# GTU Department of Computer Engineering
# CSE 222/505 - Spring 2021
# Homework # 5

**AHMET OKUR**
1801042655

## 1. PROBLEM SOLUTION APPROACH

### PART 1

first I created a class(AOHashMap) in part 1 expended to HashMap. I can now use the features of HashMap.
I created an interface class named iterator_map. I added Hasprev and Prev methods into it. Then I implemented the iterator class to my AOHashMap class. I used HashMap.keySet ().iterator() properties in my iterator class

### PART 2

In this part, I implemented KW HashMap with 3 different way.

I made a chain rule using a LinkedList as the 1st path.I have benefited from the book in most of this part.

I used TreeSet on the 2nd path While writing this part, I had to implement the comparable class to the entry class.This is because TreeSet need comparison

As the 3rd way, I first wrote the find method. This method performs quadratic operations and returns index.
I wrote the put and remove methods easily because I had an index.

## 2. TEST CASES,RUNNING AND RESULTS

### PART 1

Firstly,I added the elements

```
map_ahmet<Integer,String> Galatasaray=new map_ahmet<>();
Galatasaray.put(1,"Muslera");
Galatasaray.put(2,"Yedlin");
Galatasaray.put(3,"Donk");
Galatasaray.put(4,"Marcao");
Galatasaray.put(5,"Saracchi");
Galatasaray.put(6,"Arda");
Galatasaray.put(7,"Emre");
Galatasaray.put(8,"Taylan");
Galatasaray.put(9,"Gedson");
Galatasaray.put(10,"Babel");
Galatasaray.put(11,"Halil");
System.out.println("MAP->"+Galatasaray);
```

you can see the map

```
MAP->{1=Muslera, 2=Yedlin, 3=Donk, 4=Marcao, 5=Saracchi, 6=Arda, 7=Emre, 8=Taylan, 9=Gedson, 10=Babel, 11=Halil}
```

I started the iteretor from key number 3

```
Iterator_map iter=Galatasaray.iterator( key: 3);
System.out.println("\n" +"The iterator started from the 3rd key");
```

Next() method

```
System.out.println("\n" +"next method");
while (iter.hasNext()){
    System.out.println(iter.next());
}
```

```
The iterator started from the 3rd key

next method
3
4
5
6
7
8
9
10
11
```

Prev() method

```
System.out.println("\n" +"prev method");
while (iter.hasPrev()){
    System.out.println(iter.prev());
}
```

```
prev method
11
10
9
8
7
6
5
4
3
2
1
```

## PART 2

## Chain->linkelist

Put method

```
test.put(3,"Muslera");
test.put(12,"Yedlin");
test.put(13,"Donk");
test.put(25,"Marcao");
test.put(23,"Arda");
test.put(51,"Taylan");
test.put(42,"Gedson");
System.out.println("****After Put****");
System.out.println(test);
```

Fine work

```
****HashTableChain->LinkedList TableSize=10***
****After Put****

index= 1[key=51 value=Taylan]
index= 2[key=42 value=Gedson, key=12 value=Yedlin]
index= 3[key=23 value=Arda, key=13 value=Donk, key=3 value=Muslera]
index= 5[key=25 value=Marcao]
```

Remove test

```
System.out.println("\n****After Remove Key=12,51,23****");
test.remove( key: 12);
test.remove( key: 51);
test.remove( key: 23);
System.out.println(test);
```

Fine work

```
****After Remove Key=12,51,23****

index= 2[key=42 value=Gedson]
index= 3[key=13 value=Donk, key=3 value=Muslera]
index= 5[key=25 value=Marcao]
```

with and without. Get method

```
System.out.println("\nGet key 42 ->"+test.get(42));
System.out.println("Get key 111 ->"+test.get(111));
```

111 null because Not in the hash table

```
Get key 42 ->Gedson
Get key 111 ->null
```

Size methot

```
System.out.println("\n***HashMap Size***->"+test.size());
```

```
***HashMap Size***->4
```

## Chain->TreeSet

Put method

```
AOHashtableChainTree<Integer,String> test=new AOHashtableChainTree<>();
System.out.println("****HashTableChain->TreeSet TableSize=10***");
test.put(3,"Muslera");
test.put(12,"Yedlin");
test.put(13,"Donk");
test.put(25,"Marcao");
test.put(23,"Arda");
test.put(51,"Taylan");
test.put(42,"Gedson");
System.out.println("****After Put****");
System.out.println(test);
```

Fine work

```
****HashTableChain->TreeSet TableSize=10***
****After Put****

index= 1[key=51 value=Taylan]
index= 2[key=12 value=Yedlin, key=42 value=Gedson]
index= 3[key=3 value=Muslera, key=13 value=Donk, key=23 value=Arda]
index= 5[key=25 value=Marcao]
```

Remove test

```
System.out.println("\n****After Remove Key=12,51,23****");
test.remove( key: 12);
test.remove( key: 51);
test.remove( key: 23);
System.out.println(test);
```

Fine work

```
****After Remove Key=12,51,23****

index= 2[key=42 value=Gedson]
index= 3[key=3 value=Muslera, key=13 value=Donk]
index= 5[key=25 value=Marcao]
```

with and without. Get method

```
System.out.println("\nGet key 42 ->"+test.get(42));
System.out.println("Get key 111 ->"+test.get(111));
```

111 null because Not in the hash table

```
Get key 42 ->Gedson
Get key 111 ->null
```

Size method

```
System.out.println("\n***HashMap Size***->"+test.size());
```

```
***HashMap Size***->4
```

## Open->Quadratic

Put method (example pdf)

```
AOHashMapOpenQuadratic<Integer,String> test=new AOHashMapOpenQuadratic<>();
System.out.println("****HashTableOpen TableSize=10***");
test.put(3,"Muslera");
test.put(12,"Yedlin");
test.put(13,"Donk");
test.put(25,"Marcao");
test.put(23,"Arda");
test.put(51,"Taylan");
test.put(42,"Gedson");
System.out.println("****After Put****");
System.out.println(test);
```

Fine work

```
****HashTableOpen TableSize=10***
****After Put****

index= 1->key=51 value=Taylan
index= 2->key=12 value=Yedlin
index= 3->key=3 value=Muslera
index= 4->key=13 value=Donk
index= 5->key=25 value=Marcao
index= 6->key=42 value=Gedson
index= 7->key=23 value=Arda
```

Remove method

```
System.out.println("\n****After Remove Key=12,51,23****");
test.remove( key: 12);
test.remove( key: 51);
test.remove( key: 23);
System.out.println(test);
```

Fine work

```
****After Remove Key=12,51,23****

index= 2->key=42 value=Gedson
index= 3->key=3 value=Muslera
index= 4->key=13 value=Donk
index= 5->key=25 value=Marcao
```

with and without. Get method

```
System.out.println("\nGet key 42 ->"+test.get(42));
System.out.println("Get key 111 ->"+test.get(111));
```

111 null because Not in the hash table

```
Get key 42 ->Gedson
Get key 111 ->null
```

Size method

```
System.out.println("\n***HashMap Size***->"+test.size());
```

```
***HashMap Size***->4
```

## BIG NUMBER TEST FOR  THEREE STRUCTER

```
AOHashtableChain<Integer,Integer> test1=new AOHashtableChain<>();
for(int i=0;i<200;i++){
    test1.put(i,i*2);
}
for(int i=0;i<200;i++){
    test1.remove(i);
}
System.out.println(test1);

AOHashtableChainTree<Integer,Integer> test2=new AOHashtableChainTree<>();
for(int i=0;i<200;i++){
    test2.put(i,i*2);
}
for(int i=0;i<200;i++){
    test2.remove(i);
}
System.out.println(test2);

AOHashMapOpenQuadratic<Integer,Integer> test3=new AOHashMapOpenQuadratic<>();
for(int i=0;i<200;i++){
    test3.put(i,i*2);
}

for(int i=0;i<200;i++){
    test3.remove(i);
}
System.out.println(test3);
```

```
C:\Users\user\.jdks\corretto-11.0.10\bin\java.exe "-javaagent:C:\Pr




Process finished with exit code 0
```

## PERFORMANCE TEST FOR  THEREE STRUCTER

```java
public static void test_performance(){
    long startTime = System.nanoTime();
    AOHashtableChain<Integer,Integer> test1=new AOHashtableChain<>();
    for(int i=0;i<1000;i++){
        test1.put(i,i*2);
    }
    long endTime = System.nanoTime();
    long estimatedTime = endTime - startTime;
    double seconds = (double)estimatedTime/1000000000;
    System.out.println("Performance chain->Linkedlist= "+seconds+" second");


    long startTime2 = System.nanoTime();
    AOHashtableChainTree<Integer,Integer> test2=new AOHashtableChainTree<>();
    for(int i=0;i<1000;i++){
        test2.put(i,i*2);
    }
    long endTime2 = System.nanoTime();
    long estimatedTime2 = endTime2 - startTime2;
    double seconds2 = (double)estimatedTime2/1000000000;
    System.out.println("Performance chain->TreeSet= "+seconds2+" second");

    long startTime3 = System.nanoTime();
    AOHashMapOpenQuadratic<Integer,Integer> test3=new AOHashMapOpenQuadratic<>();
    for(int i=0;i<1000;i++){
        test3.put(i,i*2);
    }
    long endTime3 = System.nanoTime();
    long estimatedTime3 = endTime3 - startTime3;
    double seconds3 = (double)estimatedTime3/1000000000;
    System.out.println("Performance Open addressing= "+seconds3+" second");
```

```
Performance chain->Linkedlist= 0.0037001 second
Performance chain->TreeSet= 0.0044627 second
Performance Open addressing= 8.415E-4 second
```

**Open addressing faster than chain**