Data Structures in Java: Programming Assignment 7
Creating Your Own HashMap

1. **Objective**
   Your goal is to create your own HashMap data structure, called MyHashMap, that uses chaining to resolve collisions.

2. **Problem**
   Download the following files from CourseWorks:
   - MyHashMap.java
   - Map.java
   - Entry.java

   You will implement the 5 methods found in the MyMap interface inside the MyHashMap class. They are found below:

```
/**
 * Returns the number of key-value mappings in this map.
 * @return the number of key-value mappings in this map
 */
int size();

/**
 * Returns true if this map contains no key-value mappings.
 * @return true if this map contains no key-value mappings
 */
boolean isEmpty();

/**
 * Returns the value to which the specified key is mapped, or null if this
 * map contains no mapping for the key.
 * @param  key the key whose associated value is to be returned
 * @return the value to which the specified key is mapped, or null if this
 *         map contains no mapping for the key
 */
V get(K key);

/**
 * Associates the specified value with the specified key in this map. If the
 * map previously contained a mapping for the key, the old value is replaced
 * by the specified value.
 * @param key   the key with which the specified value is to be associated
 * @param value the value to be associated with the specified key
 * @return the previous value associated with key, or null if there was no
 *         mapping for key
 */
V put(K key, V value);

/**
 * Removes the mapping for a key from this map if it is present.
 * @param key the key whose mapping is to be removed from the map
 * @return the previous value associated with key, or null if there was no
 *         mapping for key
 */
V remove(K key);
```

In addition to these 5 methods, you will also implement rehash(), a private method found inside the MyHashMap class.

3. **Mode of Operation**

MyHashMap is implemented as an array of Entry references. The array starts out with size 101, but the max load factor is set to 0.75, so rehashing will be required when further insertions cause the table to exceed ¾ capacity.

To insert a key-value pair into a MyHashMap object, call key.hashCode() to get the key's hashcode. You must call `private int hash(K key)` from within `get()`, `put()`, `remove()`, and `rehash()` to ensure the hashcode is non-negative. Mod the returned hashcode by the table size to determine the index where the Entry should be inserted. Each Entry object has a key, value, and reference to the next Entry in the list. Managing the next pointers is required to maintain a list of Entry objects. Note that there is no LinkedList class in use, and you MUST NOT use a LinkedList or any of the Java Collections classes. table[index] is either null, or it refers to an Entry. If the key is already found in the table, update its value and return the old value associated with the key. If the key it not found, insert the key-value pair and return null. If another Entry hashes to the same index in the table, insert it at the head of the list at that index.

After inserting a new Entry into the table, you must check if the current load factor exceeds the max load factor. If it does, you will need to create a new array of the next prime size. primeIndex is initially set to 0, so that you can index into the primes array and retrieve the size 101. Increment primeIndex when rehashing. Note that the allowed prime sizes are 101, 211, 431, 863, 1733, 3467, 6947, 13901, 27803, 55609, 111227, and 222461. Every Entry in the original table must be inserted into the new table. You will need to take each Entry's hashCode and mod it by the new table size to determine its index in this larger table. Do not simply copy all the Entries in one chain in the original table to the same index in the new table, since modding by the new table size may not yield the same value. If the table is already of size 222461 and the load factor is exceeded, do not rehash.

To retrieve the value associated with a key, find the index of the key in the table. If the Entry at that index is null, the key is not found. Otherwise, iterate over the Entries, looking for the key. If it's found, return the value in the Entry object. If it's not found, return null.

Removal requires slightly more work than retrieval. The notable difference is when the Entry to remove is one of several within the linked list. Be sure to take appropriate and different actions when either removing the head of the list versus some non-head Entry.

4. **Tips**

Be sure to test your work thoroughly. Experiment with the code in the main() method. There is even a toString() method that helps you see how the data resides in the MyHashMap data structure.

If you’re having trouble, I suggest starting with a smaller table size, such as 17. With a smaller size, you should be able to more easily see a failure in your code.

Work on the put() and rehash() methods first, then implement get() and remove(). size() and isEmpty() are one-line methods that can be completed at any time.

You must reuse the same Entry objects from the original table when you rehash. Avoid using the new operator, as this will make things run slowly. Do not call put() from rehash either, since put() does additional work not needed in a rehash.

With respect to when rehashing should take place, always insert the Entry into the table first. Most of the time, it will work fine. After inserting, check the new load factor. If it exceeds the max allowed load factor, rehash all Entries into the larger table.

5. **Submission**

Create a zip file called hw7.zip containing only:

- `MyHashMap.java`

Do not put your code in a Java package or submit extraneous files or folders. Upload your submission to Canvas. Only the final submission will be graded.