COMPUTER SCIENCE 21A (SPRING TERM, 2018)
DATA STRUCTURES

PROGRAMMING ASSIGNMENT 2 - PART ONE

Chess Leaderboards with AVL trees

due Wednesday, March 14 @ 11:30pm

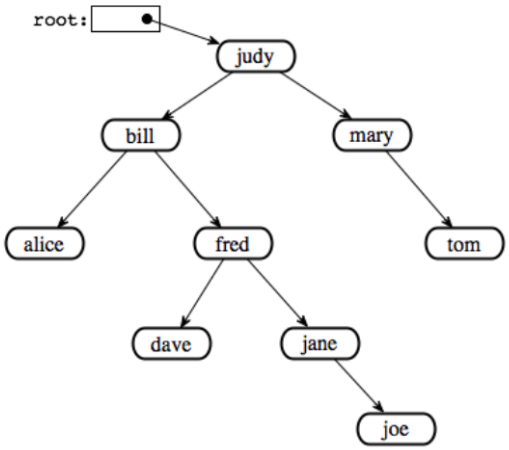
Background:

You have been hired by an international chess society to update their scoreboard system. The chess society is a membership organization, the members play chess against one another and the society keeps track of the outcomes. Each member has an Elo rating (a score based on their prior performance in chess matches), and the members are ranked based on their Elo ratings. Whenever there is a chess match between two society members, the winner's score increases and the loser's score decreases, and the rankings may also change. The society members are very competitive, so they frequently check their own ranking and Elo rating and the rankings and ratings of their opponents.

Your predecessor, who dropped out of data structures class after the first assignment, wrote the old scoreboard code using a linked list with insertion sort. This wasn't a problem when the society was small, but the society is growing and members are complaining that it takes too long to look up the rankings and ratings of individual players. Of course, we know that this is because searching for a particular member in a linked list takes O(n) time. You have been instructed by the chess society that these lookups need to be worst case O(log(n)) time or you will be fired. To accomplish this task, you will need to build a AVL tree, augmented to allow fast computation of player ranking.

Your Task:

For the first part of this programming assignment you will only be implementing a generic AVL tree, so don't worry about the chess leaderboard for now. Your tree should do self-balancing AVL insertions. Your tree must also allow for deletions, but self-balancing deletions is extra credit. For full credit you need only implement a delete method which returns the tree in a valid state. Be warned that these unbalanced deletions can break the AVL property of your tree, so once you delete from a tree there is no guarantee that insertions will work properly. If you are not doing the extra credit, you should not expect insertions to work properly after deletions when testing your code. Your tree will need a method getData(double value) which returns the data object stored in the node with this.value=value. Lastly, your tree node class should have a method that returns a String of the tree of names (toString() method of the data), using parentheses to separate subtrees. For example, for the tree below, the printout would be:
(((alice)bill((dave)fred(jane(joe))))judy(mary(tom)))



Code:

AVLNode.java

This will be the class for the nodes of your AVL tree. Each node will have a left child, a right child, and a generic data object, a double (the value that the node is sorted based on), balance factor for AVL balancing, number of nodes in the right subtree. You can add other fields you think will be helpful.

You will also need methods for AVLNode insert(T data, double value), AVLNode delete(double value), T getData(double value), a method for creating the tree string in parentheses form (as described earlier). Internally, you will want to write methods for left and right rotations. Make sure you maintain the balance factor and number of nodes in the right subtree after each time you modify the tree structure. Feel free to add other methods you think will be helpful. All insert methods should be self-balancing, and for those doing the extra credit portion, your delete code should also be self-balancing.

AVLUnitTesting.java

You will need to write JUnit tests for part one of the assignment. The tests your write should test all functionality of the AVL tree thoroughly. The quality of your tests will determine 30% of your grade for the first part of the programming assignment, so test your code well! If all your tests pass, you should feel confident that you have a flawless data structure.

Submission:

Submit via Latte a zip file with all the following files:

- AVLNode.java
- AVLUnitTesting.java

The zip file should be titled <your-username>-PA2-Part1.zip - for example, if your brandeis email is dilant@brandeis.edu, your zip file should be called dilant-PA2-Part1.zip

IMPORTANT

1. Your code should be well commented:
   - Add your name and email address at the beginning of each .java file.
   - Write comments within your code when needed. You must use Javadoc comments for your classes and methods.
2. Excluding arrays, you may only use data structures that you implement yourself.