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.

Hash Maps

In this homework, you will implement a key-value hash map with a linear probing collision resolution strategy. A hash map maps keys to values and allows $O(1)$ average case lookup of a value when the key is known. If adding to the table would cause the LF to exceed (greater than, not greater than or equal to) the max load factor provided in `HashMap.java`, then the hash map should be resized to have a capacity of $2n+1$ where n is the current capacity. See the javadocs for specific instructions on when to resize.

The table **should not add duplicate keys**, but duplicate values are acceptable. In the event of a duplicate key, replace the existing value with the new value.

There are two constructors in `HashMap.java`. As per the javadocs, you should use constructor chaining to implement the no-arg constructor.

Hash Functions

You should not write your own hash functions for this assignment; use the `hashCode()` method (every `Object` has one). If this is a negative value, mod by table length first, then take the absolute value (it must be done in this order to prevent overflow in certain cases).

Linear Probing

Your hash map must implement a linear probing collision policy. If the hash value of the key is occupied, probe in linear increments. For example, if the hash value of your key is 7 with a backing array of capacity 9, and index 7 in the array is occupied, check index $7+1 \mod 9$, then $7+2 \mod 9$, then $7+3 \mod 9$, etc. until we either hit a null spot in the array or until we attempt `table.length` probes.

Adding Items

When adding a key/value pair to a hash map, add the pair to the array in the correct position. Also remember that keys are unique in a hash map, so you must ensure that duplicate keys are not added. When searching for a spot to add, after ensuring no duplicates, you should add at the first encountered removed spot (if there are any). If no removed spots were encountered, add at the null spot that terminated your search.

Removing Items

Since linear probing is an open addressing scheme, you cannot set removed entries to null. Instead, you need to implement a "soft remove" using the removed flag in `MapEntry.java`. Keep in mind that all the flag does is provide a way to keep track what entries have been removed, but you need to implement the logic for what to do with the removed entries. Though the objects may still be in memory, as far as the user is concerned, the data has been removed, and your code's behavior should reflect this expectation.

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:	
put	20pts
remove	10pts
get	10pts
containsKey	10pts
keySet	5pts
values	5pts
resizeBackingTable	10pts
clear	5pts
Other:	
Checkstyle	10pts
Efficiency	15pts
Total:	100pts

Keep in mind that add functions are necessary to test other functions, so if an add doesn't work, remove tests might fail as the items to be removed were not added correctly. Additionally, the size function is used many times throughout the tests, so if the size isn't updated correctly or the method itself doesn't work, many tests can fail.

A note on JUnits

We have provided a **very basic** set of tests for your code, in `HashMapStudentTests.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 LinkedNode<Integer>()` instead of `new LinkedNode()`. 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. `HashMap.java` This is the class in which you will implement the methods for a HashMap. Feel free to add private helper methods but **do not add any new public methods, inner/nested classes, instance variables, or static variables**.
2. `MapEntry.java` This class stores a key-value pair and a removed attribute for your hash map. **Do not alter this file.**
3. `HashMapStudentTests.java` This is the test class that contains a set of tests covering the basic operations on the `HashMap` 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. `HashMap.java`