Arizona State University
Lect. Ruben Acuña

SER222: Data Structures & Algorithms
Revised 4/16/2016 [Spring] [Concurrent]

Unit 12: Programming
Implementing an Hash-based Maps

Summary: In this assignment, you will implement two versions of a map ADT.

1 Background

In this assignment, will practice applying our knowledge of ADTs, maps, and hashing, to implement a pair of map ADTs which use relatively standard hashing techniques. Most likely, these will be the most conceptually complicated data types you have built. Provided for you is ChainMap: a sample implementation of a chaining approach to implementing a hash-based map. This file is similar to what is seen in the slides, but is more comprehensive and follows a Map interface. We will be implementing two techniques for hash-based maps:

TwoProbeChainMap: This technique is similar to the list chaining method that is covered in the slides and Appendix E. In a normal chaining approach, keys hash to exactly one index and the key/value pair must reside in the list at that particular index. In this new approach, we will instead calculate two hashes, that indicate two different indices, and then add the new key/value to whichever of two lists, at the two indices, is the shortest. The result is that the chains (lists) will end up being shorter in general since we split in half where the key/value pairs are placed.

LinearProbingMap: This technique is covered in the text and slides under open addressing. The idea is that we hash to a specific index in the internal array. If the index is empty, we add the key/value pair to it. If occupied, we increment the index by 1 and try again. This repeats until an empty index is found.

Sample UML for these classes is shown in Figure 1. This is provided only as a hint. Your requirement is to follow the interface specification, not the UML.

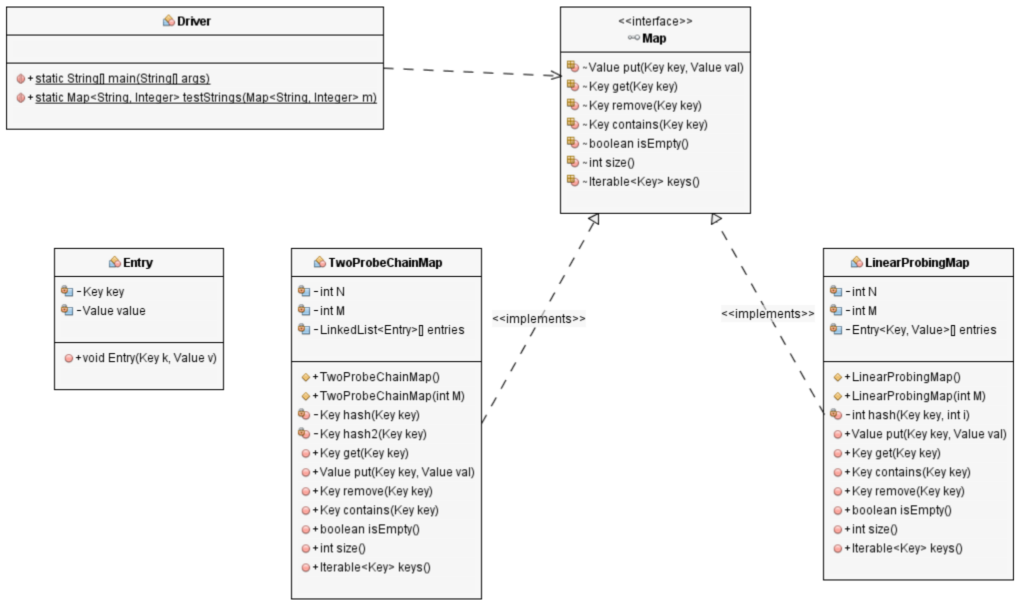


Figure 1: Sample Map Implementations UML

This document is separated into four sections: Background, Requirements, Testing, and Submission. You have almost finished reading the Background section already. In Requirements, we will discuss what is expected of you in this homework. In Testing, we give some basic suggestions on how the map implementations can be tested. Lastly, Submission discusses how your source code should be submitted on BlackBoard.

2 Requirements [80 points, 10 extra credit]

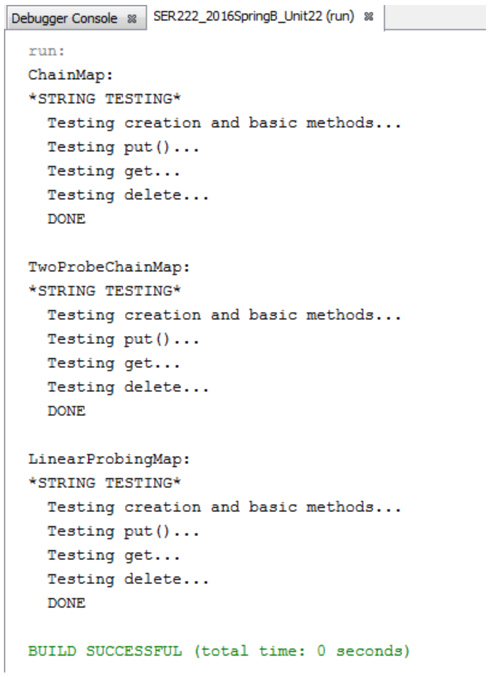
In this assignment you will implement two types of hash-based maps. Download the attached Driver.java, Map.java. The first file defines some tests for maps, the second is the map interface. Also included is ChainMap.java, which contains a sample implementation of a chaining hash-based map.

- Write a class called TwoProbeChainMap that implements a two-probe chaining hash-based map. Two-probe hashing means that you will hash to two positions, and insert the key in the shorter of the two chains at those two positions. The easiest way to implement this class is to spend some time understanding ChainMap, and then redesign it to use the two probe technique. [35 points total]
  - Proper hash calculations. [5 points]
    - For the first hash function, use: hash(k)=(k.hashCode() & 0x7fffffff) % M.
    - For the second hash function, use: hash2(k)= (((k.hashCode() & 0x7fffffff) % M) * 31) % M.
  - void put(Key key, Value val) - see interface. [10 points]
  - Value get(Key key) - see interface. [10 points]
  - void remove(Key key) - see interface. [10 points]
- Write a class called LinearProbingMap that implements a linear probe hash-based map. [45 points total]
  - Proper hash calculations. [5 points]
    - An decent way to structure the hash function is: hash(k, i) = ((k.hashCode() & 0x7fffffff) + i) % M, where k is the key and i is the number of collisions. Each time there is a collision, i should be incremented so that the hash increases by 1. An example hash sequence might look like: 587, 588, 589, 590, 581...
  - LinearProbingMap() - a constructor that defaults to an array of size 997 . [3 points]
  - void put(Key key, Value val) - see interface. [10 points]
  - Value get(Key key) - see interface. [10 points]
  - void remove(Key key) - see interface. [10 points extra credit]
  - boolean contains(Key key) - see interface. [3 points]
  - boolean isEmpty() - see interface. [3 points]
  - int size() - see interface. [3 points]
  - Iterable<Key> keys() - see interface. [3 points]
  - There is no requirement to support array resizing.
- Both classes must implement the provided Map interface.
- Do not import any packages other than Queue or LinkedList in your map implementations.
- No testing report is required this time - test as much or as little as it takes to convince yourself that the implementations work.

3 Testing

The provided driver file already contains several tests for the operations in the interface. Out of the box, the ChainMap implementation should pass these tests. Note that the tests require assertions to be enabled. The tests are designed to check not only the interface operations in isolation, but how they interact with each other. For example, they check that certain properties of the map are invariant over the operations. An invariant is something that does not change. Like the size of the ADT before and after checking if an element is contained. Figure 2 shows the expected output from the driver, once the two classes (TwoProbeChainMap, and LinearProbingMap) have been implemented. Initially, the driver won't compile since it depends on those two classes. Be aware that while a considerable number of operations are tested, what is provided is not comprehensive. For example, all tests use the String data type and do not check how well the ADT functions with other types (e.g., Integer, Double, a custom class, etc).

For this assignment, there is no testing write up required. However, you may want to do your own testing to verify the completeness of your implementation.



```
run:
ChainMap:
*STRING TESTING*
  Testing creation and basic methods...
  Testing put()...
  Testing get...
  Testing delete...
  DONE

TwoProbeChainMap:
*STRING TESTING*
  Testing creation and basic methods...
  Testing put()...
  Testing get...
  Testing delete...
  DONE

LinearProbingMap:
*STRING TESTING*
  Testing creation and basic methods...
  Testing put()...
  Testing get...
  Testing delete...
  DONE

BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 2: Tree UML Overview

4 Submission

The submission for this assignment has only one part: a source code submission. It should be uploaded to the submission link on BlackBoard.

Writeup: Not required.

Source Code: Please zip your source code files together as "LastNameMap.zip" (e.g. "AcunaMap.zip"). It should contain only two files: TwoProbeChainMap.java, and LinearProbingMap.java. Be sure that you use the correct compression format - if you do not use the right format, we may be unable to your submission. Do not include your project files, or leave code in your files that is specific to your IDE/project.