CMSC 132

Sections 020X/030X

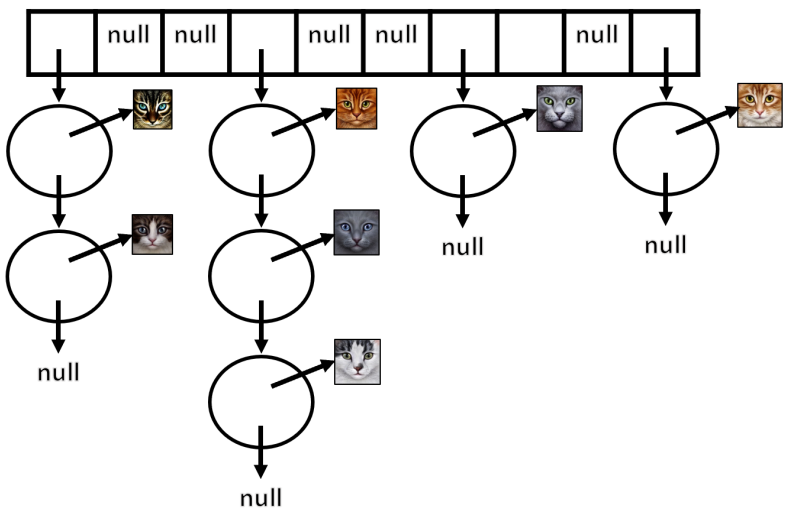
DEPARTMENT OF COMPUTER SCIENCE

Assignment: Project #5

Due Date: Sunday 04/03, 11:00PM

Open/Closed policy: CLOSED

Hash Table



Introduction

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

The picture above depicts a set of 7 cats, stored in the kind of hash table that we will be creating. Note that although the picture contains what looks like an array, we're actually going to use an ArrayList, because generics with arrays are a serious pain. Notice that the elements of the ArrayList are references to nodes, and are sometimes null. (An empty table will be an ArrayList of a certain size, containing all null entries.)

Download this zip file, which contains the starter code, and import it into Eclipse, as usual.

Before you read any further, look over the starter code that has been provided for the MyHashSet class. Notice that we are using the type variable E throughout. (E for "element"). You'll find a Node class nested inside the MyHashSet class. As you would expect, each Node has a data element (the cats in the picture, above) and 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 specifications (including hints) are contained in comments embeded within the source code distribution. Below are some additional remarks that are important for you to understand before you implement this project.

Size vs. Capacity

In this specification, 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). For example, the hash table pictured above is size 7, capacity 10. Make sure you understand this before reading further.

Hashing

You may assume that objects of type E to be inserted into the MyHashSet will satisfy Java's "HashCode Contract". You should use the following simple formula to determine which bucket a particular object belongs in, based on that object's hash code. (The object's hash code is obtained by simply calling the hashCode() method on the object. Writing a hashing function is not your job; that was handled by whomever implemented the objects of type E, whatever they are.)

```
bucket number = | hashCode % n |
```

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.

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.
- You may not store the data in two separate structures at any time. In other words, you may not create another array or Collection to temporarily hold all of the data in order to make the programming easier. The only exception to this is in the add() method in the case where the load factor has gotten too large, and you need to increase the size of the table: While you are re-hashing the data into the larger table, you will have the smaller table and the new, larger table in memory at the same time. After re-hashing the data, the smaller table should become garbage.
  If you fail to adhere to this rule, your grade on the project could be as low as 0.
- When re-sizing the table, you should double it. Don't mutiply the size by some value other than 2, or you will fail many of the tests.
- 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.
- Avoid redundant code as much as you can.
- Do not modify the DEFAULT_INITIAL_CAPACTIY constant, or the LOAD_FACTOR constant.
- Be sure to remove all of the comments that serve as "instructions to CMSC132 students" before submitting.

Grading

- The public tests will be worth 36% of your grade.
- The release tests will be worth 54% of your grade.
- Style will be worth 10%
- Note that there are no secret tests this time.
- Keep in mind that we will be inspecting your code to verify that you followed the rules above. If you didn't, you will be penalized accordingly. It is possible to receive a 0 on a fully functional project if the implementation violates the specification in a way that trivializes the assignment.

Submission

Submit your project using the "Submit Project" option (available by right clicking on the project folder).

Academic Integrity

Please make sure you read the academic integrity section of the syllabus so you understand what is permissible in our programming projects. We want to remind you that we check your project against other students' projects and any case of academic dishonesty will be referred to the Office of Student Conduct