BINARY SEARCH TREE class that IMPLEMENTS the Java ***Set*** interface. Further, your container must be implemented as a generic container that can accommodate any object type that extends the "Comparable" interface. Your implementation must use a pointer-based, binary-search tree ADT.

   In order to correctly build the BST when you are comparing the DATA OBJECTS you must use the "Comparable" interface of your data items. Use

the compareTo(..) method to determine the ordering for insertion. The class declaration will be something like:

```
public class BSTreeSet<T extends Comparable<T>> implements Set<T>, CompareCount
```

2. You do not have to implement all of the public methods, but you must implement at least the following:

    i. **add(O), clear(), contains(O), isEmpty(), iterator(), and size()**
    ii. **getLastCompareCount( ),** but you only need to count comparisons in the contains method.
    iii. Consider the other methods a "challenge" exercise (in particular the remove(O) method).

3. Name your class "BSTreeSet" so that it can be tested with the jUnit tests that I have provided. It must implement both the java class library "Set" interface, and the "CompareCount" interface that I have provided.

4. Your implementation of "add" and "contains" MUST be written using recursive code. You already have experience with writing various iterative code, so you need to get some experience with recursive methods. We will be building upon these recursive methods in the next lab.

5. Your iterator implementation MUST use your QUEUE version of your doubly linked list from lab #1 to implement a breadth first walk through your tree.

6. **As long as you have a good start on all of the required methods, you will get full credit the #3a lab grade.**

Lab 3b:

1. Complete all 6 of the methods specified in 3a, and all of the requirements noted above. Test your binary search tree class using the jUnit tests that I have provided.

2. Get the height set correctly for each node by writing a "fixHeight(Node)" method that sets the height of a node by getting the greatest height of it's two children, adding one to that value and storing the result into the height field of the given in the parameter.

3. Now that you have a pointer based binary search tree class, edit the *Dictionary* class and replace the use of the "TestList" class with your BSTreeSet ADT. Run the spellchecker. The BSTreeSet SHOULD fail with a stack overflow exception on the sorted dictionary. Think about why this occurs even if your code passes all of the Junit tests. Now, change the text file used for the dictionary to the randomized dictionary text file (in the SpellCheckUser class). Execute the spellchecker and it should now run correctly without an exception. Think about why this now runs correctly.

Submission

Turn in your complete source, project files, directory structure, etc. of you ECLIPSE project directory. Please **export** the entire project using the Eclipse export utility). This zipped file should be turned in using the same method and done for previous labs.

3a) You must have the BSTreeSet class header defined correctly along with the needed imports, have generated all of the Set and CompareCount method shells, and started work on all of the required methods. Additionally you should have the "Node" inner class and the Iterator inner class. However, your code does not need to compile or correctly execute the jUnit tests at this time.

3b) Complete all of the required methods, implement a breadth first iterator using your Queue from Lab #2, get the height set correctly on all nodes, run and pass all of the jUnit tests, and update the Dictionary class to now use your BSTreeSet class. Run the spellchecker testing as described in the above section on 3b requirements.