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, instance/global variables, 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.ArrayList` for an Array 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 inefficient 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 required to implement an AVL tree. An AVL is a special type of binary search tree that follows all the same rules: each node has 0-2 children, all data in the left subtree is less than the node's data, and all data in the right subtree is greater than the node's data. The AVL differs from the BST with its own self-balancing rotations, which you must implement.

All methods in the AVL tree that are not O(1) **must be implemented recursively**. Good recursion with simple, focused states is strongly encouraged for this assignment in particular.

Your AVL tree 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.

Balancing

Each node has two additional instance variables, `height` and `balanceFactor`. The `height` variable should represent the height of the node (recall that a node's height is `max(child nodes' heights)+1`. The balance factor of a node should be equal to its left child's height minus its right child's height. The tree should rotate appropriately to make sure it's always balanced. Keep in mind that you will have to update these instance variables; they are not updated automatically.

Grading

Here is the grading breakdown for the assignment. There are various deductions not listed that are incurred when breaking the rules listed in this PDF, and in other various circumstances.

Methods:	
add	21pts
remove	22pts
get	5pts
contains	5pts
getSecondLargest	7pts
equals	7pts
height	3pts
clear	2pts
constructor	3pts
Other:	
Checkstyle	10pts
Efficiency	15pts
Total:	100pts

Keep in mind that some functions are dependent on others to work, such as remove methods requiring the add methods to work. Also, the size function is used many times throughout the tests, so if the size isn't updated correctly, many tests can fail.

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 Raymond Ortiz (rortiz9@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. Any Javadocs you write must be useful and describe the contract, parameters, and return value of the method; random or useless javadocs added only to appease Checkstyle will lose points.

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:

Bad: `throw new IndexOutOfBoundsException("Index is out of bounds.");`

Good: `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 or nested classes
- Lambda Expressions
- Method References

If you're not sure on whether you can use something, and it's not mentioned here or anywhere else in the homework files, just ask.

Debug print statements are fine, but nothing should be printed when we run your code. We expect clean runs - printing to the console when we're grading will result in a penalty. If you submit 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 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`. Feel free to add private helper methods but **do not add any new public methods, inner/nested classes, instance variables, or static variables**.

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`