T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY of ENGINEERING
DEPARTMENT of COMPUTER ENGINEERING

DIFFERENT IMPLEMENTATIONS OF KWHASHMAP

CSE 222 DATA STRUCTURES AND ALGORITHMS
HOMEWORK 5 - PART 2 REPORT

STUDENT
Şeyda Nur DEMİR
12 10 44 042

LECTURER
Prof. Dr. Fatih Erdoğan SEVİLGEN

TEACHING ASSISTANT
Başak KARAKAŞ
Mehmet Burak KOCA

KOCAELİ, 2021

1.DESCRIPTION

This homework has 2 parts.

Part 1 : 40 pts

Write a custom iterator class MapIterator to iterate through the keys in a HashMap data structure in Java. This class should have the following methods:

- next(): The function returns the next key in the Map. It returns the first key when there is no not-iterated key in the Map.
- prev(): The iterator points to the previous key in the Map. It returns the last key when the iterator is at the first key.
- hasNext(): The method returns True if there are still not-iterated key/s in the Map, otherwise returns False.
- MapIterator (K key): The iterator should start from the given key and still iterate though all the keys in the Map. The iterator starts from any key in the Map when the starting key is not in the Map or not specified (zero parameter constructor).

Part 2 : 60 pts

Implement KWHashMap interface in the book using the following hashing methods to organize hash table:

- Use the chaining technique for hashing by using linked lists (available in the text book) to chain items on the same table slot.
- Use the chaining technique for hashing by using TreeSet (instead of linked list) to chain items on the same table slot.
- Use the Coalesced hashing technique. This technique uses the concept of Open Addressing to find first empty place for colliding element by using the quadratic probing and the concept of Separate Chaining to link the colliding elements to each other through pointers (indices in the table). The deletion of a key is performed by linking its next entry to the entry that points the deleted key by replacing deleted entry by the next entry. See the following illustration as an example:

Input = {3, 12, 13, 25, 23, 51, 42}
Hash Function = (data % 10)

Test all the three hash table implementations empirically. Use small, medium, and large-sized data and hash tables in suitable sizes for testing. Perform different tasks over the tables to compare their performance results (like accessing existing/non-existing items or adding/removing items).

Restrictions

- Can be only one main class in project
- Don’t use any other third part library

General Rules

- For any question firstly use course news forum in Moodle, and then the contact TA.
- You can submit assignment one day late and will be evaluated over sixty percent (%60).

Technical Rules

- You must write a driver function that demonstrates all possible actions in your homework. For example, if you are asked to implement an array list and perform an iterative search on the list then. The driver function should run when the code file is executed.
- Implement clean code standards in your code ;
  - Classes, methods and variables names must be meaningful and related with the functionality.
  - Your functions and classes must be simple, general, reusable and focus on one topic.
  - Use standard java code name conventions.

Report Rules

- Add all javadoc documentations for classes, methods, variables …etc. All explanation must be meaningful and understandable.
- You should submit your homework code, Javadoc and report to Moodle in a “studentid_hw5.tar.gz” file.
- Use the given homework format including selected parts from the table below:
  - Problem solutions approach
  - Test cases
  - Running command and results

Grading

- No OOP design: -100
- No interface: -95
- No method overriding: -95
- No error handling: -50
- No inheritance: -95
- No polymorphism: -95
- No javadoc documentation: -50
- No report: -90
- Disobey restrictions: -100
- Cheating: -200
- Your solution is evaluated over 100 as your performance.

2.REPORT

I detailed here what I did in my homework Part 2.

2.1.Problem Solutions Approach

Note : I googled the way you said in the report to problem solving approach, but I could not any useful article, what I found was either paid or long. After all I found a useful post from medium. I prepared this part according to this post. I hope I got it right.

- **Clearly understanding and/or defining the problem :**
  I understood and defined the problem clearly.
    - We should implement KWHashMap interface by three different way
    - First, we use chaining technique, and use linked list to store data
    - Second, we use chaining technique, and use tree set to store data
    - Third, we use open addressing technique, and design as described, and I used array of entries to store the data
- **Breaking down large problems into smaller problems :**
  I broke down large problem KWHashMap interface implementations,
  to small problems HashingUsingLinkedList, HashingUsingTreeSet, HashingUsingEntryTable classes, and also a Test and a Main class.
    - We have a large problem, KWHashMap implementations.
    - I have now small problems, five basic class implementations.
- **Solving the problem at an abstract level first :**
  I solved the problem at an abstract level first.
    - I thought a lot about the problem.
    - I scribbled something about this subject in the ledger.
    - Something started to take shape in my head.
    - I used my knowledge of data structures, and it is.
- **Using notes and pseudo-code :**
  I noted what I thought, and wrote permanently pseudo-codes mixed with java codes.
    - If i understand or find something new, I noted.
    - I wrote pseudo-codes mixed with java codes, not clear.
- **Running code early and often :**
  I wrote real codes and run them often.
    - I turned my pseudo-codes into real java codes.
    - I coded them in ide and run them often.

2.2.Test Cases

Testing Requirements

Test Case	**Pres / Posts**	**Done**
Hashing with different implementations of KWHashMap	• Implement KWHashMap interface by three different way • Construct three objects of these implementations • Put, get and remove with different numbers of entries to/from these map objects • Calculate all operation’s total execution time and print all steps • Print which implementation has the best performance	Done
Hashing with given input data	• Construct an ArrayList object • Add given input datas to ArrayList • Construct EntryTable implementation object • Put entries to the map • Remove entry from the map • Print all steps of the table	Done

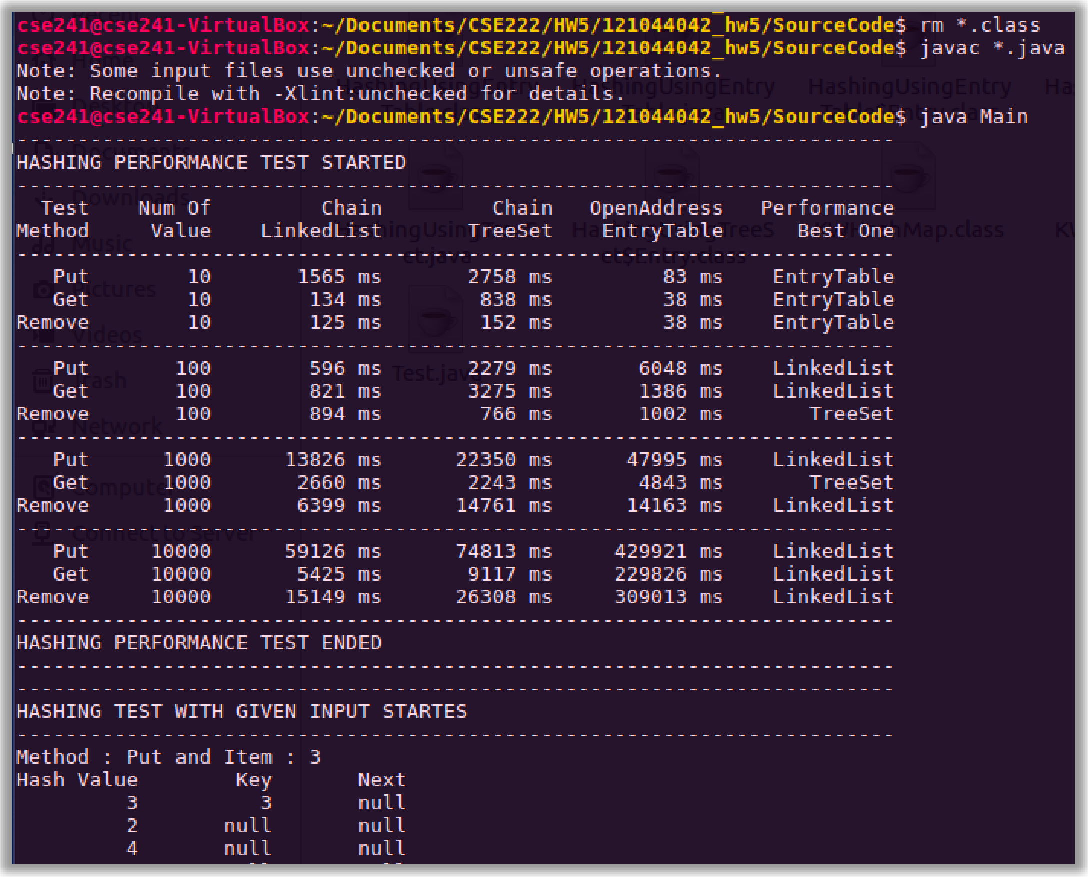
2.3.Running Commands and Results

Compile and Run Commands, Testing Steps

Usage of my program :

1. Compile program with “**javac *.java**” command
2. Run program with “**java Main**” command

Simply Follow This Screenshot



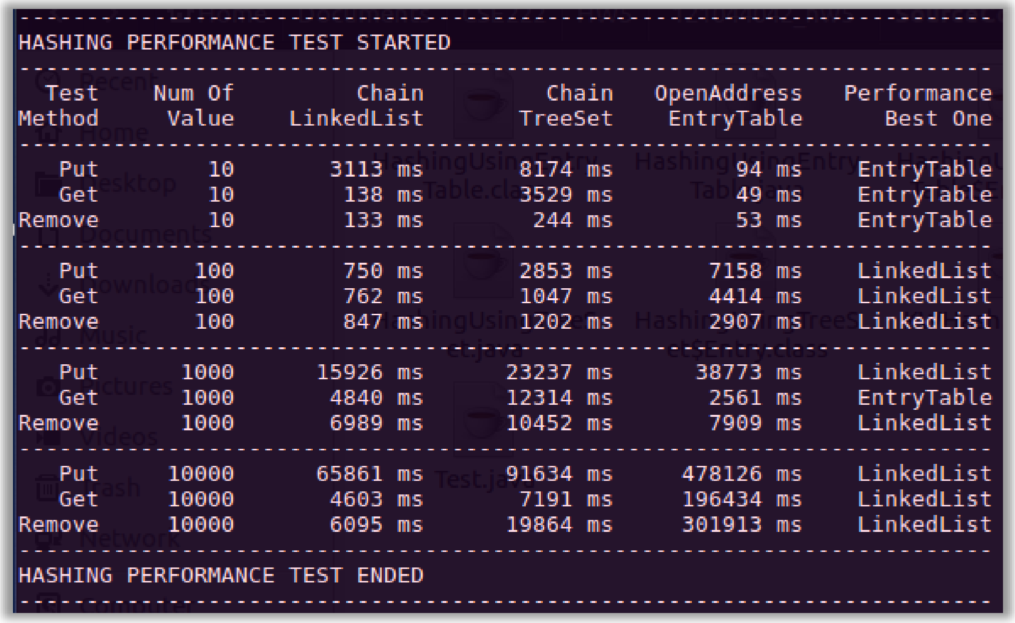
```
cse241@cse241-VirtualBox:~/Documents/CSE222/HW5/121044042_hw5/SourceCode$ rm *.class
cse241@cse241-VirtualBox:~/Documents/CSE222/HW5/121044042_hw5/SourceCode$ javac *.java
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
cse241@cse241-VirtualBox:~/Documents/CSE222/HW5/121044042_hw5/SourceCode$ java Main
-------------------------------------------------------------------------
HASHING PERFORMANCE TEST STARTED
-------------------------------------------------------------------------
  Test    Num Of         Chain         Chain  OpenAddress   Performance
Method     Value    LinkedList       TreeSet   EntryTable      Best One
-------------------------------------------------------------------------
   Put        10       1565 ms       2758 ms        83 ms    EntryTable
   Get        10        134 ms        838 ms        38 ms    EntryTable
Remove        10        125 ms        152 ms        38 ms    EntryTable
-------------------------------------------------------------------------
   Put       100        596 ms       2279 ms      6048 ms    LinkedList
   Get       100        821 ms       3275 ms      1386 ms    LinkedList
Remove       100        894 ms        766 ms      1002 ms       TreeSet
-------------------------------------------------------------------------
   Put      1000      13826 ms      22350 ms     47995 ms    LinkedList
   Get      1000       2660 ms       2243 ms      4843 ms       TreeSet
Remove      1000       6399 ms      14761 ms     14163 ms    LinkedList
-------------------------------------------------------------------------
   Put     10000      59126 ms      74813 ms    429921 ms    LinkedList
   Get     10000       5425 ms       9117 ms    229826 ms    LinkedList
Remove     10000      15149 ms      26308 ms    309013 ms    LinkedList
-------------------------------------------------------------------------
HASHING PERFORMANCE TEST ENDED
-------------------------------------------------------------------------
-------------------------------------------------------------------------
HASHING TEST WITH GIVEN INPUT STARTES
-------------------------------------------------------------------------
Method : Put and Item : 3
Hash Value          Key       Next
         3            3       null
         2         null       null
         4         null       null
```

Commands and Results with Screenshots

Hashing Performance Test

1. Program prints a table shows that
   how effects the hashing performance the used technique and used data structure to store data.
2. Puts, gets and removes with different number of entries to hashing implementations,
   and calculates execution times,
   then prints all informations and the best one of them.
3. We can see clearly which implementation we should use,
   a. As used techique (chaining / open addressing)
   b. As data structure to store the data entry table (Linked List / Tree Set / Array of Entry)
   c. To put, get and remove operations.

Test Results are Here

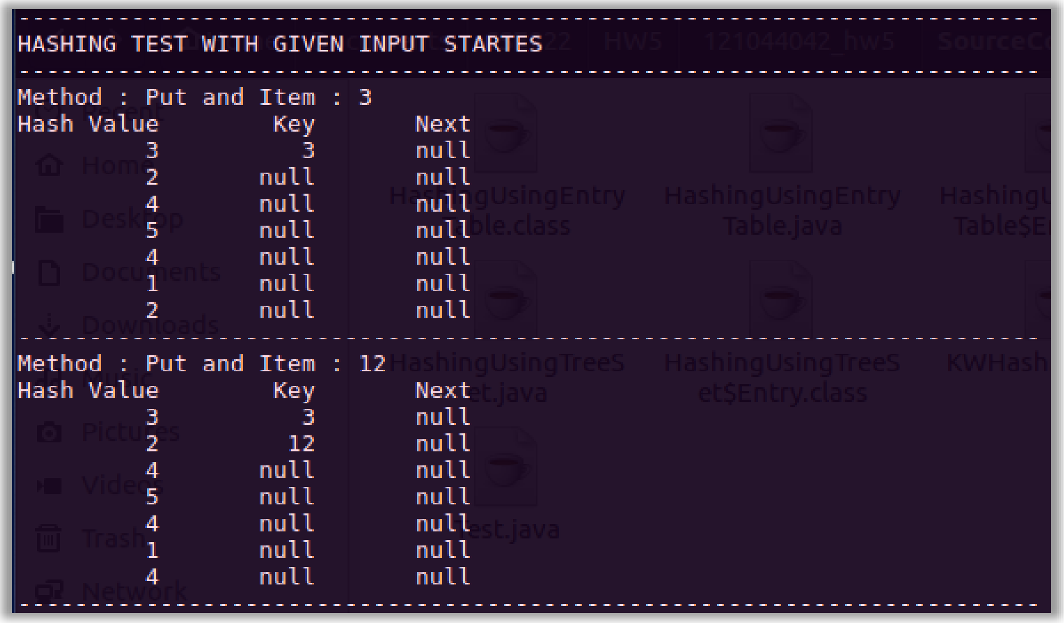


```
------------------------------------------------------------------------
HASHING PERFORMANCE TEST STARTED
------------------------------------------------------------------------
  Test    Num Of         Chain         Chain   OpenAddress   Performance
Method     Value    LinkedList       TreeSet    EntryTable      Best One
------------------------------------------------------------------------
   Put        10       3113 ms       8174 ms         94 ms    EntryTable
   Get        10        138 ms       3529 ms         49 ms    EntryTable
Remove        10        133 ms        244 ms         53 ms    EntryTable
------------------------------------------------------------------------
   Put       100        750 ms       2853 ms       7158 ms    LinkedList
   Get       100        762 ms       1047 ms       4414 ms    LinkedList
Remove       100        847 ms       1202 ms       2907 ms    LinkedList
------------------------------------------------------------------------
   Put      1000      15926 ms      23237 ms      38773 ms    LinkedList
   Get      1000       4840 ms      12314 ms       2561 ms    EntryTable
Remove      1000       6989 ms      10452 ms       7909 ms    LinkedList
------------------------------------------------------------------------
   Put     10000      65861 ms      91634 ms     478126 ms    LinkedList
   Get     10000       4603 ms       7191 ms     196434 ms    LinkedList
Remove     10000       6095 ms      19864 ms     301913 ms    LinkedList
------------------------------------------------------------------------
HASHING PERFORMANCE TEST ENDED
------------------------------------------------------------------------
```

Hashing Test With Given Input Data

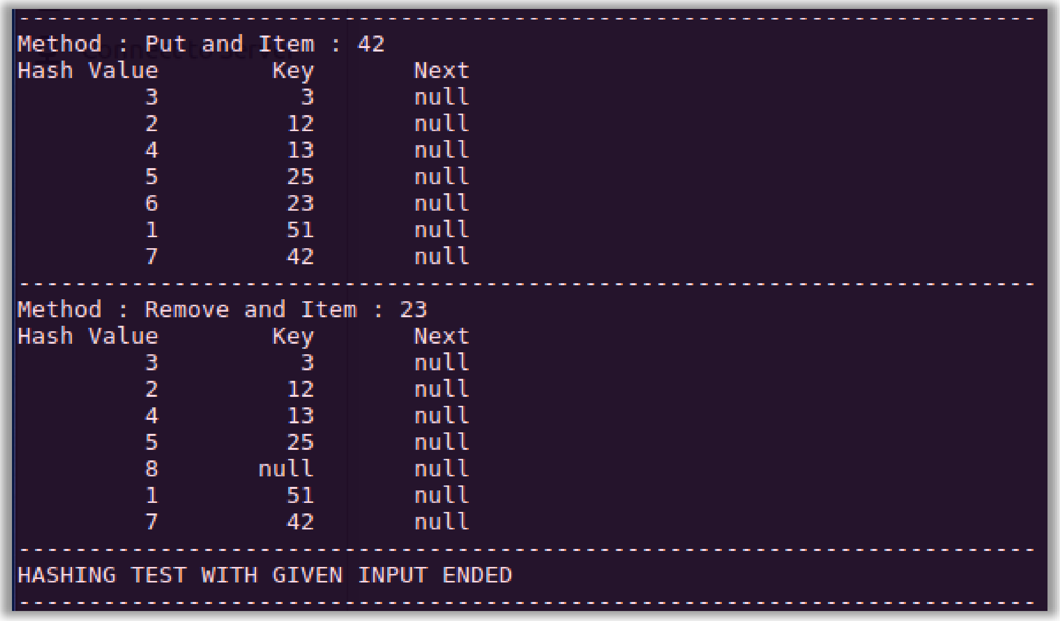
1. Program adds given input data 3, 12, 13, 25, 23, 51, 42 item by item.
2. Program uses EntryTable, last implementation of hashing interface KWHashMap, as described assignment, also uses given hash function (data % 10) to hashing values of keys.
3. Program deletes item 13.
4. Program prints step by step the table's status, and we can clearly see how it works.

Test Results are Here



```
-----------------------------------------------------------------------------
HASHING TEST WITH GIVEN INPUT STARTES
-----------------------------------------------------------------------------
Method : Put and Item : 3
Hash Value         Key         Next
         3           3         null
         2        null         null
         4        null         null
         5        null         null
         4        null         null
         1        null         null
         2        null         null
-----------------------------------------------------------------------------
Method : Put and Item : 12
Hash Value         Key         Next
         3           3         null
         2          12         null
         4        null         null
         5        null         null
         4        null         null
         1        null         null
         4        null         null
-----------------------------------------------------------------------------
```

...



```
-----------------------------------------------------------------------------
Method : Put and Item : 42
Hash Value         Key         Next
         3           3         null
         2          12         null
         4          13         null
         5          25         null
         6          23         null
         1          51         null
         7          42         null
-----------------------------------------------------------------------------
Method : Remove and Item : 23
Hash Value         Key         Next
         3           3         null
         2          12         null
         4          13         null
         5          25         null
         8        null         null
         1          51         null
         7          42         null
-----------------------------------------------------------------------------
HASHING TEST WITH GIVEN INPUT ENDED
-----------------------------------------------------------------------------
```

END OF THE REPORT

LAST UPDATE

May 12, 2021   Wednesday   23:30

STUDENT
Şeyda Nur DEMİR
12 10 44 042

LECTURER
Prof. Dr. Fatih Erdoğan SEVİLGEN

TEACHING ASSISTANT
Başak KARAKAŞ
Mehmet Burak KOCA

KOCAELİ, 2021