Homework 3: Hashing

CS 1332 Section C

Fall 2021

1 Important

There are general homework guidelines you must always follow. If you fail to follow any of the following guidelines, you risk receiving a **0** for the entire assignment

1. All submitted code must compile under **JDK 11**. This includes unused code, so don't submit extra files that don't compile. Any compile errors will result in a 0.
2. Do not include any package declarations in your classes.
3. Do not change any existing class headers, constructors, instance/global variables, or method signatures. For example, do not add **throws** to the method headers since they are not necessary.
4. Do not add additional public methods. You may create helper methods, but any helper method you create should be **private**.
5. Do not use anything that would trivialize the assignment. (e.g. don't import/use java.util.ArrayList for an ArrayList assignment. Ask if you are unsure.)
6. Always consider the efficiency of your code. Even if your method is *O(n)*, traversing the structure multiple times is considered inefficient unless that is absolutely required (and that case is very rare).
7. You must submit your source code - the .java files. Do not submit compiled code - the .class files.
8. Only the last submission will be graded. Make sure your last submission has all required files. Resubmitting voids prior submissions.

2 Style and Formatting

It is important that your code is not only functional, but written clearly and with good programming style. Your code will be checked against a style checker. The style checker is provided to you and it located on Canvas. A point is deducted for every style error that occurs. Please double check before you submit that your code is in the appropriate style so that you don't lose any unnecessary points!

2.1 Javadocs

Javadocs should be written for any private helper methods that you create. They should follow a style similar to the existing javadocs on the assignment. Any javadocs you write must be useful and describe the contract, parameters, and return value of the method. Random or useless javadocs added only to appease checkstyle will lose points.

2.2 Vulgar/Obscene Language

Any submission that contains profanity, vulgar, or obscene language will receive an automatic zero on the assignment. This policy applies to all aspects of your code, such as comments, variable names, and javadocs.

2.3 Exceptions

When throwing exceptions, you must include a message by passing in a String as a parameter. **The message must be useful and tell the user what went wrong.** "Error", "Oof - Bad things are happening", and "FAIL" are *not* good messages. Additionally, the name of an exception itself is not a good message.

In addition, you may not use try catch blocks to catch an excpetion unless you are catching an exception you have explicitly thrown yourself with the **throw new ExceptionName('Exception Message");** syntax

2.4 Generics

If available, use the generic type of the class; do not use the raw type of the class. For example, use new **LinkedList<Integer>()** instead of **new LinkedList()**. Using the raw type of the class will result in a penalty.

3 Forbidden Statements

You may not use any of the following in your code at any time in CS 1332. If you are not sure whether you can use something, and it is not explicitly listed here, just ask. Debug print statements are fine, but should be either removed or commented out prior to submission. If print statements are left in, assignments are messy to grade, and checkstyle points will be deducted.

- package
- System.arraycopy()
- clone()
- assert()
- Arrays class
- Thread class
- Collections class
- Collection.toArray()
- Refleciton APIs
- Inner or nested classes
- Lambda Expressions
- Method References (using the :: operator to obtain a reference to a method)

4 JUnits

We have provided a **very basic** set of tests for your code. These tests do not guarantee the correctness of your code, nor do they guarantee you any grade. You may additionally post your own set of tests for others to use on the Georgia Tech GitHub as a gist. Do **NOT** post your tests on the public GitHub.

5 Deliverables

You must submit all of the following file(s) to the corresponding assignment on Gradescope. Make sure all file(s) listed below are in each submission, as only the last submission is graded. Make sure the filename(s) matches the filename(s) below, and that *only* the following file(s) are present.

1. HashMap.java

5.1 Hash Map

You are to code an **HashMap**, a key-value hash map with an **external chaining** collision resolution strategy. A HashMap maps unique keys to values and allows O(1) average case lookup of a value when the key is known. The table should **not** contain duplicate keys, but **can** contain duplicate values. In the event of trying to add a duplicate key, replace the value in the existing (key, value) pair with the new value and return the old value. You should implement two constructors for this HashMap. As per the javadocs, you should use constructor chaining to implement the no-arg constructor.

5.1.1 Capacity

The starting capacity of the HashMap using the default constructor should be INITIAL_CAPACITY defined in HashMap.java. Reference the constant as-is. Do not simply copy the value of the constant. Do not change the constant. Do not regrow the backing array when removing elements.

If adding to the table would cause the load factor (LF) to exceed (greater than, not greater than or equal to) the max load factor constant provided in the java file, the table should be resized to have a capacity of 2n + 1, where n is the current capacity before adding the parameterized element. See the javadocs for specific instructions on when to resize. There is a method called resizeBackingTable that you should use for resizing.

5.1.2 Hash and Compression Functions

You should **not** write your own hash functions for this assignment. Instead, use the hashCode() method that every Object has. For the compression function, mod by table length first, then take the absolute value (it must be done in this order to prevent overflow in certain cases). As a reminder, you should be using the hashCode() method on **only the keys** (and not the MapEntry object itself) since that's what is used to look up the values. After converting a key to an integer with a hash function, it must be compressed to fit in the array backing the HashMap.

5.1.3 External Chaining

Your hash map must implement an external chaining collision policy. That is, in the event of a collision, colliding entries are stored as a chain of MapEntry objects at that index. As such, if you need to search for a key, you'll need to traverse the entire chain at the hashed index to look for it. See MapEntry.java to see what is stored and what methods are available for use; **do not use Java's LinkedList to handle the chaining functionality.**

5.1.4 Adding

When adding a key/value pair to a hash map, add the pair to the front of the chain in the correct position. Also remember that keys are unique in a hash map, so you must ensure that duplicate keys are not added. Each index of the table should point to an MapEntry that represents the head of a linked list. That linked list contains all elements that collide at that index.

5.1.5 Removing

When removing a key/value pair from a hash map using external chaining, you can safely remove the item unlike in open addressing techniques such as linear probing where you must use a DEL marker. Removing from a chain is very similar to removing from a Singly-Linked List, treating the first table entry as the head, so refer to your notes and homework assignments from earlier in the course if you need a refresher. As usual, if the entry you are removing is the only entry at that index, you should make sure to null out that spot rather than leaving it there.

6 Grading

Here is the grading breakdown for the assignment. There are various deductions not listed that are incurred when breaking the rules listed in the PDF and in other various circumstances. Note that for this assignment efficiency is explicitly mentioned as contributing to your point total. In contrast to past assignments, unimplemented methods will result in losing efficiency points.

- HashMap − 80pts
- Efficiency − 10pts
- Checkstyle − 10 pts
- Total: 100 pts