Important

There are general submission guidelines you must always follow. If you fail to follow any of the following guidelines you risk receiving a **0** for the entire assignment.

1. All submitted code must compile under **JDK 8**. This includes unused code, so don't submit extra files that don't compile.
2. Do not include any package declarations in your classes.
3. Do not change any existing class headers, constructors, or method signatures.
4. Do not add additional public methods when implementing an interface.
5. Do not use anything that would trivialize the assignment. (e.g. don't import/use `java.util.LinkedList` for a Linked List assignment. Ask if you are unsure.)
6. You must submit your source code, the `.java` files, not the compiled `.class` files.
7. Do not add any new instance variables.
8. All methods must be efficient, even if a runtime is not specified.
9. After you submit your files redownload them and run them to make sure they are what you intended to submit. You are responsible if you submit the wrong files.

AVL Tree

You are to code an AVL. An AVL is like binary search in that it is collection of nodes, each having a data item and a reference pointing to the left and right child nodes. The left child nodes (and all of its children) are less than the data. The right child nodes and all of its children are greater than or equal to the data. However, unlike a BST, an AVL tree is self-balancing, and you must implement the appropriate rotations to ensure the tree is always balanced.

All methods in the AVL Tree that are not O(1) must be implemented recursively, except for level order traversal.

Your AVL tree implementation will implement the AVL interface provided. It will have two constructors: the no-argument constructor (which should initialize an empty tree), and constructor that takes in data to be added to the tree, and initializes the tree with this data.

Nodes

The AVL tree consists of nodes. The `AVLNode` class will be given to you; do not modify it.

Methods

You will implement all standard methods for a Java data structure (add, remove, etc.) See the interface for details. Note that some methods are worth more than others. If add is incorrect, then you are likely to fail most tests, as adding is crucial to the usability of a data structure.

Traversals

You will implement 4 different ways of traversing a tree: pre-order traversal, in-order traversal, post-order traversal, and level-order traversal. The first 3 MUST be implemented recursively; level-order may be implemented iteratively. You may import Java's LinkedList/ArrayList/Queue classes as appropriate for these methods (but they may only be used for these methods).

Balancing

Unlike a Binary Search Tree, an AVL Tree must be self-balancing. Each node has two additional variables, `height` and `balanceFactor`. The height variable represents the height of the node (see below). The balance factor of a node is equal to its left child's height minus its right child's height. The tree should rotate appropriately to make sure the tree is always balanced.

Height

You will implement a method to calculate the height of the tree. The height of any given node is max(child nodes height)+1. A leaf node has a height of 0. Also, the height variable of each node must be set to be the height of the node (calculated using the above formula).

Exceptions

When throwing exceptions, you must include a message by passing in a String as a parameter. For example:

```
throw new PDFReadException("Did not read PDF, will lose points.");
```

Style and Formatting

It is important that your code is not only functional but is also written clearly and with good style. We will be checking your code against a style checker that we are providing. It is located in resources along with instructions on how to use it. We will take off a point for every style error that occurs. If you feel like what you wrote is in accordance with good style but still sets off the style checker please email Jonathan Jemson (jonathanjemson@gatech.edu) with the subject header of "CheckStyle XML".

Javadocs

Javadoc any helper methods you create in a style similar to the Javadocs for the methods in the interface.

Forbidden Statements

You may not use these in your code at any time in CS 1332.

- `break` may only be used in switch-case statements
- `continue`
- `package`
- `System.arrayCopy()`
- `clone()`
- `assert()`
- `Arrays` class
- `Array` class
- `Collections` class
- Reflection APIs

Debug print statements are fine, but should not print anything when we run them. We expect clean runs - printing to the console when we're grading will result in a penalty. If you use these, we will take off points.

Provided

The following file(s) have been provided to you. There are several, but you will only edit one of them.

1. `AVLInterface.java` This is the interface you will implement when designing your AVL. All instructions for what the methods should do are in the javadocs. **Do not alter this file.**
2. `AVL.java` This class is where you will be implementing your AVL. **Do not add any public methods to this file.**
3. `AVLStudentTest.java` These are some sample JUnits, similar to those that will be used to grade your assignment. **Passing this does not guarantee any sort of grade.**
4. `AVLNode.java` This class is for the `AVL` to use only. **Do not alter this file**

Deliverables

You must submit all of the following file(s). Please make sure the filename matches the filename(s) below. Be sure you receive the confirmation email from T-Square, and then download your uploaded files to a new folder, copy over the interfaces, recompile, and run. It is your responsibility to re-test your submission and discover editing oddities, upload issues, etc.

1. `AVL.java`

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