Home Depot: AoA Edition

Who wants to rely on someone else's implementation of a hashSet? Not US! As a result we will be implementing our own version of the HashSet Data Structure, called MyHashSet. **One quick clarification, if I use the term "size" we are referring to the number of elements physically inserted into the hashSet. If I say capacity, I am referring to the actual length of the table (or slots in the hashSet). Also, choose one of the different open addressing methods to account for collisions! Please leave a class comment telling me which one you used.**

As you might guess, we will use a hash table to store the elements in the set. The "slots" in the table will be implemented as linked lists.

Please take a look at the other document on myMCPS to fnd out which methods to write. This comes directly from a UMD implementation of this project, so it will give you great exposure to what you would do in college.

The MyHashSet class will implement the Iterable interface, make sure you take a look back at the original HomeDepot exercise to see what it should look like. Your Iterator will allow the user to iterate over all of the elements stored in the table.

Hashing

You can assume the following equation is being used in order to determine which slot to place the element in:
bucket number = | (p * hashCode) % n |
where p is a large prime of your choosing, and n is the capacity (length) of the table. **Don't change the prime at any point!**

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.

Your Iterator must implement all three of the methods in the interface: next, hasNext, and also remove. We suggest implementing all of the other features of the MyHashSet class before you tackle the Iterator. You can get most of the points for this project even if your iterator doesn't work, but we are hoping that you will succeed with it, too!

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.

Your "remove" method should throw an IllegalStateException in two cases:

1. If someone tries to call it before they have called "next".
2. If someone calls it twice without a call to "next" in between.

****Write a driver class and insert some Strings into your object to make sure it works correctly!****