|  |  |
| --- | --- |
| **Project #9** | **CMSC 132** |
| **Due:  Sunday, 12/11 at 11:00PM** | **Object-Oriented Programming II** |
| **Type of project: Closed** | **Fall 2016** |

**Hash Table**

**Introduction**

For this project you will create a class called MyHashSet, which is a slightly simplified version of a HashSet. As you might guess, we will use a hash table to store the elements in the set. The "buckets" in the table will be implemented as linked lists.

If you take a look at the code distribution, you'll find a Node class nested inside the MyHashSet class. The Hash table will be an array of Node references. An empty bucket will be represented as a null entry in the array. Each Node encapsulates a data element and has a "next" reference, which will either refer to another Node in this same bucket, or will be null if this Node is the last one in the bucket.

The MyHashSet class will implement the Iterable interface, which of course means that you must implement the "iterator" method. Your Iterator will allow the user to iterate over all of the elements stored in the table. Note that you do not have to implement the remove method for the iterator.

**Implementation Details**

Most of the details are specified  [here in the Javadoc.](http://docs.google.com/doc/index.html) But here are some additional remarks:

### Instantiating the Table

You'll notice that the Node class has a static method called "makeArray". You must rely on this method anytime you need to create a new table. (You will need to create a new table when the MyHashSet is constructed and also when it is expanded because it has become too full.) We implemented this method for you because generics get a little tricky when used with arrays.

### Size vs. Capacity

In the Javadoc, we will use the term "size" to indicate the number of data elements that are stored in the table. The term "capacity" will be used to describe the length of the table (the number of buckets).

### Hashing

You may assume that objects to be inserted into the MyHashSet will satisfy the "HashCode Contract", as described in class. You should use the "Multiplicative Congruence" method for determining which bucket a particular object belongs in:

bucket number = | (p \* hashCode) | % n

where p is a large prime of your choosing, and n is the capacity (length) of the table. Be sure to always use the same prime, p, throughout the life of the table!

### Iterator

Your Iterator should allow the user to iterate through all of the elements in the MyHashSet. The Iterator class should be implemented as an inner class of the MyHashSet class. **You do not have to implement the remove method for the iterator.**

The order in which the iterator traverses the data is irrelevant, and doesn't need to be the same each time.

Your "next" method should throw an IllegalStateException if someone attempts to call it at a time when hasNext would return false.

By the way, it is okay if your Iterator does not throw ConcurrentModificationExceptions even when it should.

**Additional Requirements**

* You may not add any instance variables or static variables to the MyHashSet class. If you cheat by using any additional variables/structures, your grade on this project will be 0.
* Hash tables are supposed to be fast! Make sure that your methods make use of hashing whenever possible. For example, consider the "contains" method, which will allow the user to ask whether or not a specified element is in the set. One (very poor) idea for implementing this method would be to simply iterate through the elements in the table looking for the specified element. Of course this would defeat the purpose of using a hash table! You are required to take full advantage of hashing whenever possible while implementing this project.
* You must use generics properly. Eclipse should not generate any warnings (in yellow) for your code.
* You may not store the data in two separate structures at any time. In other words, you may not create an ArrayList or something that holds a separate copy of the data at any time (including within your Iterator class). If you fail to adhere to this rule, your grade on the project could be as low as 0.
* Avoid redundant code as much as you can.
* Do not modify the DEFAULT\_INITIAL\_CAPACTIY constant, or the LOAD\_FACTOR constant.

**Grading**

The public tests and release tests together will comprise 100% of your grade on the project. However, keep in mind that we will be inspecting your code to be sure that you followed the rules above. If you didn't, you may receive a 0 on this project.

[Web Accessibility](https://www.umd.edu/web-accessibility)