CSC 355. Discrete Structures and Basic Algorithms
Project 5. Hashtables

General Guidelines.

The method signatures indicate the required methods. You may need additional methods or classes that will not be directly tested but may be necessary to complete the assignment, and you are welcome to add those into the classes.

Unless otherwise stated in this handout, you are welcome to add to/alter any provided java files as well as create new java files as needed, but make sure that your code works with the provided test cases. Your solution must be coded in Java.

Note on academic dishonesty: Please note that it is considered academic dishonesty to read anyone else's solution code, whether it is another student's code, code from a textbook, or something you found online. You MUST do your own work! It is also considered academic dishonesty to share your code with another student. Anyone who is found to have violated this policy will be subject to consequences according to the syllabus and university policy.

Note on grading and provided tests: The provided tests are to help you as you implement the project and (in the case of auto-graded sections) to give you an idea of the number of points you are likely to receive. Please note that the points indicated when you run these tests locally are not your final grade. Please also note that these test cases are not likely to be exhaustive and find every possible error. Part of programming is learning to test and debug your own code, so if something goes wrong, we can help guide you in the debugging process, but we will not debug your code for you.

More specifically, you are provided skeleton code and must implement specific operations for the data structures.

Part 1. Hashtable (60 points)

In the file called `Hashtable.java`, you must complete the implementation of a Hashtable that uses linear probing to handle collisions. Please pay close attention to the specific guidelines for resizing and hashing so that you can pass the provided tests.

Guidelines

- First, familiarize yourself with the Pair.java class that is provided for you. This is a general class for key-value pairs that will be used throughout the project.
- Second, familiarize yourself with the Hashtable.java class that is provided. Much of the basic set up is done for you. You just need to fill in the code for the operations.
- I recommend you have a private method for finding the hash value for a key. You should just use the `hashcode` function (which exists for all Java objects) along with modular hashing.
- I recommend having a private method for resizing the table. Make sure that you skip over deleted items when you resize, and don't forget to rehash all the elements.
- Instead of requiring the table sizes to be prime (which would necessitate a certain amount of overhead), I have provided a simple function to get a number that is somewhat likely (but not guaranteed) to be prime. Use this when you resize.
- You will have to figure out how you want to manage deleted items. I suggest either denoting deleted items in the Pair class (which you can change as needed) or keeping a boolean array to keep track of what has been deleted.
- Although I've tried to simplify the generics for you by providing the skeleton code, you will still need to do a little casting. The generic types will be treated as Objects, which will need to be cast to whatever the actual type is before you can do anything useful with them.

Required Methods to be Implemented

Method Signature	**Description**
`public V get(K key)`	`return the value associated with the given key or null if the key does not exist in the table`
`public void put(K key, V val)`	`Insert the (key, value) pair into the table or, if the key already exists in the table, update the value.`
`public V delete(K key)`	`Delete the key and associated value from the table. Don't forget that you don't want gaps in your clusters, so be careful how you do this. Return the deleted value OR null if the key`

	was not in the table.
public boolean isEmpty()	Return true if the table is empty and false otherwise. This function should be very easy to implement.
public int size()	Return the number of key-value pairs in the table. This function should be very easy to implement.

Grading.
Note: You are allowed to modify the code in the skeleton if something doesn’t work with your implementation.

Submission

Submit your java files in D2L.