Important

There are a few guidelines you must follow in this homework. If you fail to follow any of the following guidelines you will receive a **0** for the entire assignment.

1. All submitted code must compile under **JDK 7**. This includes unused code, don't submit extra files that don't compile. (Java is backwards compatabile so if it compiles under JDK 6 it *should* compile under JDK 7)
2. Don't include any package declarations in your classes.
3. Don't change any *existing* class headers, constructors, or method signatures. (It is fine to add extra methods and classes)
4. Don't import anything that would trivialize the assignment. (e.g. don't import `java.util.LinkedList` for a Linked List assignment. Ask if you are unsure.)
5. You must submit your source code, the `.java` files, not the compiled `.class` files.

After you submit your files redownload them and run them to make sure they are what you intended to submit. We are not responsible if you submit the wrong files.

Assignment

For this assignment you will be implementing an AVL tree. The idea behind this type of BST is that it maintains its balance; unlike a normal BST, the AVL tree performs rotations to make sure that runtime does not degrade based on the order of entries.

The methods that you will be implementing are very similar to those in the BST assignment (in fact if you coded your BST's recursively you can write one extra method, and convert a BST to an AVL tree by changing 1-2 lines of code in each method). The only exception is that you will be maintaining the balance of the tree as you add and remove nodes.

Important things for this assignment:

1. balance factor is left height - right height
2. a left rotation means the tree is right heavy (i.e. add 10, 11, 12)
3. a right rotation means the tree is left heavy (i.e. add 10, 9, 8)
4. use the predecessor when removing something with two children
5. treat nulls as positive infinity

There are multiple ways the above things could be done to correctly implement an AVL tree, we are requiring you to implement them this way or you may lose points.

add

You will be adding a data entry to the avl tree. You will want to do this recursively. As your recursive calls are returning up, you will want to update the heights and balance factors of nodes. After updating a nodes value, you will want to call rotate to balance the tree if it needs to be.

addAll

This will add all of the entries in a collection. You can just repeatedly call add.

remove

This is another case where the tree may be thrown out of balance. Identical to the technique used in add, you will do the updating and rotating as you return out of your recursive calls.

contains

Checks to see if the tree contains a certain data entry

The following two methods are private, thus we cannot directly test them. What they should do is recommended below, however you can do whatever you like as long as your avl tree has the correct behavior. (you could even delete/rename any of the private methods – although this is strongly discouraged)

updateHeightAndBF

This method simply updates the height and balance factor for a single node.

rotate

This method first calls updateHeightAndBF on the given node, then if it needs a rotation, rotates the node and returns the new root of the subtree. (this is the only place you need to call updateHeightAndBF)

Deliverables

You must submit all of the following files.

1. `AVL.java`

You may attach them each individually, or submit them in a zip archive.