Motivation: A Rite of Passage

We have used hash-based data structures throughout this class on various programming assignments, and we have proven certain properties about them in lecture and recitation. The time has now come to embark on a rite of passage that every budding computer scientist must take. It is time to implement a hash table.

But seriously, too many people go around blissfully using this magical all-operations-expected-\(O(1)\) data structure unaware of how it works, how to get the most out of it, and when something else might be better. In this homework, to ensure you aren’t that person, you will implement a production-grade hash map which will conform to the Java Map interface. We provide a BaseAbstractMap which your implementation should extend from, which reduces the work for a few complex Map methods. Before you get started, take a look at the java.util.Map interface to familiarize yourself with the API.

Part One: HashMap (40 points)

Files to submit: HashMap.java, BaseAbstractMapFactoryImpl.java, HashMapTest.java

Recall the method of chaining, discussed in lecture and in recitation. A hash table first hashes an Object’s hash code into a bucket index appropriate for the length of the backing array. Each bucket is a linked list of map entries that can be added to, modified in place, or removed from.

We have provided a fair bit of skeleton code for you, namely the constructors, the table buckets, and the hashing methods. You need to worry about the unimplemented method stubs such as:
1. get(Object key)
2. containsKey(Object key)
3. put(K key, V value)
4. resize(int newCapacity)
5. remove(Object key)
6. containsValue(Object value)
7. clear()
8. entryIterator()

Each method stub contains further instructions. It is critical that you read both the Javadoc comments and the implementation comments for each method. They contain necessary information on both the external and internal behavior of these methods.

It is also critical that you understand the provided code and methods. You will want to pay special attention to the threshold and loadFactor variables, the hash(int h, int length) method, and the Entry inner class. Your solution will explicitly invoke these entities.

The trickiest part of this implementation will be correctly handling null keys and null values. Hash maps can support both null keys and null values. Be careful to explicitly handle and test those, and don’t be afraid to repeat some code if you want to explicitly isolate the null cases.

**Part Two: Runtime**

No files to submit.

We have provided a simple test program in the stub to benchmark the actual measured runtime for your HashMap on various data inputs. This is done by following the Maven setup instructions and setting up an Eclipse Run Configuration to run the `edu.upenn.cis121.hw6.TestHarness` class with the parameter provided being the class name for your `BaseAbstractMapFactory` implementation.

If you’re using the default provided stub, this will be the string: `edu.upenn.cis121.hw6.BaseAbstractMapFactoryImpl`. 
Style & Tests (10 points)

This homework is worth a total of 50 points. There will only be 10 points assigned for code style, documentation and sensible tests instead of the usual 15 points. Style is worth 5 of the 10 points, and you will be graded according to the 121 style guide.

Also, please refer to the Java testing guide on the course website.