CS 310 Data Structures
Programming Assignment 2
Spring 2021

Suggested Steps, more discussion.

Part 1. Setting up

a. Transfer the pa2setup directory system (zip file provided) to your development area. T". To do this, first download the provided zip file by clinking on the link in your browser--this will put pa2setup.zip in your Downloads directory under your home directory, but you don't need to access it that way. Just Open the downloaded zip with the browser and agree to expanding it, and then choose the target to be a pa2 directory next to the top-level directory you used for pa1, possibly c:/cs310/pa2 on Windows or ~/cs310/pa2 on Mac or Linux.
b. Display the directories of pa2 from the command line by using tree on Windows or du on Mac or Linux. Also look at them using Windows File Explorer or Mac Finder. Create a project using eclipse's Open Project from File System or similar capability in your favorite IDE, if any. If not using an IDE, you can ignore the bin file tree.
c. Find the line in HashMap1.java that says that it extends AbstractMap. This is an important fact you need to know. This means that if you accidentally comment out get in your own HashMap, the client code will just use AbstractMap's get instead! Find get in AbstractMap to bring this point home.
d. Run TestMap: it comes ready to run with HashMap1.java, a working HashMap source, as well as JDK HashMap and TreeMap. See the assignment for the needed commands.

Part 2. **Encapsulating the HashMap**

a. Copy HashMap1.java to HashMap2.java.
b. Look at the Map API to see what methods should remain public. I would find it by Googling "JDK Map Javadoc" and picking out the Java 8 version, but another way would be to use the link to all the JDK v8 docs on the class web page and then selecting Map.
c. Look at the methods in HashMap2.java and mark the ones not in the API as private.
d. Now find the inner classes in HashMap2 and mark them private.
e. Run TestMap to check you haven't broken HashMap2. First edit TestMap.java to use HashMap2.

Part 3. **Using ArrayList<Node> for collision lists.**

a. Copy HashMap2.java to HashMap3.java.
b. Create the new instance variable of HashMap3 named table1, of type array of ArrayList<Node> for the new hash table array. Put it right after the old HT variable, table, which we are keeping around so that entrySet still works for us.
c. Find HashMap3's no-args constructor. "no-args" means no arguments. This is where to initialize the array of ArrayLists we want to use as the hash table. Note that the no-args constructor is the only HashMap constructor we'll use for testing here.
d. In the no-args constructor, create the new array of ArrayList<Node> for your new instance variable table1, of size 16. Then create empty ArrayList<Node> in all the array entries. Now you have a healthy but empty HT in table1.
e. Study the hash() functions on pg. 465 and in HashMap1.java and see that one in the book ends up with a value that can be used as a bucket number but the HashMap one does not, instead ending up with a 32-bit hashed value "hash" that needs further processing to be used to determine a bucket number. That further processing is `(n-1)&hash` where n is a power of 2. Read the "Important Note" in the assignment about this calculation.
f. Replace put() with code that finds the bucket number the same way as before, but now that bucket has a ArrayList<Node> to hold its collision list. You need to write code to scan that list to try to find the key, and if you find it, replace its value with the value coming in via put's parameter. And also pick up the old

value for return from the method. If the list-scan doesn't find the key, then you need to add the key-value pair to the list in a new Node. In that case return null from the method. Test with TestMap.

g. Replace get() accordingly.

Part 4. **Getting resize working.**

First, copy HashMap3.java to HashMap4.java.

One way to go:

a. "You can build a new HT twice as big as before using code like that in the constructor, and then move all the key-value pairs from old to new HT." For this approach, note that this new HT array does not need an instance variable, because it lives and dies in one method. Just use a local variable typed like table1, and created like the no-args constructor code.
b. To find all the key-value pairs, set up a loop over the buckets, and within that loop, a loop over the collision list's Nodes, in which lie the key-value pairs. Process each key-value pair using code like you have in put() already, but using this new temporary array.
c. Finally, copy the local array reference into table1, replacing the whole array. The old array is no longer referenced from"live" variables, so will be garbage-collected.

Other way to go. In the above description, note that we have to copy put's code and use it with a different array. Copying code is a minor code crime. How can we avoid it? By creating a temporary HashMap4 and using the real put() to load it.This is the same idea as used in S&W's resize method on page474.

a. This temporary HashMap4 can be built in a local variable, loaded, and stranded, so garbage-collected.
b. To find all the key-value pairs, set up a loop over the buckets, and within that loop, a loop over the collision list's Nodes, in which lie the key-value pairs. Process each key-value pair by calling put() on the temporary HashMap4.
c. Finally, copy the temporary HashMap4's table1 reference into table1, the current HashMap's HT. As your method returns, the local variable for the reference to the temporary HashMap4 will disappear, so the whole temporary HashMap4 will be garbage-collected. You might be worried that you don't have access to the temporary HashMap4's table1 because of encapsulation, but it's the same class, so available.