Homework #5 - 1801042634

Student ID	1801042634
Student Name	Muhammed
Student Surname	Oğuz

Problem solutions approach

This homework is expecting implement map and hashTable interfaces. I personally like using map in other languages.

- First of all, in part-1. I have to create a new class which `extends` hashMap class. Reason is this, we expected to implement an iterator class to hashMap. When I `extend` this class, I will have all features of this class and also will chance to add new feature such as implementing `iterator`. So I created `HashMapIterable` class to implements `HashMap` class and contains two method to call `MapIterator` iterator.
  After than, I create my `MapIterator` class and implement expected features using `HashMap.keySet().iterator()` method. This approach is reduce all my hard work and make this part very easy to me.
  A problem was implementing this is, When adding/removing an element to map, and using iterator next/prev method raise error. I wrote a method to avoid this error.
- In part-2, We expected that, using `KWHashMap` interface and implement it with 3 different way.
  Frist of all, we have to implement with `LinkedList`. This one was very easy, because book's implementation was very clear and we were allowed to use this implementation. Besides, remove method were missed and not implemented with book instructors. I implemented it with using `linkedList` iterator and it handle it for me.
- In part-2 second part, We have to implement same methods with `TreeSet`. Hopefully, That one was easy too after implementing `LinkedList`. Just changing few lines were enough for this implementation.
  A key difference was, using Key values as `Comparable` and implementing `Comparable` interface with `Entry` class

- In part-2 third and last part was tough. It was has two different method's complete approach. I do some research to make assumptions.
  After, I started to implement. In beginning, my choice of representing table was using array and using `entry` class with a node ref to hold next item. After some struggle, I changed my choice to `ArrayList` and implemented with this.
  `put` and `remove` methods were enough to implement main idea. I started to implement with `put` method. There was some important bounds, after figuring those out. Implementing were clear and I implement it.
- In `Test` class, I have to wrote all possible test values. So I try to wrote all possible using approach.

Test Cases

Part-1

In first part, First I have to created a object to hold values and insert values. After that I would be able to use my new iterator class. So I put new values and use my iterator to print to screen.

```
18    public static void part1_test() {
19        System.out.println("Testing Custom Class HashMapIterable which extended from HashMap\nContains MapIterator");
20        System.out.println("Insert following entries.");
21        System.out.println("{foo=0, bar=1, YSA=2, Erdogan Hoca=3, Burak Hoca=4, John=5, Doe=6, Peace=7, Lennon=8, Beatles=9}");
22
23        HashMapIterable<String, Integer> map = new HashMapIterable<>();
24
25        map.put("foo", 0);      map.put("bar", 1);
26        map.put("YSA", 2);      map.put("Erdogan", 3);
27        map.put("Burak", 4);    map.put("John", 5);
28        map.put("Doe", 6);      map.put("Peace", 7);
29        map.put("Lennon", 8);   map.put("Beatles", 9);
30
31        System.out.println("\nPrint map\n");
32        System.out.println(map);
33
34        System.out.println("\nCreate zero-parameter iterator -> map.iterator()");
35        System.out.println("Use while(it.hashNext()) and print map with iterator");
36        System.out.println("Printing Key-Value Pair");
37        MapIterator<String, Integer> it = map.iterator();
38        printMap(map, it);
```

My iterator print function in main

```
73
74    public static void printMap(HashMapIterable<String, Integer> map ,MapIterator<String, Integer> it){
75        int i = 0;
76        while (it.hasNext())
77        {
78            String key = it.next();
79            System.out.print(key + "\t" +map.get(key) +  "\t");
80            if (i % 2 == 1) System.out.println();
81            i++;
82        }
83    }
84
```

Also put new items after writing to show, putting new items not raise erros

```
48
49          map.put("stone", 10);    map.put("queens", 11);
50          map.put("forever", 12); map.put("covid", 13);
51          map.put("foo", 14);      map.put("bar", 15);
52
```

After, I have to see, how can I iterate, `next()` and `prev()` methods, so I called those methods 1k time respectively.

```
52
53          System.out.println("\nAfter putting new entries. Call 1000 times next and 1000 times prev methods again");
54          for (int i = 0; i < 1000; i++) it.next();
55          for (int i = 0; i < 1000; i++) it.prev();
56
57          System.out.println("\nPrint map without iterator");
58          System.out.println(map);
59          System.out.println("Print map with iterator (Remember that prints from last position after 1000 next() and prev() methods)");
60          printMap(map, it);
61
```

Finally, showed, using methods are correct again

```
    System.out.println("\nStart iterator with given key's position");
    System.out.println("it = map.iterator('YSA')");
    it = map.iterator("YSA");
    System.out.println("Use next() method and prev() method respectively -> " + it.next() + " " + it.prev());
    System.err.println("\nTry with non existing value. (it = map.iterator('Not exist')");
    it = map.iterator("Not exist");
    System.out.println("Use next() method and prev() method respectively -> " + it.next() + " " + it.prev());

    System.out.println("\nPart 1 Test Finished\nThanks for Testing <3");
}
```

Part-2

`LinkedList Implementation`

Frist, I create object with `string,integer` pair and `integer,integer` pair to see, this implementation fits possible data tpyes

```
System.out.println("\nPart2 Tests are starting.\n");
System.out.println("\n----------Linked List Implementation----------");
System.out.println("\nFirst, test HashTableChainLinkedList implementation");
System.out.println("CAPACITY = 3, HOLD_THRESHOLD= 3.0\n");
System.out.println("Test with string keys");
HashTableChainLinkedList<String,Integer> linkedHash = new HashTableChainLinkedList<>();
TestHashTableChain_1(linkedHash);
System.out.println("\nAfter String keys, Test with Integer keys");
HashTableChainLinkedList<Integer,Integer> linkedHash2 = new HashTableChainLinkedList<>();
TestHashTableChain_2(linkedHash2);
```

`TreeSet Implementation`

Have same implementation. So has some test functions

```
    TestHashTableChain_2(linkedHash2);

    System.out.println("\n----------TreeSet Implementation----------");
    System.out.println("\nSecond, test HashTableChainTreeSet implementation");
    System.out.println("CAPACITY = 3, HOLD_THRESHOLD= 3.0\n");
    System.out.println("Test with string keys");
    HashTableChainTreeSet<String,Integer> treeSetHash = new HashTableChainTreeSet<>();
    TestHashTableChain_1(treeSetHash);
    System.out.println("\nAfter String keys, Test with Integer keys");
    HashTableChainTreeSet<Integer,Integer> treeSetHash2 = new HashTableChainTreeSet<>();
    TestHashTableChain_2(treeSetHash2);
```

Show strings works fine

```
ublic static void TestHashTableChain_1(KWHashMap<String,Integer> map) {

  System.out.println("Insert following entries.");
  System.out.println("{foo=0, bar=1, YSA=2, Erdogan Hoca=3, Burak Hoca=4, John=5, Doe=6, Peace=7, Lennon=8, Beatles=9}");

  map.put("foo", 0);      map.put("bar", 1);
  map.put("YSA", 2);      map.put("Erdogan", 3);
  map.put("Burak", 4);    map.put("John", 5);
  map.put("Doe", 6);      map.put("Peace", 7);
  map.put("Lennon", 8);   map.put("Beatles", 9);

  System.out.println("\nPrint map");
  System.out.println(map);

  System.out.println("Strings working fine");
```

Show Integer and big number implementations works fine

First add new 30 values. Selected value number as 30 to keeping toString() result shorter
Than, try to remove more than 30 values.
Lastly, add 10k and remove 10k elements and show working fine.

```
public static void TestHashTableChain_2(KWHashMap<Integer,Integer> map) {

    System.out.println("First, Test with a loop that iterates 30 items and print");

    for (int i = 0; i < 30; i++) map.put(i, i*i*i);
    System.out.println(map);


    System.out.println("Remove all elements and more invalid elements and print.\n(Probably it will print only null array values [indexes])");
    for (int i = 0; i < 100; i++) map.remove(i);

    System.out.println(map);

    System.out.println("\nNow, Add 10k elements and remove 10k elements respectively.");

    for(int i = 0; i < 10000; i++) map.put(i, i*i);
    for(int i = 0; i < 10000; i++) map.remove(i);


    System.out.println("\nTest Finished. Thanks for testing <3\n");
}
```

`Coalesced Implementation`

PDF Example

First, show pdf's example to working correct.
After, add new test cases to this example.

```
}

public static void TestCoalescedHash_example() {
    HashTableCoalesced<Integer, Integer> table = new HashTableCoalesced<>();

    System.out.println("CoalescedHashMap Testing Starting");
    System.out.println("Load same example in pdf");
    System.out.println("Input = {3, 12, 13, 25, 23, 51, 42}");
    System.out.println("Print table");

    table.put(3, 0);    table.put(12, 0);
    table.put(13, 0);   table.put(25, 0);
    table.put(23, 0);   table.put(51, 0);
    table.put(42, 0);

    System.out.println(table);

    System.out.println("\nDelete 13 and print table again");
    table.remove(13);
    System.out.println(table);

    System.out.println("\nDo some other testings with this data");
    System.out.println("Add new data to this example to show hashing working fine");
    System.out.println("New Inputs: {17,18,24,59,96,146,5,4,66,18,70,71}\n");

    table.put(17, 0);   table.put(18, 0);
    table.put(24, 0);   table.put(59, 0);
    table.put(96, 0);   table.put(146, 0);
    table.put(5, 0);    table.put(4, 0);
    table.put(66, 0);   table.put(18, 0);
    table.put(70, 0);   table.put(71, 0);

    System.out.println(table);

    System.out.println("Remove 3 and print again");
    table.remove(3);

    System.out.println(table);
    System.out.println("Add existing element. Remove an element which not in the list.");
    table.put(51, 1); table.remove(999);

    System.out.println("Test finished.");

}
```

Big Values Example

First, add 300 values. (selected 300, because with this Capacity and LOAD_THRESHOLD values, It would be seems fine.
After, try same thing with 1k value. This raise error. This example shows, with more elements, this is not good handler.
Finally, create empty table and show how works when there are just only 1 value.

```java
}

public static void TestCoalescedHash() {
    HashTableCoalesced<Integer, Integer> table = new HashTableCoalesced<>();

    System.out.println("\nAnother test for CoalescedHashTable.\nCAPACITY = 10 , LOAD_THRESHOLD = 0.25\n");
    System.out.println("Fill this table with 300 values through a for loop.");
    System.out.println("Try in try-catch block in case raise error.");

    try {
        for (int i = 0; i < 300; i++) table.put(i, 0);
    } catch (Exception e) {
        System.out.println(e.toString());
    }

    System.out.println("Created Successfully.\n\nLets try with 1000 entry.");
    System.out.println("Creating new CoalescedHashMap and fill this with for loop");

    table = new HashTableCoalesced<>();

    System.out.println("\nThis is also have same fields.\nCAPACITY = 10, LOAD_THRESHOLD = 0.5");
    System.out.println("Try in try-catch block in case raise error.");

    try {
        for (int i = 0; i < 1000; i++) table.put(i, 0);
    } catch (Exception e) {
        System.out.println("\n" +  e.toString() + "\n");
    }

    System.out.println("Throws error. :(. This LOAD_THRESHOLD not enough for 1k entry. It could be incremented but reduce performance of course.");

    System.out.println("\nCreate new CoalescedHashTable with one String element and print");

    HashTableCoalesced<String,Integer> table2 = new HashTableCoalesced<>();

    table2.put("Foo", 0);
    System.out.println(table2);

    System.err.println("CoalescedHashTable testings are finished. Thanks for Testing <3\n");
}
```

Running Command and Results

Part-1

```
----------Welcome to world best test class----------

Part_1 Test and Part_2 Test will called


Testing Custom Class HashMapIterable which extended from HashMap
Contains MapIterator
Insert following entries.
{foo=0, bar=1, YSA=2, Erdogan Hoca=3, Burak Hoca=4, John=5, Doe=6, Peace=7, Lennon=8, Beatles=9}

Print map

{Burak=4, bar=1, YSA=2, Beatles=9, foo=0, Erdogan=3, John=5, Peace=7, Doe=6, Lennon=8}

Create zero-parameter iterator -> map.iterator()
Use while(it.hashNext()) and print map with iterator
Printing Key-Value Pair
Burak   4       bar     1
YSA     2       Beatles 9
foo     0       Erdogan 3
John    5       Peace   7
Doe     6       Lennon  8
```

```
After calling while loop with it.hasNext() method
Call next method 1000 times and prev method 1000 times to show there is no error happening

Add new following pairs
{stone=10, queens=11, forever=12, covid=13, foo=14, bar=15} (inserted already existed keys)

After putting new entries. Call 1000 times next and 1000 times prev methods again

Print map without iterator
{covid=13, YSA=2, Beatles=9, foo=14, John=5, Lennon=8, stone=10, Burak=4, bar=15, Erdogan=3, Peace=7, queens=11, Doe=6,
forever=12}
Print map with iterator (Remember that prints from last position after 1000 next() and prev() methods)
Peace   7       queens  11
Doe     6       forever 12

Start iterator with given key's position
it = map.iterator('YSA')
Use next() method and prev() method respectively → Beatles YSA

Try with non existing value. (it = map.iterator('Not exist')
Use next() method and prev() method respectively → covid forever

Part 1 Test Finished
Thanks for Testing <3
```

Part-2

LinkedList String Test

```
----------Linked List Implementation----------

First, test HashTableChainLinkedList implementation
CAPACITY = 3, HOLD_THRESHOLD= 3.0

Test with string keys
Insert following entries.
{foo=0, bar=1, YSA=2, Erdogan Hoca=3, Burak Hoca=4, John=5, Doe=6, Peace=7, Lennon=8, Beatles=9}

Print map
{
Index =0        | Key:Beatles Val:9 |   | Key:Doe Val:6 |
Index =1        | Key:Burak Val:4 |
Index =2        | Key:Lennon Val:8 |    | Key:YSA Val:2 |
Index =3        | Key:Peace Val:7 |
Index =4        | Key:John Val:5 |      | Key:Erdogan Val:3 |   | Key:bar Val:1 |       | Key:foo Val:0 |
}
Strings working fine
```

LinkedList Integer test

```
After String keys, Test with Integer keys
First, Test with a loop that iterates 30 items and print
{
Index =0        | Key:22 Val:10648 |     | Key:11 Val:1331 |      | Key:0 Val:0 |
Index =1        | Key:23 Val:12167 |     | Key:12 Val:1728 |      | Key:1 Val:1 |
Index =2        | Key:24 Val:13824 |     | Key:13 Val:2197 |      | Key:2 Val:8 |
Index =3        | Key:25 Val:15625 |     | Key:14 Val:2744 |      | Key:3 Val:27 |
Index =4        | Key:26 Val:17576 |     | Key:4 Val:64 |         | Key:15 Val:3375 |
Index =5        | Key:27 Val:19683 |     | Key:16 Val:4096 |      | Key:5 Val:125 |
Index =6        | Key:28 Val:21952 |     | Key:17 Val:4913 |      | Key:6 Val:216 |
Index =7        | Key:29 Val:24389 |     | Key:18 Val:5832 |      | Key:7 Val:343 |
Index =8        | Key:19 Val:6859 |      | Key:8 Val:512 |
Index =9        | Key:20 Val:8000 |      | Key:9 Val:729 |
Index =10       | Key:21 Val:9261 |      | Key:10 Val:1000 |
}
Remove all elements and more invalid elements and print.
(Probably it will print only null array values [indexes])
{
Index =0
Index =1
Index =2
Index =3
Index =4
Index =5
Index =6
Index =7
Index =8
Index =9
Index =10
}

Now, Add 10k elements and remove 10k elements respectively.

Test Finished. Thanks for testing <3
```

TreeSet String test

```
----------TreeSet Implementation----------

Second, test HashTableChainTreeSet implementation
CAPACITY = 3, HOLD_THRESHOLD= 3.0

Test with string keys
Insert following entries.
{foo=0, bar=1, YSA=2, Erdogan Hoca=3, Burak Hoca=4, John=5, Doe=6, Peace=7, Lennon=8, Beatles=9}

Print map
{
Index =0        Doe→6  Beatles→9
Index =1        Burak→4
Index =2        YSA→2  Lennon→8
Index =3        Peace→7
Index =4        foo→0  bar→1  John→5 Erdogan→3
}
Strings working fine
```

TreeSet Integer test

```
After String keys, Test with Integer keys
First, Test with a loop that iterates 30 items and print
{
Index =0        22→10648       11→1331        0→0
Index =1        23→12167       12→1728        1→1
Index =2        24→13824       13→2197        2→8
Index =3        25→15625       14→2744        3→27
Index =4        26→17576       15→3375        4→64
Index =5        27→19683       16→4096        5→125
Index =6        28→21952       17→4913        6→216
Index =7        29→24389       18→5832        7→343
Index =8        19→6859        8→512
Index =9        20→8000        9→729
Index =10       21→9261        10→1000
}
Remove all elements and more invalid elements and print.
(Probably it will print only null array values [indexes])
{
Index =0
Index =1
Index =2
Index =3
Index =4
Index =5
Index =6
Index =7
Index =8
Index =9
Index =10
}

Now, Add 10k elements and remove 10k elements respectively.

Test Finished. Thanks for testing <3
```

Coalesced PDF Example

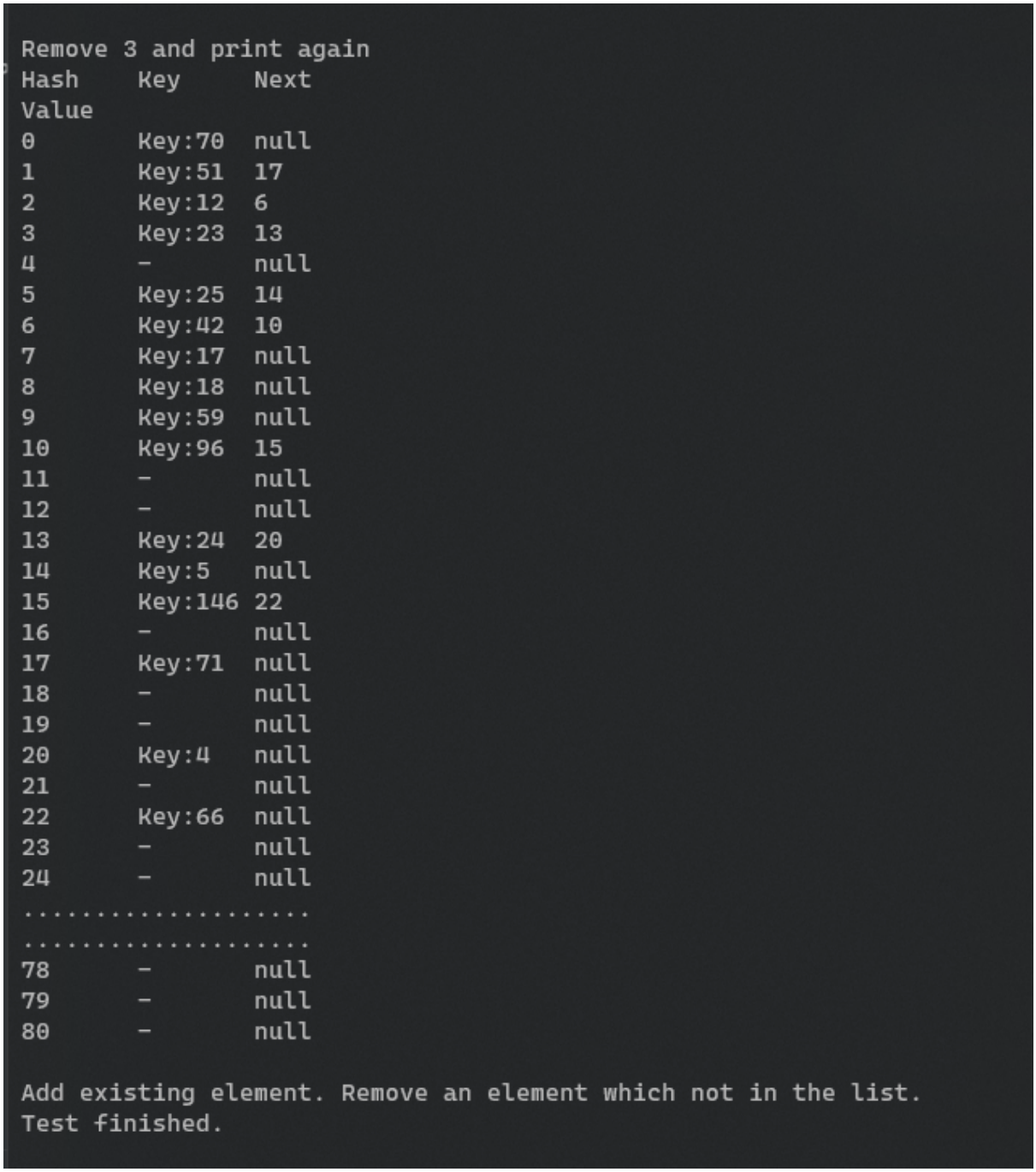
```
-----------Coalesced Hash Implementation-----------
CoalescedHashMap Testing Starting
Load same example in pdf
Input = {3, 12, 13, 25, 23, 51, 42}
Print table
Hash    Key     Next
Value
0       -       null
1       Key:51  null
2       Key:12  6
3       Key:3   4
4       Key:13  7
5       Key:25  null
6       Key:42  null
7       Key:23  null
8       -       null
9       -       null
.....................
.....................
38      -       null
39      -       null
40      -       null


Delete 13 and print table again
Hash    Key     Next
Value
0       -       null
1       Key:51  null
2       Key:12  6
3       Key:3   4
4       Key:23  null
5       Key:25  null
6       Key:42  null
7       -       null
8       -       null
.....................
.....................
38      -       null
```

Upgrade this example with additions

```
Do some other testings with this data
Add new data to this example to show hashing working fine
New Inputs: {17,18,24,59,96,146,5,4,66,18,70,71}

Hash    Key     Next
Value
0       Key:70  null
1       Key:51  17
2       Key:12  6
3       Key:3   4
4       Key:23  13
5       Key:25  14
6       Key:42  10
7       Key:17  null
8       Key:18  null
9       Key:59  null
10      Key:96  15
11      -       null
12      -       null
13      Key:24  20
14      Key:5   null
15      Key:146 22
16      -       null
17      Key:71  null
18      -       null
19      -       null
20      Key:4   null
21      -       null
22      Key:66  null
23      -       null
24      -       null
.....................
.....................
78      -       null
79      -       null
80      -       null
```



Show working with Integer, big values and empty table with one key

```
Another test for CoalescedHashTable.
CAPACITY = 10 , LOAD_THRESHOLD = 0.25

Fill this table with 300 values through a for loop.
Try in try-catch block in case raise error.
Created Successfully.

Lets try with 1000 entry.
Creating new CoalescedHashMap and fill this with for loop

This is also have same fields.
CAPACITY = 10, LOAD_THRESHOLD = 0.5
Try in try-catch block in case raise error.

java.lang.IndexOutOfBoundsException: Index -2147479015 out of bounds for length 2560

Throws error. :(. This LOAD_THRESHOLD not enough for 1k entry. It could be incremented but reduce performance of course.

Create new CoalescedHashTable with one String element and print
Hash    Key     Next
Value
0       -       null
1       -       null
2       Key:Foo null
3       -       null
4       -       null
.................
.................
8       -       null
9       -       null
```