**CacharySimpleHashMap User Manual**

A custom HashMap implementation.

Cachary Tolentino

CSCI-3327

**Table of Contents**

**Software Description3**

Detailed Description3.1

System Requirements3.2

**Installation Guide4**

**Class Overview6**

**Result Analysis8**

**Software Description**

A custom HashMap implementation.

**Detailed Description**

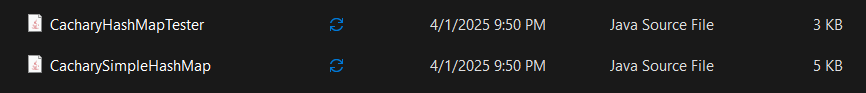
The class allows the user to create a custom HashMap using a custom Hash function which calculates the number of characters within a given string.

**System Requirements**

* A working device, primarily a desktop or laptop
* An IDE (ex: VSCode, Eclipse, etc…)
* Java JDK (Ver. 17 & up) & JRE (SE 17 & up)

**Installation Guide**

To begin using CacharySimpleHashMap, you will need to download two files. One is “CacharySimpleHashMap.java” and the other is “TestCacharyHashMap.java” (optional).



After downloading the files, simply move the files to the folder containing your project. Once done, you can open your preferred IDE (for this example we will be using VSCode). Then you can open the folder or the file itself within your IDE.

A screenshot of a computer

AI-generated content may be incorrect.

If you opened the folder containing the files then it should look similar to the image below.

A screenshot of a computer

AI-generated content may be incorrect.

If you only imported the CacharySimpleHashMap file then you can simply start using the class within your own personal project. Otherwise, if you also imported the TestCacharyHashMap, then you can open that file and run it.

The result will be displayed on the console, unless there are graphical displays being run.

**A screen shot of a computer

AI-generated content may be incorrect.**

**Class Overview**

**Global Variables**

The class used three global variables: data, dataSize, and currentCount. Data is of type LinkedList<String>[]. The indices of the array will represent the key to get the value which will be stored within the LinkedList in the index of the array. Datasize is of type int. It is meant to store the current maximum size of the array. Lastly, currentCount is of type int aswell. It is meant to track the total amount of elements added.

**Default Constructor**

The default constructor simply creates data and populates it with LinkedList objects. It is by default setting the size of the array at 10. Datasize and currentCount is initialized accordingly.

**Constructor with parameters**

Similarly to the default constructor it will also populate the data with LinkedList objects. But the size will be determined by the user defined input, inputSize. Datasize and currentCount is initialized accordingly.

**dumbHash() Function**

The dumbHash function simply returns the number of elements of a given string.

**put() Function**

The put function simply places the given value to its appropriate index based on the value’s hash value using the dumbHash function. The value of the index is then modded to ensure that the index remains within the bounds of the array index. It will automatically resize given that there are no more space left to place the values.

**contains() Function**

The contains functions simply returns true or false depending on whether the given value trying to be found is win the hashmap. It does so by finding the key value using the dumbHash function and checking at the found index.

**resize() Function**

The resize function increases the underlying array of the hashmap. It does so by creating a new array that is double the size of the current array. It will then rehash all the current values in the array and move them into the new array with their new corresponding indices.

**remove() Function**

The remove function simply removes a provided value. It does so by getting the index (the hash value of the given value) and removing the value at that index using LinkedLists’ built in remove function.

**printMap() Function**

The printMap function simply prints out all the values within the current map.

**Result Analysis**

In the following tester class, it has the following results:

**A screen shot of a computer

AI-generated content may be incorrect.**

The objects map1 and map2 were originally populated with the following values:

**A screen shot of a computer program

AI-generated content may be incorrect.**

Based on the results we can see that the class accurately resizes given that there is not enough space left. Map1 had 11 values inserted thus resizing from 10 to 20. Furthermore, the contains function works properly in checking if the entire HashMap contains the given value (works for values within the data structure or not). The additional remove function also works properly as we check if the Kiwi value still exists after manually calling the remove function.

By using the printMap function we can see how HashMap stores all the values that were added. Key notes to take away here are the nodes containing more than one value. These values all share one thing: their length modded by the current HashMap’s length are the same, thus being inserted in the same index. We can also notice that the length of the map heavily plays a role in which index each input will be placed. For example, both maps have “Incompatibility” yet they have it in different indices, this is due to the mod values being different. Ex: 15 mod 10 = 10, but 15 mod 15 = 0.

A screen shot of a computer program

AI-generated content may be incorrect.