GIT Department of Computer Engineering CSE 222/505 - Spring 2021 Homework5 # Report

Mustafa Gurler 171044034

Problem Solution Approach

We need split our homework 4 parts and 1 for main part. All the classes needs to implement our KWHashMap interface except part1 for iterator class.

I should start with my MapIterator class. It is extended by original HashMap of Collection Framework. Therefore it is acting like a real HashMap and included iterator class. Inside of iterator class we can see no parameter constructor and one parameter constructor which takes only Key of entry, starts the iterator index with given key . Iterator class has 3 different features: next(), prev(), and hasNext().

**next() ➔ gives the current key value of keyset. If it reaches to end, gives the 0^{th} key of keyset.*

**prev() ➔ works like next method but opposite.*

**hasNext() ➔ controls if the iterator reaches the end*

Part2 of homework starts with chaining technique which have two different way to do it , first one array holds the data's in a linkedList. Every key has a unique code , that unique code shows which index they are going to insert. Array size starts with 11(prime number) and it needs to be dynamically grown because inserted elements size needs to be smaller than array size.

LoadFactor ➔ (Array size / Inserted element Size) ➔ if we do not obey this rule , HashMap does not give us O(1) time complexity.

Also TreeSet has been inserted with same logic. But Trees has to be comparable so Comparable class has been added to generic values.

TreeSet chaining technique gives more opportunity than linkedlist, Every array index holds one treeset, so searching values are more faster than linkedlist chaining technique (almost O(1)), but other methods are same.

Lastly, using Coalesced hashing technique is a bit different than other two technique implementation. First we can insert only one value and key to array, if key's hashcode has been inserted before we need to use Quadratic Probing in hashing and try to insert to value next appropriate index in the array.

*Quadratic Probe ➔ index = ((firstIndex + (i*i)) % bucketSize*

Remove method removes the key and value and inserts the its next node in the array and last node of the array index which has the same hash code removed key, has been changed its dumpedValue to True. It shows the index has been inserted before but it has been removed. That node is available now.

Performance Results

Put method(1000 value added):

Linkedlist ➔*12231100 nanosn*

TreeSet ➔*37523200 nanosn*

Open-Adressing ➔*4399400 nanosn*

Put method(3000 value added):

Linkedlist ➔*8511400 nanosn*

TreeSet ➔*24654300 nanosn*

Open-Adressing ➔*5648500 nanosn*

Put method(10000 value added):

Linkedlist ➔*60372000 nanosn*

TreeSet ➔*96505300 nanosn*

Open-Adressing ➔*106579600 nanosn*

Get method(1000 value):

Linkedlist ➔*2035000 nanosn*

TreeSet ➔*2659500 nanosn*

Open-Adressing ➔*1250900 nanosn*

Remove method(1000 value):

Linkedlist ➔*4562200 nanosn*

TreeSet ➔*7941300 nanosn*

Open-Adressing ➔*35486100 nanosn*

Test Case Running and Results

MapIterator-Part1

Next() method:

Main

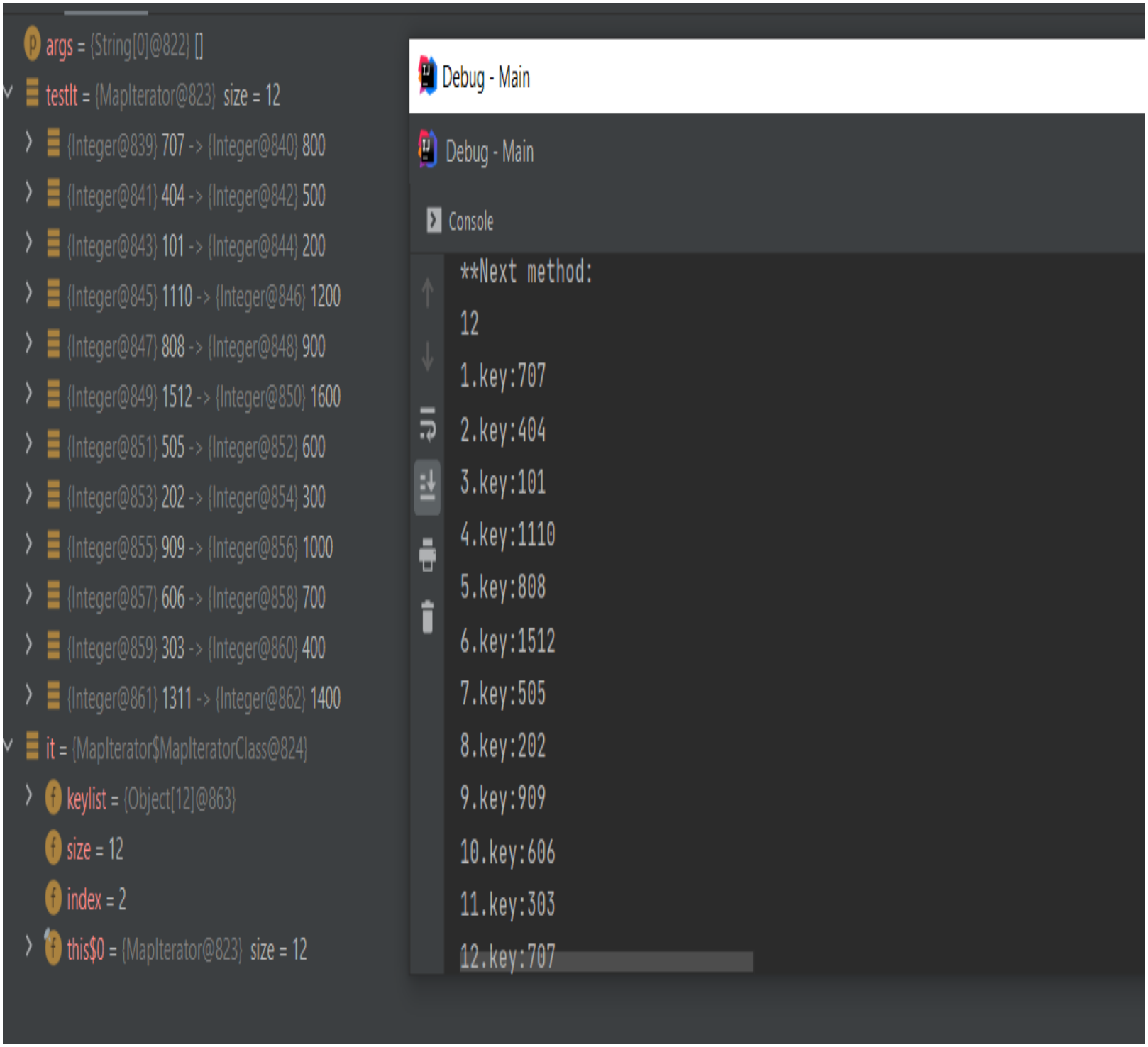
```
System.out.println("*****************************PART1-->Iterator*******************************");

MapIterator<Integer, Integer> testIt = new MapIterator<>();
testIt.put(101,200);
testIt.put(202,300);
testIt.put(303,400);
testIt.put(404,500);
testIt.put(505,600);
testIt.put(606,700);
testIt.put(707,800);
testIt.put(808,900);
testIt.put(909,1000);
testIt.put(1110,1200);
testIt.put(1311,1400);
testIt.put(1512,1600);

MapIterator.MapIteratorClass it = testIt.MapIterator();


System.out.println("**Next method:");
System.out.println(it.size);
for(int i=0 ; i<it.size*2 ; i++){
    System.out.println((i+1) + ".key:" + it.next());
}
```

Console

```
*****************************PART1-->Iterator*******************************
12
1.key:707
2.key:404
3.key:101
4.key:1110
5.key:808
6.key:1512
7.key:505
8.key:202
9.key:909
10.key:606
11.key:303
12.key:707
13.key:404
14.key:101
15.key:1110
16.key:808
17.key:1512
18.key:505
19.key:202
20.key:909
21.key:606
22.key:303
23.key:707
24.key:404
```

As you see size is 12, but after the next method reaches the end, starts the beginning again.

Debugger for next method



Prev Method with given key iterator:

Main

```
MapIterator.MapIteratorClass it2 = testIt.MapIterator( key: 1512);// given key finds it

System.out.println();
System.out.println();
System.out.println();

for(int i=it.size-1 ; i>5 ; i--){
    System.out.println(i + ".key:" + it2.prev());
}
```

Console PREV

```
**Prev method:
previous key:808
previous key:1110
previous key:101
previous key:404
previous key:707
previous key:1311
```

Console NEXT

```
1.key:707
2.key:404
3.key:101
4.key:1110
5.key:808
6.key:1512
```

As you see they move opposite direction.

hasNext() method: next method uses hasNext method to check if it reaches the at the end of iterator.

Chaining Technique-Linkedlist

Put () and getCollision() methods Main :

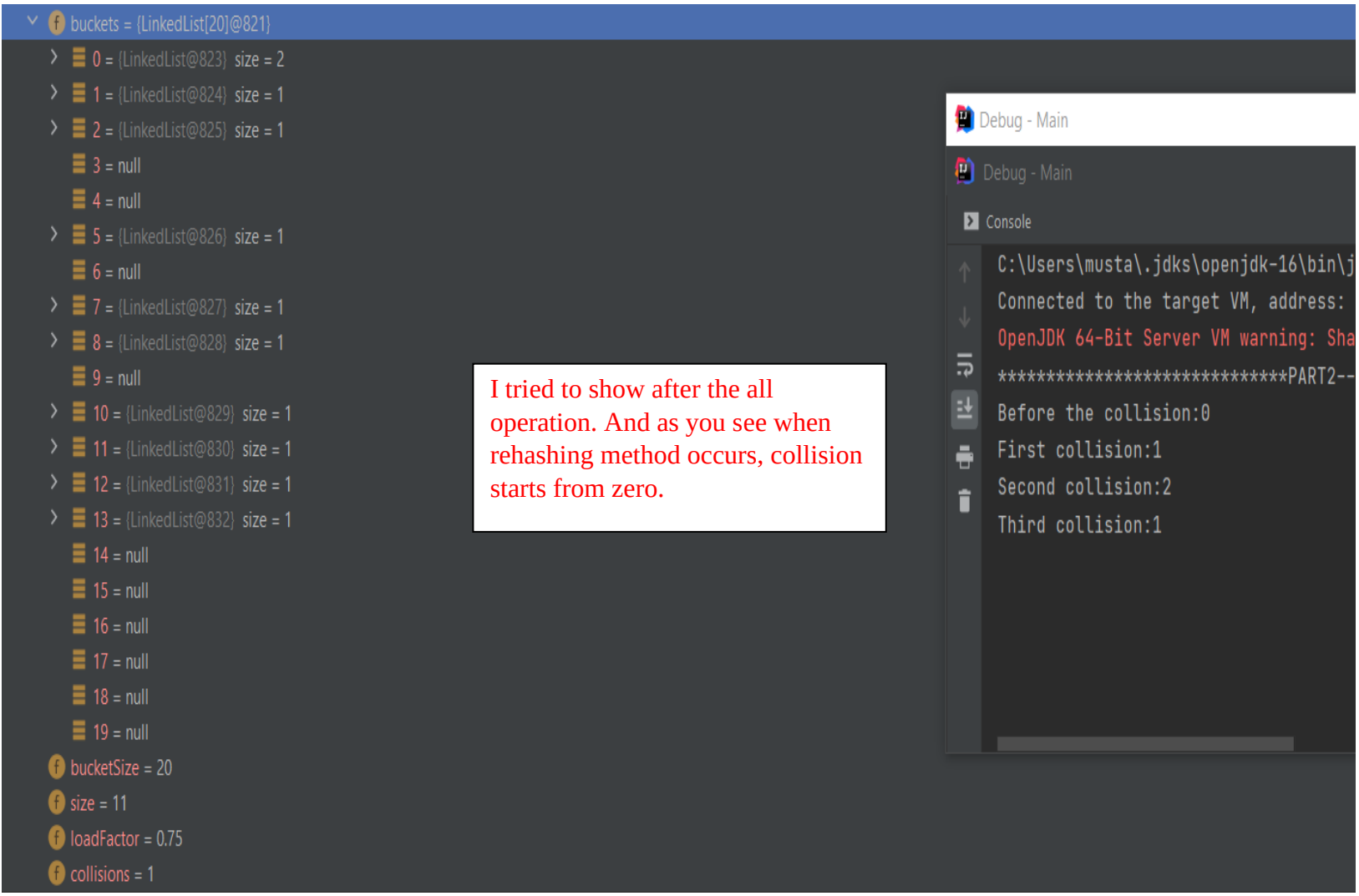
```
System.out.println("*****************************PART2-->LINKEDLIST*****************************");

part2_chaining_linkedlist<Integer,Integer> test = new part2_chaining_linkedlist<>();
test.put(1,10); //1
test.put(10,1); //2
test.put(2,9); //3
System.out.println("Before the collision:" + test.getCollisions());
test.put(100,8); // 4
System.out.println("First collision:" + test.getCollisions());
test.put(1000,100);// here comes the collision // 5
System.out.println("Second collision:" + test.getCollisions());
test.put(100,1000); // second collision // 6
test.put(1000,2);// 7
test.put(8,3);// 8
test.put(7,4);// 9
test.put(5,6);// 10 // Here bucket size needs to be doubled by rehash method
test.put(11,20);// 11
test.put(12,19);// 12
test.put(13,18);// 13
System.out.println("Third collision:" + test.getCollisions()); // old collisions does not count
                                                               // because of the rehashing
System.out.println("size:" + test.getSize() + " bucket size:" + test.getBucketSize());
```

10,100,1000 has to be same bucket index, So 2 collision occurred. After 10th insertion rehash method has been runned by the program. 2 times 100 and 2 times 1000 has been inserted so size is two less.

Console:

```
*****************************PART2-->LINKEDLIST*******************************
Before the collision:0
First collision:1
Second collision:2
Third collision:1
size:11 bucket size:20
```

Debugger to show inside of the our bucket:



get() Method Main:

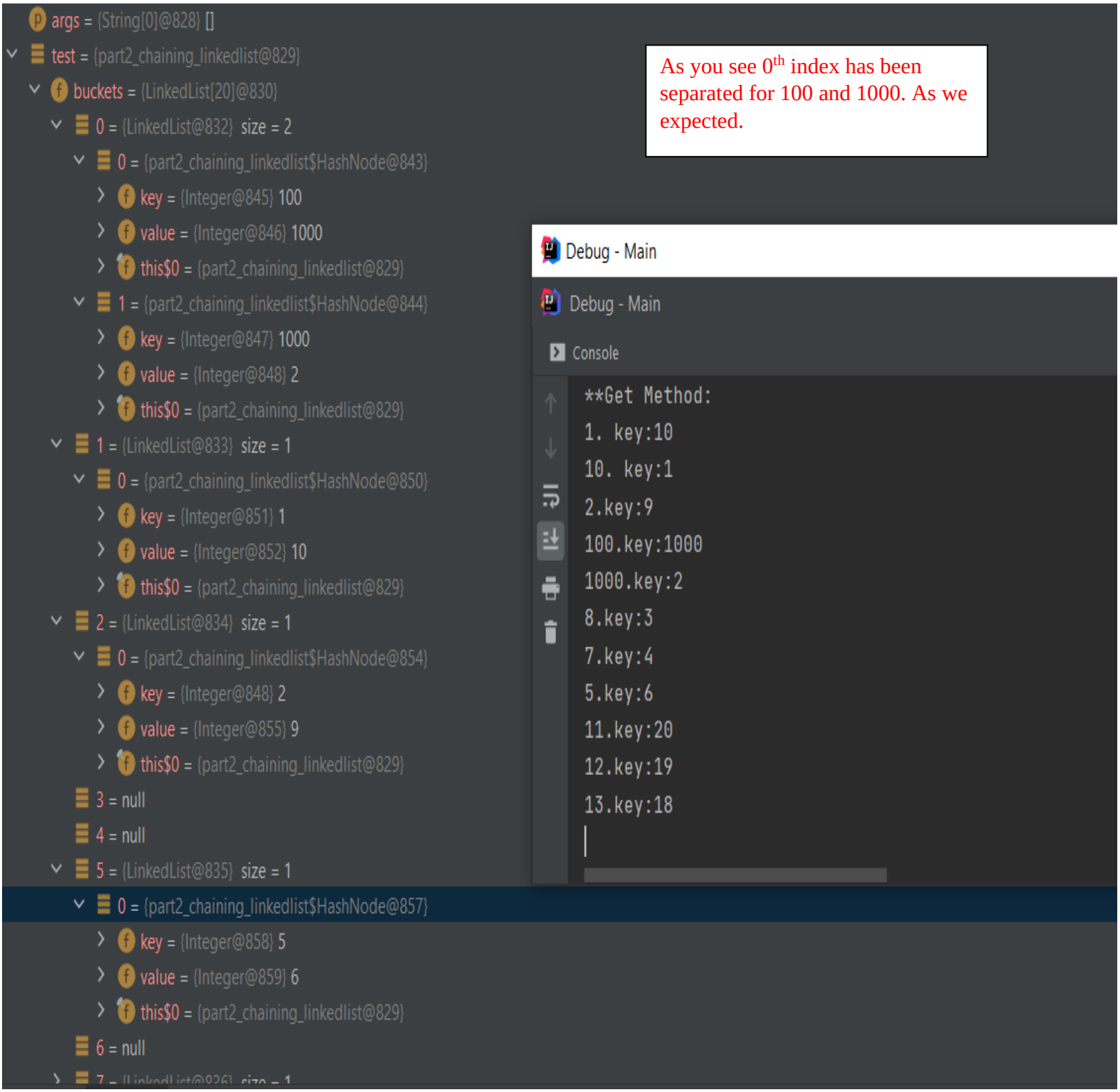
```
System.out.println("**Get Method:");
// I am going to get all the added elements
System.out.println("1. key:" + test.get(1));
System.out.println("10. key:" + test.get(10));
System.out.println("2.key:" + test.get(2));
System.out.println("100.key:" + test.get(100));
System.out.println("1000.key:" + test.get(1000));
System.out.println("8.key:" + test.get(8));
System.out.println("7.key:" + test.get(7));
System.out.println("5.key:" + test.get(5));
System.out.println("11.key:" + test.get(11));
System.out.println("12.key:" + test.get(12));
System.out.println("13.key:" + test.get(13));
System.out.println("They have been tested in a hashmap with 20 size");
```

All the keys and values have been inserted above the put method, They gave the same result as expected.

Console:

```
**Get Method:
1. key:10
10. key:1
2.key:9
100.key:1000
1000.key:2
8.key:3
7.key:4
5.key:6
11.key:20
12.key:19
13.key:18
They have been tested in a hashmap with 20 size
```

Debugger to show insede of the our bucket and Result:



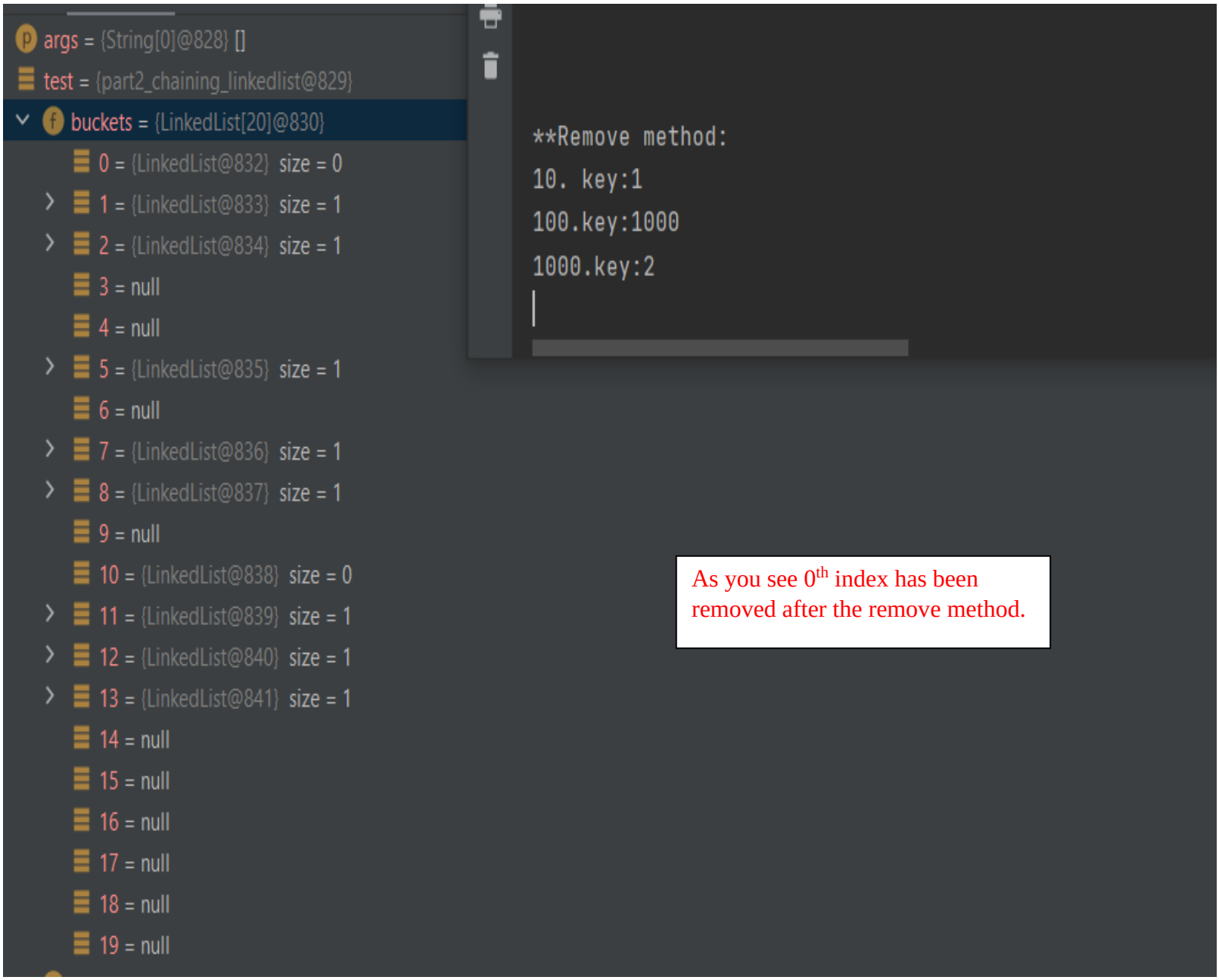
Remove method Main:

```
System.out.println("**Remove method:");
// I am going to get all the added elements
System.out.println("10. key:" + test.remove( key: 10));
System.out.println("100.key:" + test.remove( key: 100));
System.out.println("1000.key:" + test.remove( key: 1000));
System.out.println("They have been tested in a hashmap with 20 size");
```

Console:

```
**Remove method:
10. key:1
100.key:1000
1000.key:2
They have been tested in a hashmap with 20 size
```

Debugger to show inside of our bucket after the remove:



Chaining Technique-TreeSet

Put and getCollisions method:

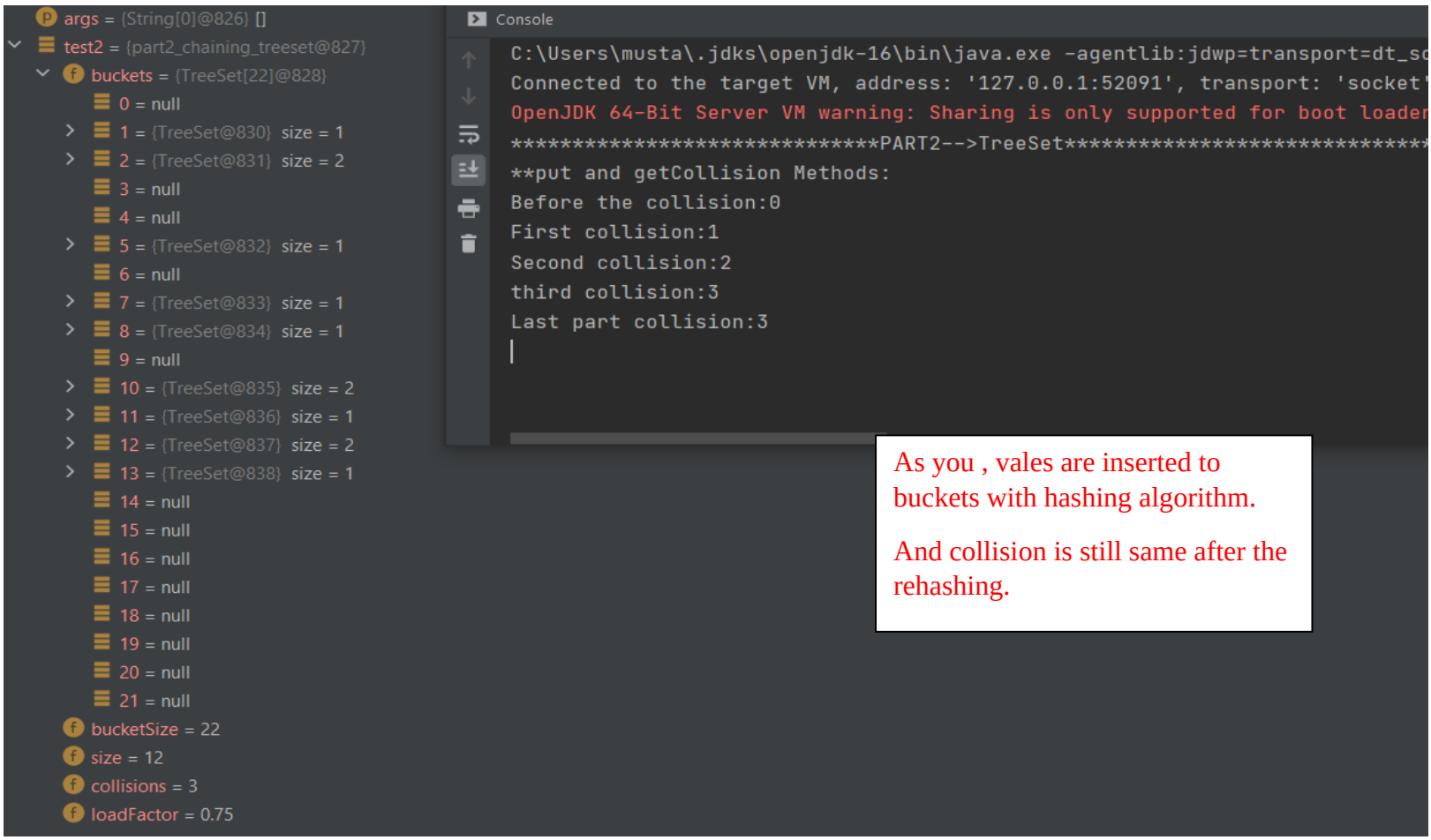
```
System.out.println("*****************************PART2-->TreeSet*******************************");
part2_chaining_treeset<Integer,Integer> test2 = new part2_chaining_treeset<>();

System.out.println("**put and getCollision Methods:");

test2.put(1,500); //1
test2.put(10,1002); //2
test2.put(2,532); //3
System.out.println("Before the collision:" + test2.getCollisions());
test2.put(100,1231); // 4
System.out.println("First collision:" + test2.getCollisions());
test2.put(1000,222);// here comes the collision // 5
System.out.println("Second collision:" + test2.getCollisions());
test2.put(20000,9023); // second collision // 6
System.out.println("third collision:" + test2.getCollisions());
test2.put(1000,4224);// 7
test2.put(8,123123);// 8
test2.put(7,2323);// 9
test2.put(5,24423);// 10 // Here bucket size needs to be doubled by rehash method
test2.put(11,6534);// 11
test2.put(12,3);// 12
test2.put(13,22);// 13
System.out.println("Last part collision:" + test2.getCollisions()); // old collisions does not count
// because of the rehashing
System.out.println("size:" + test2.getSize() + " bucket size:" + test2.getBucketSize());
```

10,100,1000 and 10000 has to be same bucket index, So 3 collision occurred. After 10th insertion rehash method has been runned by the program. 2 times 100 and 2 times 1000 has been inserted so size is two less.

Console:

```
*****************************PART2-->TreeSet*******************************
**put and getCollision Methods:
Before the collision:0
First collision:1
Second collision:2
third collision:3
Last part collision:3
size:12 bucket size:22
```

Debugger for put method:



Get() method():

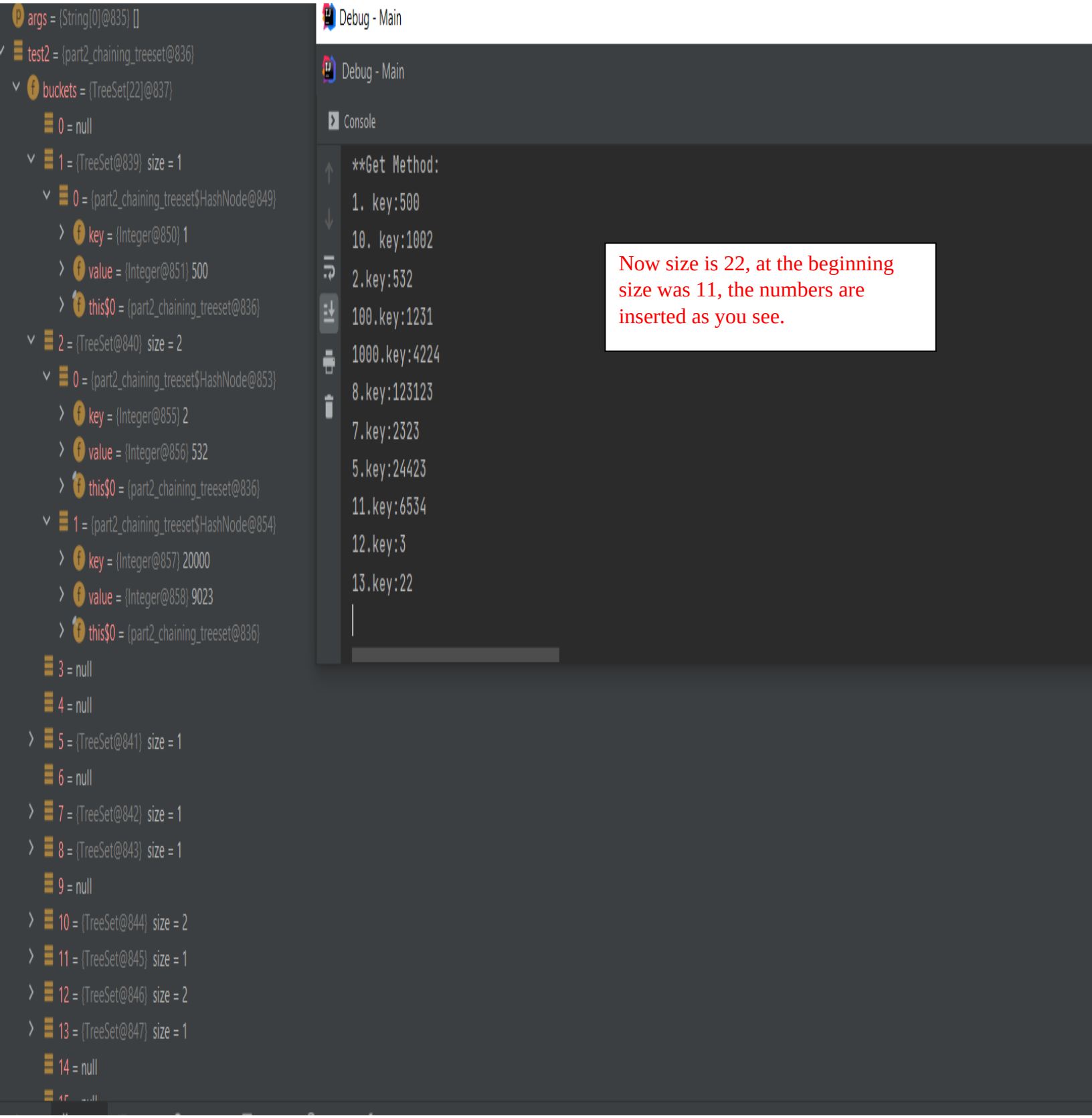
```
System.out.println("**Get Method:");
// I am going to get all the added elements
System.out.println("1. key:" + test2.get(1));
System.out.println("10. key:" + test2.get(10));
System.out.println("2.key:" + test2.get(2));
System.out.println("100.key:" + test2.get(100));
System.out.println("1000.key:" + test2.get(1000));
System.out.println("8.key:" + test2.get(8));
System.out.println("7.key:" + test2.get(7));
System.out.println("5.key:" + test2.get(5));
System.out.println("11.key:" + test2.get(11));
System.out.println("12.key:" + test2.get(12));
System.out.println("13.key:" + test2.get(13));
System.out.println("They have been tested in a hashmap with 20 size");
```

Console:

```
**Get Method:
1. key:500
10. key:1002
2.key:532
100.key:1231
1000.key:4224
8.key:123123
7.key:2323
5.key:24423
11.key:6534
12.key:3
13.key:22
They have been tested in a hashmap with 20 size
```

We got the same values what we inserted above.

Debugger for get method:



Remove method();

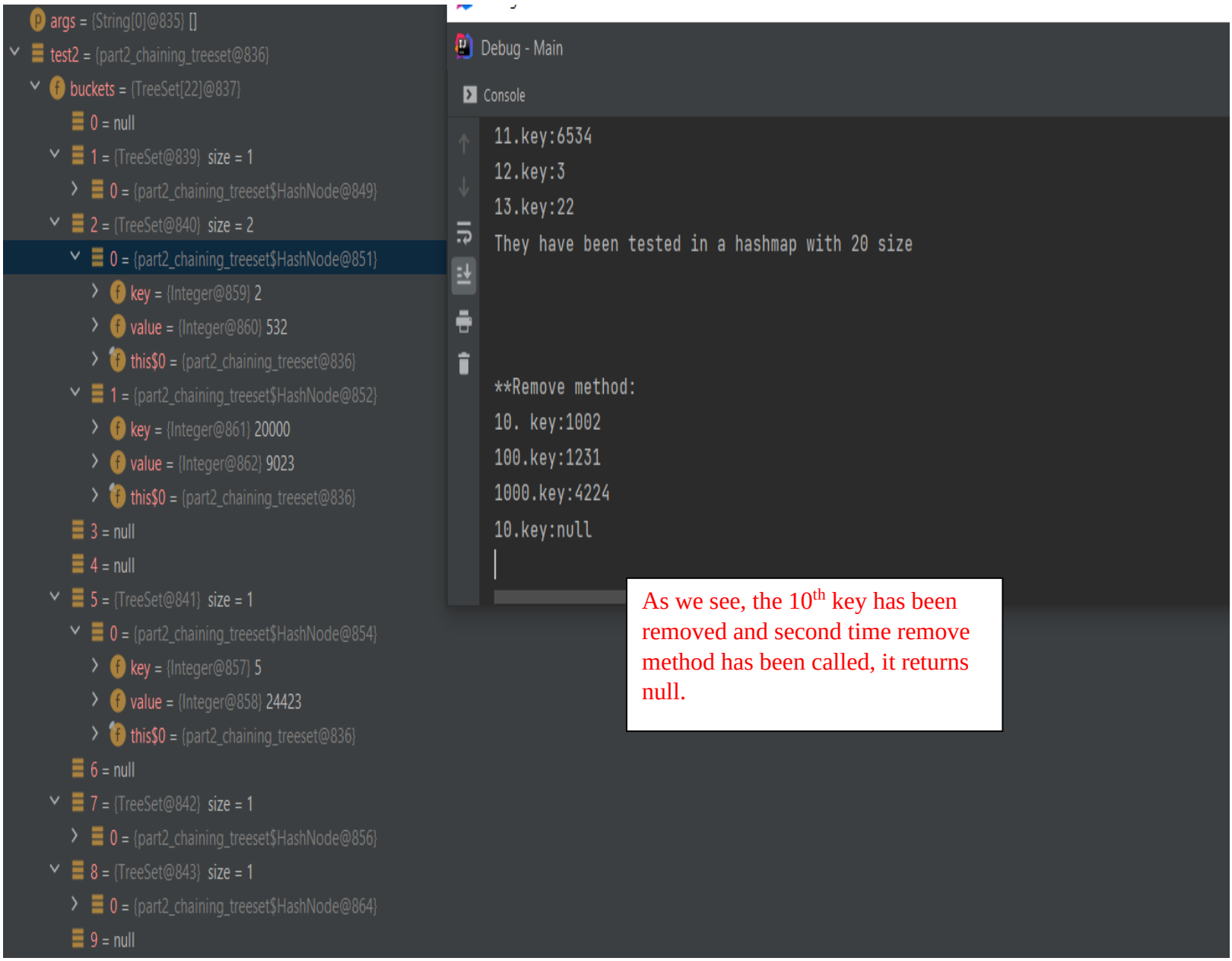
Main

```
System.out.println("**Remove method:");
// I am going to get all the added elements
System.out.println("10. key:" + test2.remove( key: 10));
System.out.println("100.key:" + test2.remove( key: 100));
System.out.println("1000.key:" + test2.remove( key: 1000));
System.out.println("10.key:" + test2.remove( key: 10));
System.out.println("They have been tested in a hashmap with 20 size");
```

Console

```
**Remove method:
10. key:1002
100.key:1231
1000.key:4224
10.key:null
They have been tested in a hashmap with 20 size
```

Debugger for remove method:



As we see, the 10th key has been removed and second time remove method has been called, it returns null.

Coalesced Hashing Technique-Open Addressing

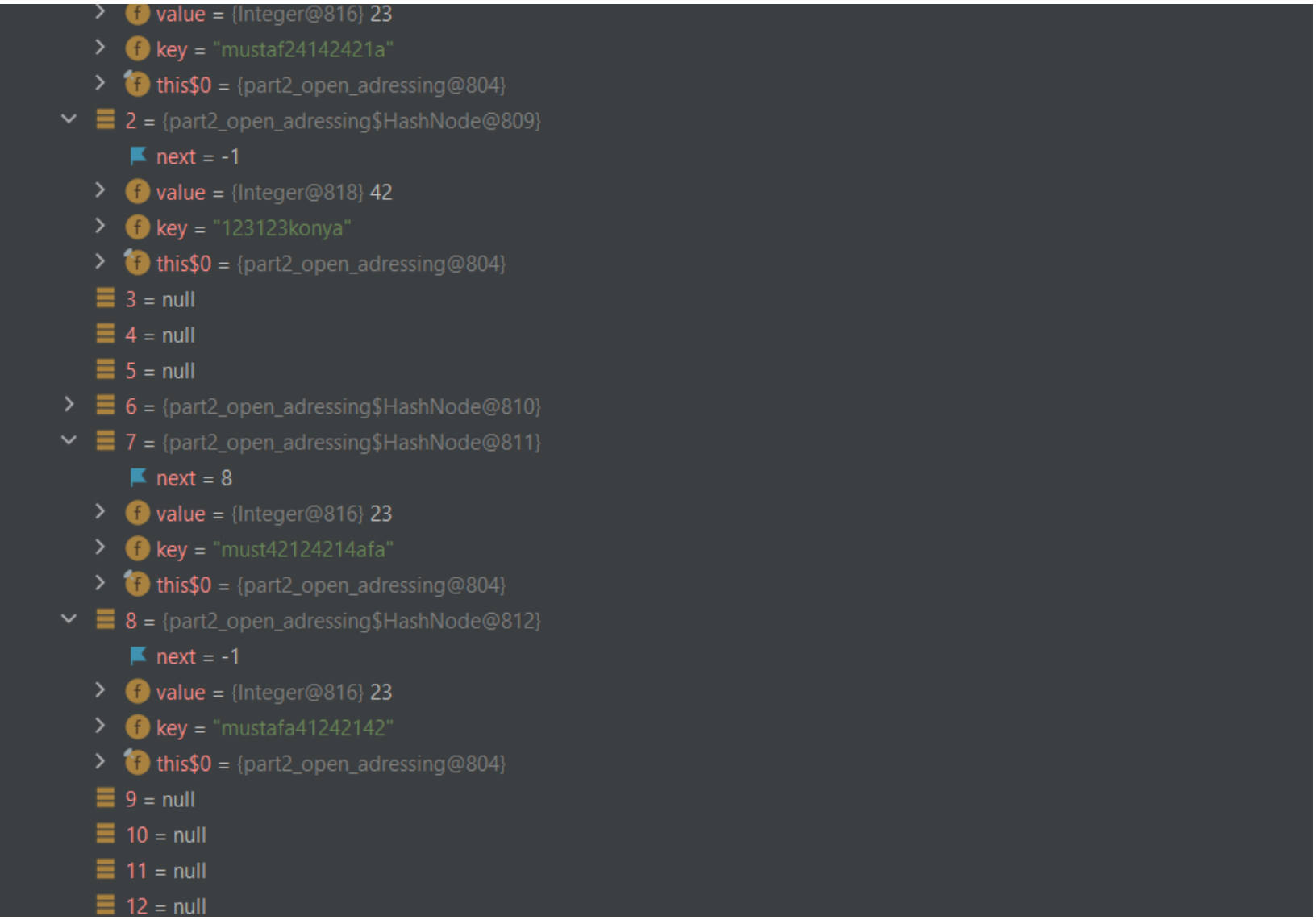
Main

```
part2_open_adressing<String, Integer> test = new part2_open_adressing<>();
test.put("mustafa", 23);//1
test.put("konya",42);//2
test.put("mustaf123132a", 23);//3
test.put("mustaf24142421a", 23);//4
test.put("konya142142",42);//5
test.put("mustafa41242142", 23);//6
test.put("must42124214afa", 23);//7
test.put("123123konya",42);//8
test.put("123132mustafa", 23);//9
test.put("142142mustafa", 23);//10
test.put("1424212konya",42);//11
test.put("mus4242tafa", 23);//12
test.put("asasdads123132",52);//13

System.out.println("Inserted items size:" + test.getSize() + " Bucket Size:" + test.getBucketSize());
```

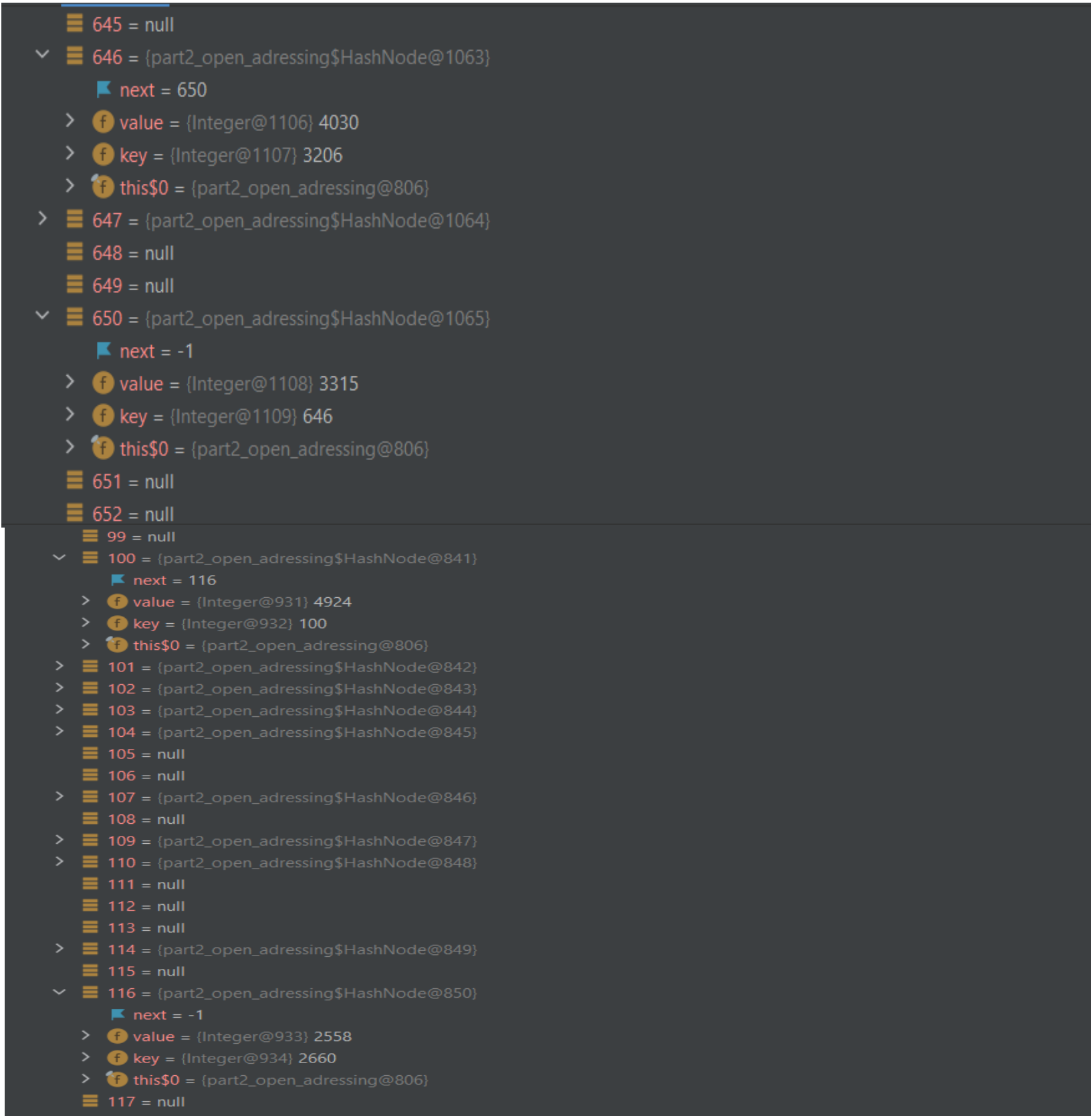
Console

```
Inserted items size:13 Bucket Size:20
```

Debugger



Other debugger with Integer key



Get() method:

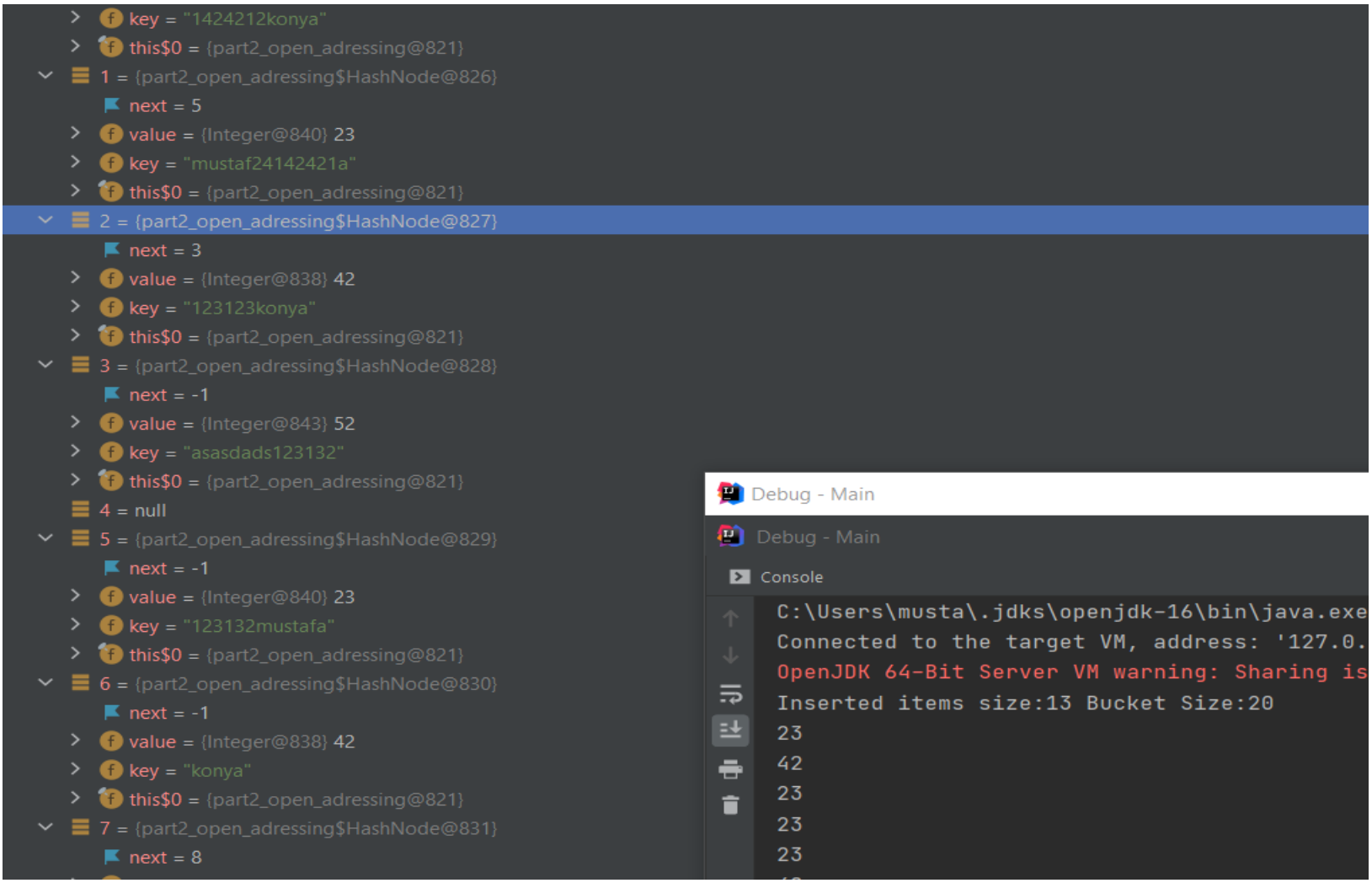
Main:

```
System.out.println(test.get("mustafa"));
System.out.println(test.get("konya"));
System.out.println(test.get("mustaf123132a"));
System.out.println(test.get("mustaf123132a"));
System.out.println(test.get("mustaf24142421a"));
System.out.println(test.get("konya142142"));
System.out.println(test.get("mustafa41242142"));
```

Console

```
23
42
23
23
23
42
23
```

Debugger


key = "1424212konya"
this$0 = {part2_open_adressing@821}
1 = {part2_open_adressing$HashNode@826}
next = 5
value = {Integer@840} 23
key = "mustaf24142421a"
this$0 = {part2_open_adressing@821}
2 = {part2_open_adressing$HashNode@827}
next = 3
value = {Integer@838} 42
key = "123123konya"
this$0 = {part2_open_adressing@821}
3 = {part2_open_adressing$HashNode@828}
next = -1
value = {Integer@843} 52
key = "asasdads123132"
this$0 = {part2_open_adressing@821}
4 = null
5 = {part2_open_adressing$HashNode@829}
next = -1
value = {Integer@840} 23
key = "123132mustafa"
this$0 = {part2_open_adressing@821}
6 = {part2_open_adressing$HashNode@830}
next = -1
value = {Integer@838} 42
key = "konya"
this$0 = {part2_open_adressing@821}
7 = {part2_open_adressing$HashNode@831}
next = 8
Debug - Main
Debug - Main
Console
C:\Users\musta\.jdks\openjdk-16\bin\java.exe
Connected to the target VM, address: '127.0.
OpenJDK 64-Bit Server VM warning: Sharing is
Inserted items size:13 Bucket Size:20
23
42
23
23
23

Remove() method:

Main

```
System.out.println(test.remove( key: "mustafa"));
System.out.println(test.remove( key: "konya"));
System.out.println(test.remove( key: "mustaf123132a"));
System.out.println(test.remove( key: "mustaf123132a"));
System.out.println(test.remove( key: "mustaf24142421a"));
System.out.println(test.remove( key: "konya142142"));
System.out.println(test.remove( key: "mustafa41242142"));
```

Console

```
23
42
23
23
23
42
23
```