Cosc 2P03: Advanced Data Structures in Java
Assignment #2
Due date: 5:00 pm, Friday, June 8th

This assignment covers the AVL Tree structure, its use, and general tree traversals.

Part 1: Binary Search Trees, AVL Trees, and Iterators

You've been provided with a Binary Search Tree, which includes an iterator that allows for a breadth-first traversal. For the first part, your task is to write an AVL Tree, based on the BST. Your new class (AVLTree.java) must *extend* the BinarySearchTree class. Additionally, you'll be creating extra iterators for the new structure.
Your AVLTree class will:

- **Override** the 'add' method.
- Include a function, *preorder()*, that creates an iterator on the AVL Tree that facilitates a preorder traversal on the tree. Note: code to help you with this is included in the slides.
- Include a function, *inorder()*, that creates an iterator on the AVL Tree that allows for an inorder traversal on the tree. Note: this is harder. Work it out on paper before writing it!
- Still function perfectly with the inherited *iterator()* function, which allows for the breadth-first traversal

Important Points:

- The preorder and inorder functions will not be accessible in for-each loops. If your version allows it, then you made a mistake
- Your iterators can't simply do an entire traversal in advance, and keep all the values in some structure until they're needed. The iterators must actually traverse the tree progressively as *next()* is called
  - This also means you can't have it start over each time *next()* is called
- If you neglect to extend the included BinarySearchTree class, you will receive zero on this entire assignment. For realsies
- You should have no problem getting generics working. If your code generates compiler warnings, expect a *heavy* penalty

Part 2: Testing and demonstration

For the second part, you'll be writing a command-line application that helps with testing BST and AVL structures. The requirements are as follows:

- You must be able to test both your AVL and the included BST on both Strings ***and*** Integers
  - For both types, the user must be able to add an arbitrary number of each
  - For both types, at least the breadth-first traversal must be useable for dumping the contents onto the screen
- You must be able to test all three iterators on the same AVL tree of the same Integer data
  - You don't need to bother coding this for Strings, if you don't want to
- You must include an option for time trials of either random data or data from a file, to compare the insertion, traversal, and `findMin` of both the AVL and the BST

    - The precise means by which you do this is up to you. If there's a significant difference in the efficiencies of either operation, you should be able to demonstrate it. My suggestion is to just see if you can find something like a Shakespeare play and add the words from that, but use your best judgement
- If you wish to add any additional features, please feel free to
- Prepare a short writeup, commenting on the performance of the AVL, compared to the BST. Please export this as a `.pdf` file; don't submit a Word document.

Final Note: class names provided are not suggestions; they are mandatory. Do not modify any of the included files at all.

Submission Guidelines:
Electronic submission is required. Slap your files into a dedicated directory on sandcastle, ssh (or putty) in, and run 'submit2p03'.
Ensure that you also include some sample output (a .txt file is fine. You may wish to use `tee` to make capturing output easier).