Project #8
Due: Monday 5/09 at 11:00 pm
Type of project: Closed

CMSC 132
Object-Oriented Programming II
Spring 2022

Heavy Bag

Introduction

"Bag" is an abstract data type that is like a set, but allows duplication of elements. By "Heavy" Bag we mean a bag that is capable of *efficiently* handling cases where some of the elements are repeated a huge number of times. For example, if the bag contains 100,000,000 copies of a particular item then you would *not* want to store 100,000,000 references to the item. Think about how you could store information about the number of copies of each element that are in the bag without physically storing multiple copies.

What You Will Do

Begin by downloading the project zip file and importing it into Eclipse, as usual.

You will write a class called HeavyBag, which will extend the AbstractCollection class. We will be employing Java generics, so your HeavyBag can be used to store any type of objects the user desires. The HeavyBag must support the "iterator" method, and so you will also be writing an inner class implementing an iterator over the HeavyBag.

Note: You must implement the remove method for the Iterator this time!

Data Structure?

You get to decide how to store the data. Make a good choice, or you will fail tests! When the user adds elements to the collection, you must record how many of each element have been added, without actually storing multiple copies.

You are free to make use of ANY classes you wish from the Java Collections framework to store the data in your HeavyBag. You do not have to do anything difficult like implementing a data structure from scratch -- just use one (or more) of the existing Java collection classes that you've learned about this semester. Note that you are not allowed to use Java Collection classes that were not covered in the 132 lectures.

Implementation Details

In a Bag, removing an item removes a single instance of the item. For example, a Bag b could contain additional instances of the String "a" even after calling b.remove("a").

The iterator for a heavy bag must iterate over all instances, including duplicates. In other words, if a bag contains 5 instances of the String "a", an iterator will generate the String "a" 5 times. **You must implement the remove() method for the iterator.** It should remove a single instance of the last element that was returned by a call to next().

In addition to the methods defined in the Collection interface, the HeavyBag class supports several additional methods: uniqueElements(), getCount(), addMany(), and choose().

The class extends AbstractCollection in order to get implementations of addAll(), removeAll(), retainAll() and containsAll(). (We will not be over-riding those). All other methods defined in the Collection interface will be implemented here.

Be sure to read over the entire API before you start writing code. Some of the methods you write may make calls to other methods that you are writing, and it is possible that one method may depend on another one that is further down in the source code file.

For more specific implementation details, please read the javadoc.

Implementing equals() with Generics

Generics are for the compiler -- they're not a runtime feature. Unfortunately, this makes it tricky to implement the equals() method with generics, since equals() needs to check types at runtime. Here's a hint:

Intead of typing "HeavyBag<T>" in your equals() method, use "HeavyBag<?>". That will make the compiler happy, and won't generate warnings or errors.

Regarding HashCodes

Since we need to override the equals() method (the user must have a way of comparing two HeavyBags for equality) we absolutely must override hashCode() as well, since failure to do so would violate the Java HashCode Contract. As you may recall, it is desireable for the hashing function to only make use of the immutable state of the object. Unfortunately, in this case, all of the state is mutable! We have no choice but to use some (or all) of the mutable state in generating the hashCode for our objects.

Using mutable state as part of the hashCode computation makes things difficult for the user of our class. Suppose the user decides to put HeavyBags into any kind of hash table (including HashSet, LinkedHashSet, HashMap, LinkedHashMap). Now every time an element that is already in the table needs to be modified in any way (adding to the HeavyBag or removing from the HeavBag), then the following steps must be taken:

1. The element must be removed from the table first
2. The element can now be modified
3. The element must be rehashed (re-added) to the table

What a pain! Unfortunately, it is unavoidable in this case.

Grading

The public tests, release tests, and secret tests together will comprise 100% of your grade on the project. There will not be a "style" grade, but we will be inspecting your code to make sure that you have followed our instructions and haven't violated anything that we've specfied. You could lose a lot of points (possibly all of them) if you implement something that passes the tests, but doesn't follow our instructions.