CptS 233 Micro Assignment #4

For this micro assignment, you must implement AVL tree functions found inside AvlTree.java. The places where code needs to be added are noted with "MA TODO" comments.

If you build and run AVL_main.java you'll get a suite of tests on the AVL implemented in AvlTree.java. These tests exercise the interfaces, and if you successfully implement the MA TODO sections all tests should pass.

Just like in MA1, this assignment has been delivered to your Git repo. It is on a branch called MA3 AVL. You MUST go to the gitlab server and merge in the merge request created by this assignment. After that, you can run 'git pull' on your computer to get the new code to start working.

NOTE: If you use an IDE (Eclipse, etc) to edit your project, you'll need to put the files back into the MA3-AVL directory once you're done working. Various IDEs like to keep their files in various directory structures and I cannot build the testing system to detect and use them all. By putting the files back into the same places when you're done working. it allows my GitLab testing code to work properly. If you don't put the files back, it won't build and test properly. Feel free to make more sub directories to work in as you see fit, but these files are in the final place they need to run from.

There's a simple build script called "Makefile" in the MA directory. This is for a build system called 'make'. If you're on Linux or have XCode installed, you can run 'make build' and 'make test' to build and test the project. This isn't required for the project to work, but it's a shortcut. Windows people can use it too, but you'll need to figure out how to install GNU make.

Your code must be added, committed, and pushed to your Git repository to be turned in. Put a small file onto blackboard to show the TA that you're done working and need to be graded.

The three functions you'll need to implement are all related to detecting tree imbalances and then the key rotations to fix it. These functions are all found in AvlTree.java and clearly denoted at the top of the class implementation. I **highly** recommend that you carefully read through the rest of the code to see what might be useful in your implementation. One notable function is setHeight, but you should also note how things are structured in general.

Another thing to pay attention to is how the test code is written. Please look at how I test the different functions/API of the tree. Every kind of rotation is tested, both at the root and at lower levels in larger trees. If you've implemented balance, rotations, and handled the double rotation cases, then the tests should start passing.

Balance

This function determines where the imbalance at root exists (either right child or left child) and calls the appropriate rotation function (rotateLeft / rotateRight). The AvlNode class contains a getBalanceFactor() function that returns the balance factor for the node.

RotateLeft / RotateRight

These functions rotate the supplied root pointer either left (rotateLeft) or right (rotateRight). Be sure to use the following rotation algorithms:

Left (counter-clockwise) Rotation

1. Let CurrentRoot = the original root
2. Let NewRoot = CurrentRoot's **right** child
3. Set CurrentRoot's **right** child = NewRoot's **left** child
4. Set NewRoot's **left** child = CurrentRoot

Right (clockwise) Rotation

1. Let CurrentRoot = the original root
2. Let NewRoot = CurrentRoot's **left** child
3. Set CurrentRoot's **left** child = NewRoot's **right** child
4. Set NewRoot's **right** child = CurrentRoot

Grading

Your submission will be graded based on the following:

1. [8] Your solution builds, does not cause any runtime issues, and passes all test cases
2. [2] Style points:
    - You provide meaningful variable names
    - You provide sufficient and has meaningful comments
    - Your code is well structured