Important

There are general homework 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. Any compile errors will result in a 0.

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.

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. Always be very conscious of efficiency. Even if your method is to be $O(n)$, traversing the structure multiple times is considered non-efficient unless that is absolutely required (and that case is extremely rare).

7. You must submit your source code, the `.java` files, not the compiled `.class` files.

8. 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

You are to code an AVL tree. An AVL tree is like a binary search tree in that it is a 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 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. You must do good, proper recursion, as done in homework 5.

Your AVL tree implementation will implement the AVL interface provided. It will have two constructors: a no-argument constructor (which should initialize an empty tree), and a 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 has been 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 BST, 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).

A note on JUnits

We have provided a **very basic** set of tests for your code, in `AVLStudentTests.java`. These tests do not guarantee the correctness of your code (by any measure), nor does it guarantee you any grade. You may additionally post your own set of tests for others to use on the Georgia Tech GitHub as a gist. Do **NOT** post your tests on the public GitHub. There will be a link to the Georgia Tech GitHub as well as a list of JUnits other students have posted on the class Piazza.

If you need help on running JUnits, there is a guide, available on T-Square under Resources, to help you run JUnits on the command line or in IntelliJ.

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 T-Square, under 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 existing Javadocs. If a method is overridden or implemented from a superclass or an interface, you may use `@Override` instead of writing Javadocs.

Exceptions

When throwing exceptions, you must include a message by passing in a String as a parameter. **The message must be useful and tell the user what went wrong**. "Error", "BAD THING HAPPENED", and "fail" are not good messages. The name of the exception itself is not a good message.

For example:

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

throw new IllegalArgumentException("Cannot insert null data into data structure.");
```

Generics

If available, use the generic type of the class; do **not** use the raw type of the class. For example, use `new LinkedList<Integer>()` instead of `new LinkedList()`. Using the raw type of the class will result in a

penalty.

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

- `Collection.toArray()`

- Reflection APIs

- Inner, nested, local, or anonymous classes

Debug print statements are fine, but nothing should be printed 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 edit only one of them.

1. `AVLInterface.java`

   This is the interface you will implement in `AVL`. All instructions for what the methods should do are in the javadocs. **Do not alter this file.**

2. `AVL.java`

   This is the class in which you will implement `AVLInterface`. **Do not add any public methods, classes, instance variables, or static variables, and do not change the method header for any existing method**.

3. `AVLNode.java`

   This class represents a single node in the AVL tree. **Do not alter this file.**

4. `AVLStudentTests.java`

   This is the test class that contains a set of tests covering the basic operations on the `AVL` class. It is not intended to be exhaustive and does not guarantee any type of grade. **Write your own tests to ensure you cover all edge cases.**

Deliverables

You must submit **all** of the following file(s). Please make sure the filename matches the filename(s) below, and that *only* the following file(s) are present. T-Square does **not** delete files from old uploads; you must do this manually. Failure to do so may result in a penalty.

After submitting, 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`