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.

Binary Search Trees

You are to code a binary search tree. A binary tree is a collection of nodes, each having a data item and a reference pointing to the left and right child nodes. What makes a binary tree a binary search tree is that it must follow the order property: for any given node, its left child and all of its children must be less than the current node while its right child and all of its children must be greater than the current node. In order to compare the data, all elements added to the tree must implement Java's generic `Comparable` interface.

It will have two constructors: the 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. Any attempts to add data that is already in the tree should be ignored (the tree shouldn't be changed, and the duplicate item shouldn't get added).

You may import Java's `LinkedList/ArrayList` classes for the 4 traversal methods, but only for these methods.

Recursion

Since trees are naturally recursive structures, all methods that are not $O(1)$ **must be implemented recursively**, except for level order traversal. You'll also notice that a lot of the public method stubs we've provided do not contain the parameters necessary for recursion to work, so these public methods act as "wrapper methods" for the user to use. These wrapper methods will just call another private helper method that is recursive. To reiterate, **do not change the method headers for the provided methods.**

For methods that change the structure of the tree in some way, we highly recommend you use a technique taught in class called pointer reinforcement. It's not required, but it will make the homework cleaner, and it'll also help greatly when we get to a later homework.

Nodes

The binary search tree consists of nodes. The `BSTNode` 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.) in addition to a few other methods. Some of these methods are functions that you'd expect from a BST (such as the traversals) while some of the other ones serve more as practice BST recursion problems for you.

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 is best implemented iteratively. For a level-order traversal, you may use Java's `Queue` interface (and an implementing class for it such as `LinkedList`).

Height

You will implement a method to calculate the height of the tree. The height of any given node is `max(left node's height, right node's height) + 1`. A leaf node has a height of 0. Based on this recursive definition, this means that `null` nodes would have a height of -1.

Comparable

As stated, the data in the BST must implement the Comparable interface. As you'll see in the java files, the generic typing has been specified to require that it implements the `Comparable` interface. You use the interface by making a method call like `data1.compareTo(data2)`. This will return an `int`, and the value tells you how `data1` and `data2` are in relation to each other.

- If positive, then `data1 > data2`.
- If negative, then `data1 < data2`.
- If zero, then `data1` equals `data2`.

Do note that the returned value can be any integer in Java's `int` range, not just -1, 0, 1 as you may have seen in some examples.

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	15pts
remove	19pts
get	6pts
contains	6pts
traversals	8pts
isBST	10pts
height	3pts
clear	3pts
constructor	5pts
Other:	
Checkstyle	10pts
Efficiency	15pts
Total:	100pts

A note on JUnits

We have provided a **very basic** set of tests for your code, in `BSTStudentTests.java`. These tests do not guarantee the correctness of your code (by any measure), nor do they 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 Canvas under Files, 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 on Canvas, under Files, 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 Tim Aveni (tja@gatech.edu) with the subject header of "[CS 1332] 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.

Vulgar/Obscene Language

Any submission that contains profanity, vulgar, or obscene language will receive an automatic zero on the assignment. This policy applies not only to comments/javadocs but also things like variable names.

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.

- `package`
- `System.arraycopy()`
- `clone()`
- `assert()`
- `Arrays` class
- `Array` class
- `Thread` class
- `Collections` class
- `Collection.toArray()`
- Reflection APIs
- Inner or nested classes
- Lambda Expressions
- Method References (using the :: operator to obtain a reference to a method)

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 we've noted the ones to edit.

1. `BST.java`

   This is the class in which you will implement the BST. Feel free to add private helper methods but **do not add any new public methods, inner/nested classes, instance variables, or static variables**.

2. `BSTNode.java`

   This class represents a single node in the BST. It encapsulates the `data`, `left`, and `right` reference. **Do not alter this file.**

3. `BSTStudentTests.java`

   This is the test class that contains a set of tests covering the basic operations on the `BST` 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. If you make resubmit, make sure only one copy of the file is present in the submission.

After submitting, double check to make sure it has been submitted on Canvas and then download your uploaded files to a new folder, copy over the support files, recompile, and run. It is your responsibility to re-test your submission and discover editing oddities, upload issues, etc.

1. `BST.java`