

List:
1. an Order based collection
2. index based
ArrayList
LinkedList
3. Dynamic Array

Set:
can not have duplicate elements.

Queue:
FIFO

Map:
stores data into key and value pair format

Iterator:
to iterate the elements from a collection

Common Methods:

- 1. add
- 2. addAll
- 3. remove
- 4. removeAll
- 5. size()
- 6. clear()
- 7. contains()
- 8. containsAll()
- 9. retain()
- 10. retainAll()

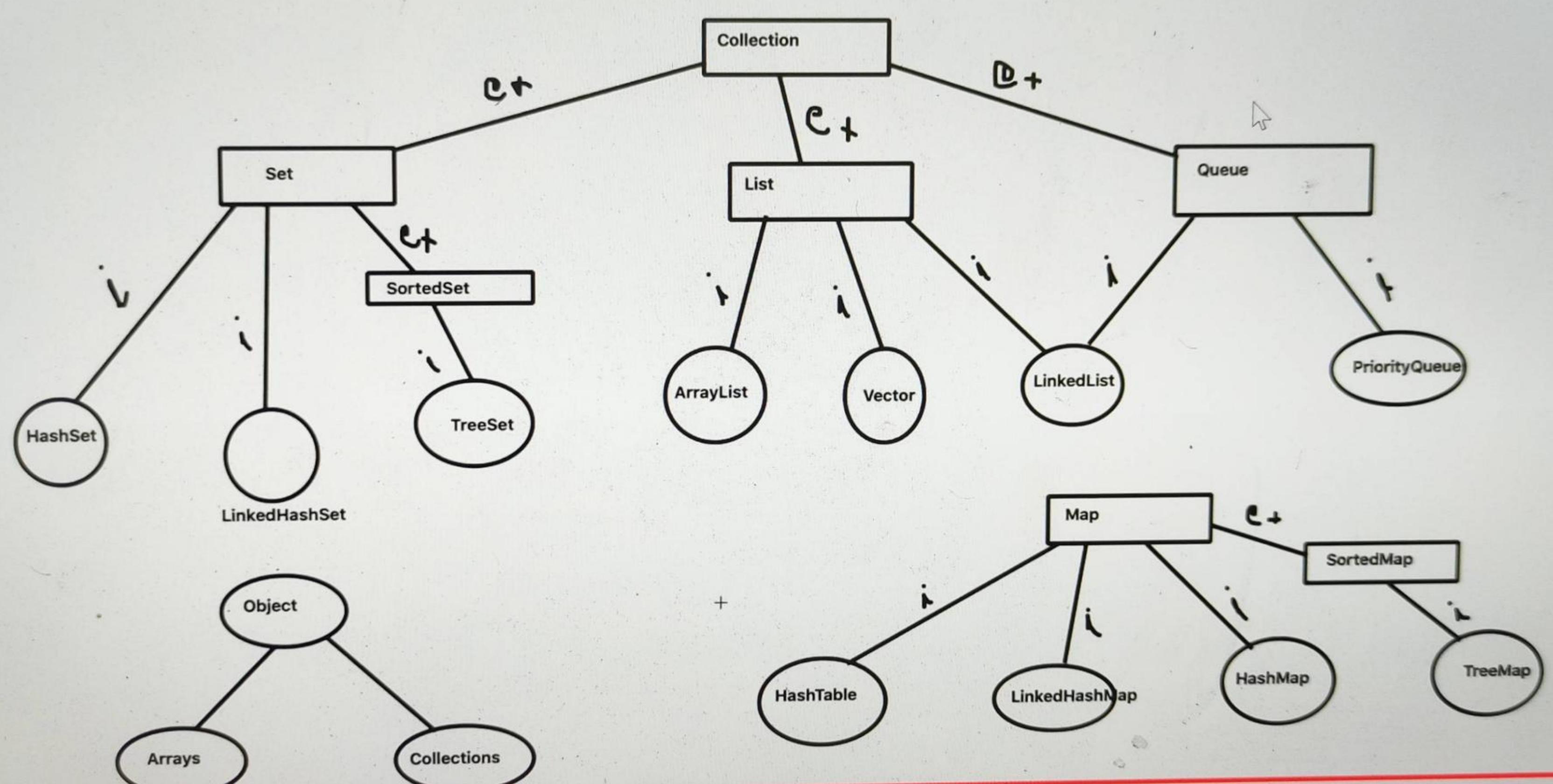
+

Common Exceptions:
Null PointerException
ClassCastException
IllegalArgumentException
IllegalStateException
UnsupportedOperationException

Advantages of Collections?

- 1. it reduces programming effort
- 2. provides in-built methods and classes
- 3. Optimized/highly performance
- 4. Increase productivity
- 5. Reduce Operational Time
- 6. Interoperability

JAVA COLLECTION FRAMEWORK - For Absolute Beginners



Collection	Ordering	Random Access	Key-Value	Duplicate Elements	Null Element	Thread Safety
ArrayList	✓	✓	✗	✓	✓	✗
LinkedList	✓	✗	✗	✓	✓	✗
HashSet	✗	✗	✗	✗	✓	✗
TreeSet	✓	✗	✗	✗	✗	✗
HashMap	✗	✓	✓	✗	✓	✗
TreeMap	✓	✓	✓	✗	✗	✗
Vector	✓	✓	✗	✓	✓	✓
Hashtable	✗	✓	✓	✗	✗	✓
Properties	✗	✓	✓	✗	✗	✓
Stack	✓	✗	✗	✓	✓	✓
CopyOnWriteArrayList	✓	✓	✗	✓	✓	✓
ConcurrentHashMap	✗	✓	✓	✗	✗	✓
CopyOnWriteArraySet	✗	✗	✗	✗	✓	✓

```
7
8
9
10 ArrayList<String> studentList = new ArrayList<String>();
11
12     studentList.add("Tom");//0
13     studentList.add("Naveen");//1
14     studentList.add("Steve");//2
15     studentList.add("Lisa");//3
16
17 //typical for loop:
18 for(int i=0; i<studentList.size(); i++) {
19     System.out.println(studentList.get(i));
20 }
21
22 System.out.println("----");
23
24 //for each loop:
25 for(String s : studentList) {
26     System.out.println(s);
27 }
28
29 System.out.println("----");
30
31 //JDK 8 - streams with lambda:
32 studentList.stream().forEach(ele -> System.out.println(ele));
33
34
```

```
35     System.out.println("----");
36 |
37 //iterator:
38 Iterator<String> it = studentList.iterator();
39
40 while(it.hasNext()) {
41     System.out.println(it.next());
42 }
43
44
45 }
46
```

Writable

```
*ArrayListMethods.java 23  System.out.println(ar1);
27 //      ar1.clear();
28 //      System.out.println(ar1);
29 //
30
31 ArrayList <String> cloneList = (ArrayList<String>)ar1.clone();
32 System.out.println(cloneList);
33
34 System.out.println(ar1.contains("python"));
35 System.out.println(ar1.contains("c"));
36 System.out.println(ar1.indexOf("ruby")>0);
37
38
39 ArrayList<String> list1 = new ArrayList<String>(Arrays.asList("Naveen", "Tom", "Naveen", "Steve", "Lisa", "Naveen"));
40 System.out.println(list1);
41
42 int i = list1.lastIndexOf("Naveen");
43 System.out.println(i);
44
45 list1.remove(1);
46 System.out.println(list1);
47
48 list1.remove("Lisa");
49 System.out.println(list1);
50
51
52 ArrayList<Integer> numbers = new ArrayList<Integer>(Arrays.asList(1,2,3,4,5,6,7,8,9,10));
53 numbers.removeIf(num -> num%2 == 0);
54 System.out.println(numbers);
```

How to synchronize ArrayList in Java:

1. Collections.synchronizedList() - method - returns synchronized list
2. copyOnWriteArrayList - class - Thread Safe variant of ArrayList

```
8 public class SynchronizedArrayList {
9
10    public static void main(String[] args) {
11
12        List<String> namesList = Collections.synchronizedList(new ArrayList<String>());
13
14        namesList.add("Java");
15        namesList.add("Python");
16        namesList.add("Ruby");
17
18        //add, remove - we dont need explicit synchronization
19
20        //to fetch/traverse the values from this list -- we have to use explicit synchronization
21
22        synchronized (namesList) {
23
24            Iterator<String> it = namesList.iterator();
25
26            while(it.hasNext()) {
27
28                System.out.println(it.next());
29            }
30
31        }
32
33
34
35
36
37    }
38
39 }
40 }
```

```
0      CopyOnWriteArrayList<String> empList = new CopyOnWriteArrayList<String>();
1      empList.add("Tom");
2      empList.add("Steve");
3      empList.add("Naveen");
4
5      //we dont need explicit synchronization for any operation: add/remove/traverse
6
7      Iterator<String> it = empList.iterator();
8      while(it.hasNext()) {
9
9          System.out.println(it.next());
10     }
11 }
```

Writable

Smart Insert

38 : 30 : 98

```
ArrayListCompare.java RemoveDuplicateElements.java
1 package ArrayListConcept;
2
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.LinkedHashSet;
6
7 public class RemoveDuplicateElements {
8
9     public static void main(String[] args) {
10
11         ArrayList<Integer> numbers = new ArrayList<Integer>(Arrays.asList(1,2,3,2,2,3,1,4,5,6,1,7,8,9,7));
12
13         //1. LinkedHashSet
14
15         LinkedHashSet<Integer> linkedHashSet = new LinkedHashSet<Integer>(numbers);
16
17         ArrayList<Integer> numbersListWithoutDuplicates = new ArrayList<Integer>(linkedHashSet);
18
19         System.out.println(numbersListWithoutDuplicates);
20
21
22     }
23
24 }
25
```

```
3 ArrayList<Integer> numbers = new ArrayList<Integer>(Arrays.asList(1,2,3,2,2,3,1,4,5,6,1,7,8,9,7));
4
5 //1. LinkedHashSet
6
7 LinkedHashSet<Integer> linkedHashSet = new LinkedHashSet<Integer>(numbers);
8
9 ArrayList<Integer> numbersListWithoutDuplicates = new ArrayList<Integer>(linkedHashSet);
10
11 System.out.println(numbersListWithoutDuplicates);
12
13 //2. JDK 8 - stream:
14
15 ArrayList<Integer> marksList = new ArrayList<Integer>(Arrays.asList(1,2,3,2,2,3,1,4,5,6,1,7,8,9,7));
16 List<Integer> marksListUnique = marksList.stream().distinct().collect(Collectors.toList());
17 System.out.println(marksListUnique);
18
19 }
20
21
22
23
24
25
26
27
28
29
30
31
32
```



```
ArrayListCompare.java 33
7 public class ArrayListCompare {
8
9     public static void main(String[] args) {
10
11         //1. sort and then equals:
12
13         ArrayList<String> l1 = new ArrayList<String>(Arrays.asList("A", "B", "C", "D", "F"));
14
15         ArrayList<String> l2 = new ArrayList<String>(Arrays.asList("A", "B", "C", "D", "E"));
16
17         ArrayList<String> l3 = new ArrayList<String>(Arrays.asList("B", "A", "C", "D", "F"));
18
19         Collections.sort(l1);
20         Collections.sort(l2);
21
22         System.out.println(l1.equals(l2)); //false
23
24         Collections.sort(l3);
25         System.out.println(l1.equals(l3)); //true
26
27
28         //2. compare two list - find out the additional elements:
29         ArrayList<String> l4 = new ArrayList<String>(Arrays.asList("A", "B", "C", "D", "F"));
30
31         ArrayList<String> l5 = new ArrayList<String>(Arrays.asList("A", "B", "C", "D", "E"));
32
33         l4.removeAll(l5);
34         System.out.println(l4);
35
36
37
38
39 }
```

```
ArrayListCompare.java  Collections.sort() <-->
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 System.out.println(l1.equals(l2));//false
22
23 Collections.sort(l3);
24 System.out.println(l1.equals(l3));//true
25
26
27
28 //2. compare two list - find out the additional elements:
29 ArrayList<String> l4 = new ArrayList<String>(Arrays.asList("A", "B", "C", "D", "F"));
30
31 ArrayList<String> l5 = new ArrayList<String>(Arrays.asList("A", "B", "C", "D", "E"));
32
33 // l4.removeAll(l5);
34 // System.out.println(l4); //f
35
36 //3. find out the missing elements:
37 l5.removeAll(l4);
38 System.out.println(l5); //E
39
40 //4. find out the common elements:
41 ArrayList<String> lang1 = new ArrayList<String>(Arrays.asList("JAVA", "Python", "Ruby", "C#", "JS"));
42
43 ArrayList<String> lang2 = new ArrayList<String>(Arrays.asList("JAVA", "Python", "Ruby", "C#", "PHP"));
44
45 lang1.retainAll(lang2);
46
47 System.out.println(lang1);
48
49
50 }
51
52 }
```

```
8
9
10    //no order - no indexing
11    //stores values -- Key-Value <k,v>
12    //key can not be duplicate
13    //can store n number of null values but only one null key
14    //hashmap is not thread-safe - unsynchronized
15
16    HashMap<String, String> capitalMap = new HashMap<String, String>();
17    capitalMap.put("India", "New Delhi");
18    capitalMap.put("USA", "Washington DC");
19    capitalMap.put("UK", "London");
20    capitalMap.put("UK", "London11");
21    capitalMap.put(null, "Berlin");
22    capitalMap.put(null, "LA");
23    capitalMap.put("Russia", null);
24    capitalMap.put("France", null);
25
26
27    System.out.println(capitalMap.get("USA"));
28    System.out.println(capitalMap.get("UK"));
29
30    System.out.println(capitalMap.get(null));
31    System.out.println(capitalMap.get("France"));
32
33
34
```

```
System.out.println(capitalMap.get("UK"));

System.out.println(capitalMap.get(null));
System.out.println(capitalMap.get("France"));

//iterator:
Iterator<String> it = capitalMap.keySet().iterator();

while(it.hasNext()) {
    String key = it.next();
    String value = capitalMap.get(key);
    System.out.println("key = " + key + " value = " + value);
}
```

Smart Insert

```
32 System.out.println(capitalMap.get(null));
33 System.out.println(capitalMap.get("France"));
34
35 //iterator: over the keys: by using keySet()
36 Iterator<String> it = capitalMap.keySet().iterator();
37
38 while(it.hasNext()) {
39     String key = it.next();
40     String value = capitalMap.get(key);
41     System.out.println("key = " + key + " value = " + value);
42 }
43
44 System.out.println("-----");
45 //iterator: over the set (pair): by using entrySet()
46
47 Iterator<Entry<String, String>> it1 = capitalMap.entrySet().iterator();
48
49 while(it1.hasNext()) {
50     Entry<String, String> entry = it1.next();
51     System.out.println("key = " + entry.getKey() + " and value = " + entry.getValue());
52 }
53
54
55
56 }
57 }
```

```
47     Iterator<Entry<String, String>> it1 = capitalMap.entrySet().iterator();
48
49     while(it1.hasNext()) {
50         Entry<String, String> entry = it1.next();
51         System.out.println("key = " + entry.getKey() + " and value = " + entry.getValue());
52     }
53     System.out.println("-----");
54
55     //iterate hashmap using java 8 for each and lambda:
56     capitalMap.forEach((k,v) -> System.out.println("key = " + k + " and value = " + v));
57 }
58
59 }
```

Writable Smart Insert 56 : 30 - 1669

```
8     HashMap<Integer, String> map1 = new HashMap<Integer, String>();
9
10    map1.put(1, "A");
11    map1.put(2, "B");
12    map1.put(3, "C");
13
14    HashMap<Integer, String> map2 = new HashMap<Integer, String>();
15
16    map2.put(3, "C");
17    map2.put(1, "A");
18    map2.put(2, "B");
19
20    HashMap<Integer, String> map3 = new HashMap<Integer, String>();
21
22    map3.put(1, "A");
23    map3.put(2, "B");
24    map3.put(3, "C");
25    map3.put(3, "D");
26
27
28 //1. on the basis of key-value: use equals method:
29 System.out.println(map1.equals(map2)); //true
30 System.out.println(map1.equals(map3)); //false
31
32
33 //2. compare hashmaps for the same keys: keySet():
34 System.out.println(map1.keySet().equals(map2.keySet())); //true
35 System.out.println(map1.keySet().equals(map3.keySet()));
36
37
38
39
40
```

Writable Smart Insert 11 : 18 [41]

```

23
24     map3 = new HashMap<Integer, String>();
25     map3.put(1, "A");
26     map3.put(2, "B");
27     map3.put(3, "C");
28     map3.put(3, "D");
29
30 //1. on the basis of key-value: use equals method:
31 System.out.println(map1.equals(map2)); //true
32 System.out.println(map1.equals(map3)); //false
33
34
35 //2. compare hashmaps for the same keys: keySet():
36 System.out.println(map1.keySet().equals(map2.keySet())); //true
37 System.out.println(map1.keySet().equals(map3.keySet())); //true
38
39 //3. find out the extra keys:
40 HashMap<Integer, String> map4 = new HashMap<Integer, String>();
41
42 map4.put(1, "A");
43 map4.put(2, "B");
44 map4.put(3, "C");
45 map4.put(4, "D");
46
47 //combine the keys from both the maps: using HashSet:
48 HashSet<Integer> combineKeys = new HashSet<>(map1.keySet());
49 //add the keyset from map4:
50 combineKeys.addAll(map4.keySet());
51 combineKeys.removeAll(map1.keySet());
52 System.out.println(combineKeys);
53

```

Scanned by TapScanner

```

<terminated> HashMapCompare [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_261/jdk/Co
true
false
true
true
[4]

```

```
52 combineKeys.removeAll(map1.keySet());
53 System.out.println(combineKeys);
54
55 //4. compare maps by values:
56 HashMap<Integer, String> map5 = new HashMap<Integer, String>();
57 map5.put(1, "A");
58 map5.put(2, "B");
59 map5.put(3, "C");
60
61 HashMap<Integer, String> map6 = new HashMap<Integer, String>();
62 map6.put(4, "A");
63 map6.put(5, "B");
64 map6.put(6, "C");
65
66 HashMap<Integer, String> map7 = new HashMap<Integer, String>();
67 map7.put(1, "A");
68 map7.put(2, "B");
69 map7.put(3, "C");
70 map7.put(4, "C");
71
72 //1. duplicates are not allowed:
73 System.out.println(new ArrayList<>(map5.values()).equals(new ArrayList<>(map6.values()))); //true
74 System.out.println(new ArrayList<>(map5.values()).equals(new ArrayList<>(map7.values()))); //false
```

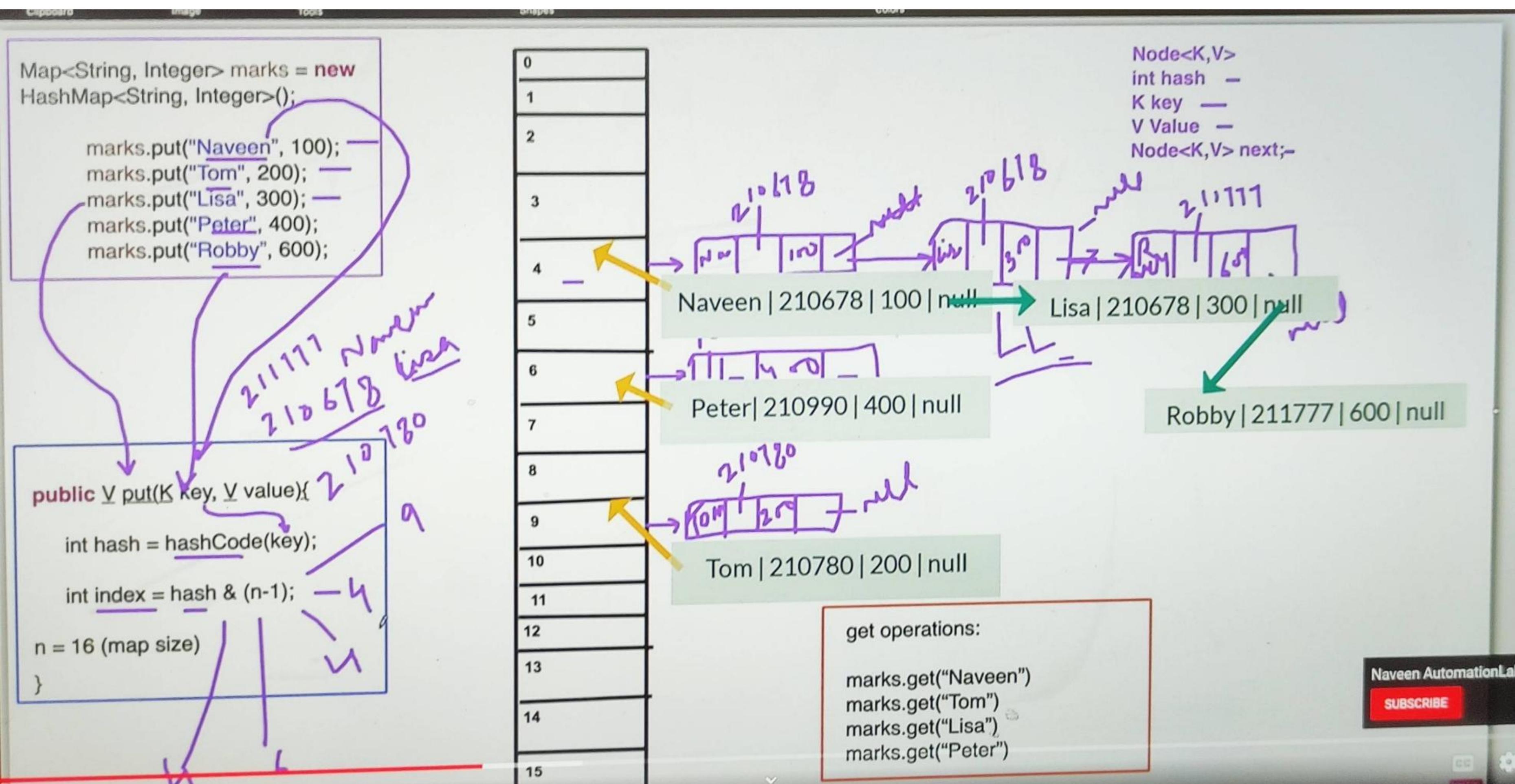
```
map6.put(6, "C");

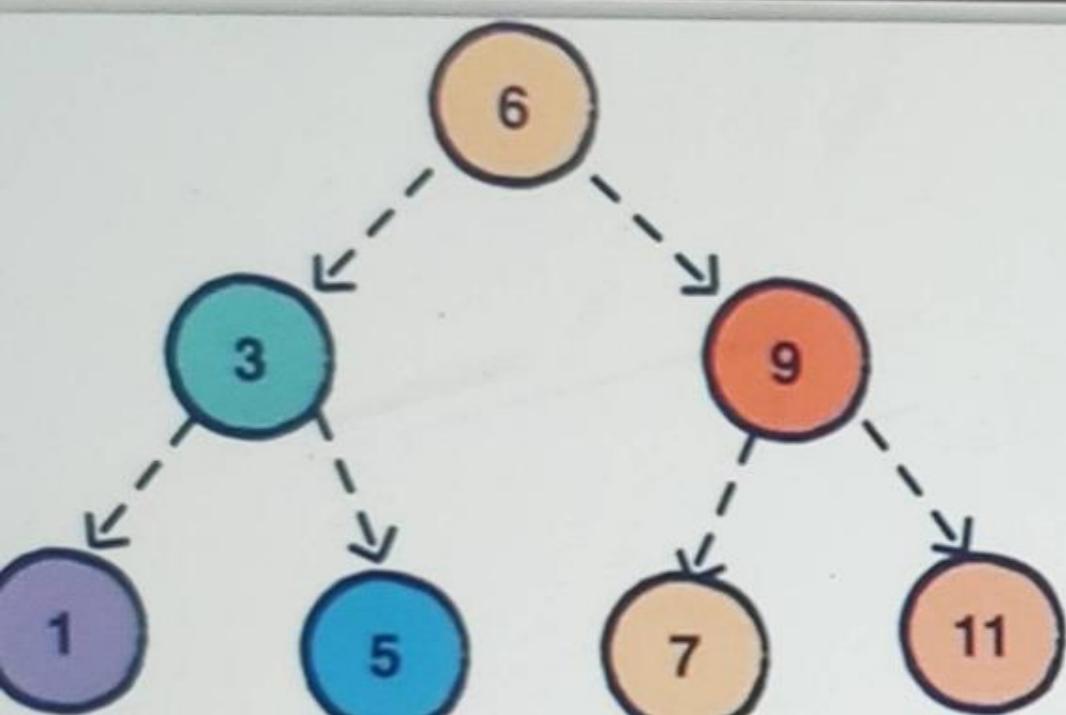
HashMap<Integer, String> map7 = new HashMap<Integer, String>();
map7.put(1, "A");
map7.put(2, "B");
map7.put(3, "C");
map7.put(4, "C");

//1. duplicates are not allowed: using ArrayList
System.out.println(new ArrayList<>(map5.values()).equals(new ArrayList<>(map6.values()))); //true
System.out.println(new ArrayList<>(map5.values()).equals(new ArrayList<>(map7.values()))); //false

//2. duplicates are allowed: Using HashSet
System.out.println(new HashSet<>(map5.values()).equals(new HashSet<>(map6.values()))); //true
System.out.println(new HashSet<>(map5.values()).equals(new HashSet<>(map7.values()))); //true
```

Writable Smart Insert 77 : 106 [5]





An example of a balanced tree

As we know now that in case of hash collision entry objects are stored as a node in a [linked-list](#) and `equals()` method is used to compare keys. That comparison to find the correct key with in a linked-list is a linear operation so in a worst case scenario the complexity becomes $O(n)$.

JDK 8 HashMap Changes

To address this issue, Java 8 hash elements use balanced trees instead of linked lists after a certain threshold is reached. Which means HashMap starts with storing Entry objects in linked list but after the number of items in a hash becomes larger than a certain threshold, the hash will change from using a linked list to a balanced tree, which will improve the worst case performance from $O(n)$ to $O(\log n)$.

Important Points

1. Time complexity is almost constant for put and get method until rehashing is not done.
2. In case of collision, i.e. index of two or more nodes are same, nodes are joined by link list i.e. second node is referenced by first node and third by second and so on.
3. If key given already exist in HashMap, the value is replaced with new value.
4. hash code of null key is 0.
5. When getting an object with its key, the linked list is traversed until the key matches or null is found on next field.

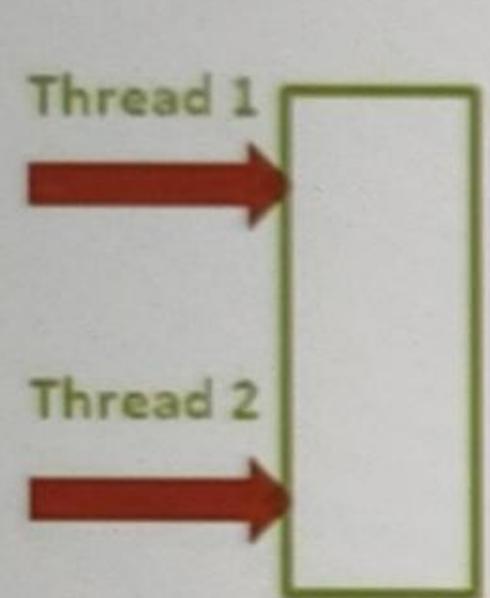
1-Different ways of creating HashMap in Java

```
24 //1. using HashMap class
25
26     HashMap<String, String> map1 = new HashMap<>();
27     Map<String, String> map2 = new HashMap<>();
28
29     //2. static way : static hashmap:
30     System.out.println(HashMapInitialization.marksMap.get("A"));
31
32     //3. immutableMap with only one single entry:
33     Map<String, Integer> map3 = Collections.singletonMap("test", 100);
34     System.out.println(map3.get("test"));
35     //map3.put("abc", 200); //UnsupportedOperationException
36
37     //4. JDK 8:
38     //creating one 2D array of Strings using Stream and collecting in the form Map
39     Map<String, String> map4 = Stream.of(new String[][] {
40
41         {"Tom", "A Grade"},
42         {"Naveen", "A+ Grade"},
43     }).collect(Collectors.toMap(data -> data[0], data -> data[1]));
44
45     System.out.println(map4.get("Tom"));
46     map4.put("Lisa", "A++ Grade");
47     System.out.println(map4.get("Lisa"));
48
49
50     //using SimpleEntry: mutable map:
51     Map<String, String> map5 = Stream.of(
52         new AbstractMap.SimpleEntry<>("Naveen", "Java"),
53         new AbstractMap.SimpleEntry<>("Tom", "Python")
54     ).collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue));
55
56     System.out.println(map5.get("Naveen"));
57
```

```
50 //using SimpleEntry: mutable map:  
51 Map<String, String> map5 = Stream.of(  
52     new AbstractMap.SimpleEntry<>("Naveen", "Java"),  
53     new AbstractMap.SimpleEntry<>("Tom", "Python")  
54 ).collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue));  
55  
56 System.out.println(map5.get("Naveen"));  
57 map5.put("Lisa", "C#");  
58 System.out.println(map5.get("Lisa"));  
59  
60 //using SimpleEntry: Immutable map:  
61  
62 Map<String, String> map6 = Stream.of(  
63     new AbstractMap.SimpleImmutableEntry<>("Naveen", "Java"),  
64     new AbstractMap.SimpleImmutableEntry<>("Tom", "Python")  
65 ).collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue));  
66  
67 System.out.println(map6.get("Tom"));  
68 map6.put("Steve", "Ruby");  
69 System.out.println(map6.get("Steve"));  
70  
71  
72 //JDK 1.9:  
73 //empty map:  
74 Map<String, String> emptyMap = Map.of();  
75 //emptyMap.put("Tom", "Python");  
76 //System.out.println(emptyMap.get("Tom")); //UnsupportedOperationException  
77  
78 Map<String, String> singtomMap Map.of("k1", "v1");  
79  
80  
81  
82  
83 }  
84
```

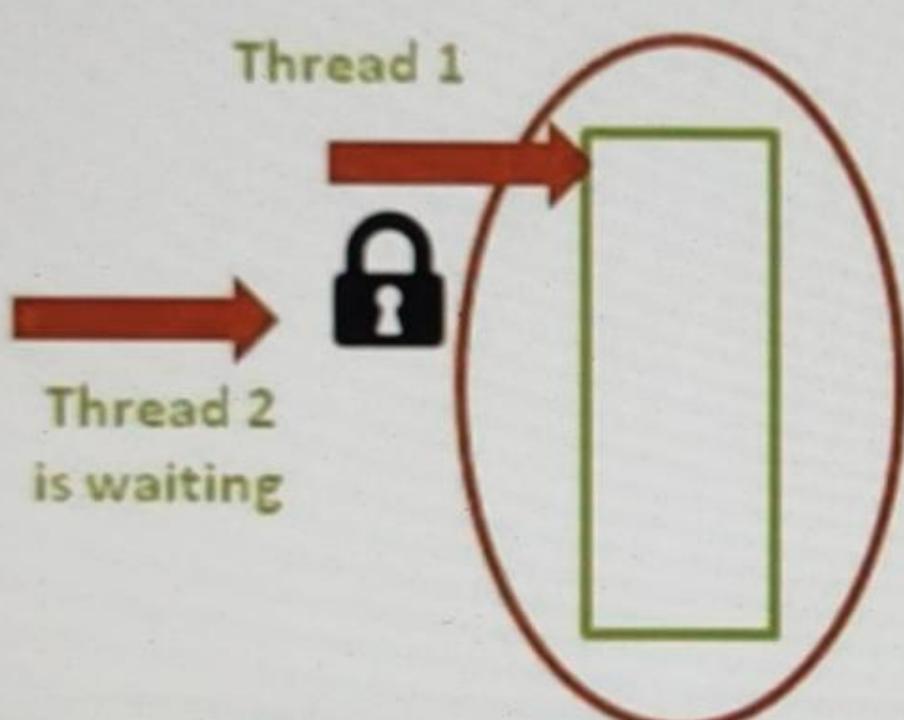
```
#11-Different ways of creating HashMap in Java
77     //empty map:
78     Map<String, String> emptyMap = Map.of();
79     //emptyMap.put("Tom", "Python");
80     //System.out.println(emptyMap.get("Tom")); //UnsupportedOperationException
81
82     //singleton map:
83     Map<String, String> singletonMap = Map.of("k1", "v1");
84     System.out.println(singletonMap.get("k1"));
85     //singletonMap.put("k2", "v2"); //UnsupportedOperationException
86
87     //multi Values Map: max 10 pairs can be stored:
88     Map<String, String> multipMap = Map.of("k1", "v1", "k2", "v2", "k3", "v3");
89     System.out.println(multipMap.get("k3"));
90
91     //ofEntries method: no limitations on pairs (key-value)
92     //Immutable Maps:
93     Map<String, Integer> map7 = Map.ofEntries(
94         new AbstractMap.SimpleEntry<>("A", 100),
95         new AbstractMap.SimpleEntry<>("B", 200),
96         new AbstractMap.SimpleEntry<>("C", 300)
97     );
98
99
100    System.out.println(map7.get("C"));
101    //map7.put("D", 400); //UnsupportedOperationException
102
103
104
105    //maps using Guava:
106    Map<String, String> titleMaps = ImmutableMap.of("Google", "Google India", "Amazon", "Amazon Shopping");
107    System.out.println(titleMaps.get("Amazon"));
108    //titleMaps.put("Rediff", "Rediff India"); //UnsupportedOperationException
109
110 }
111
112 }
113
```

HashMap



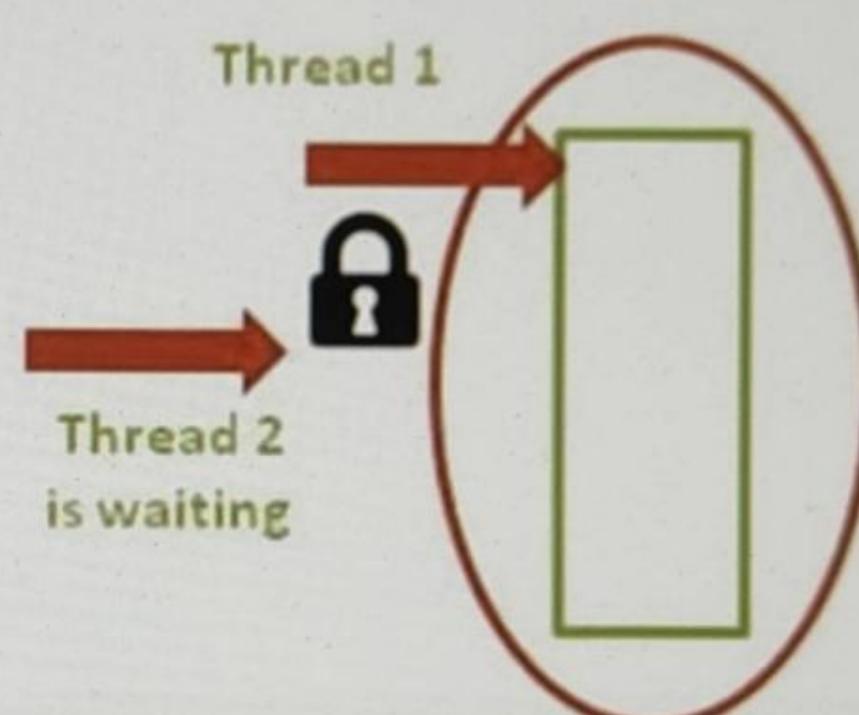
Not Thread-Safe.
Can have one null
key and multiple
null values

Hashtable



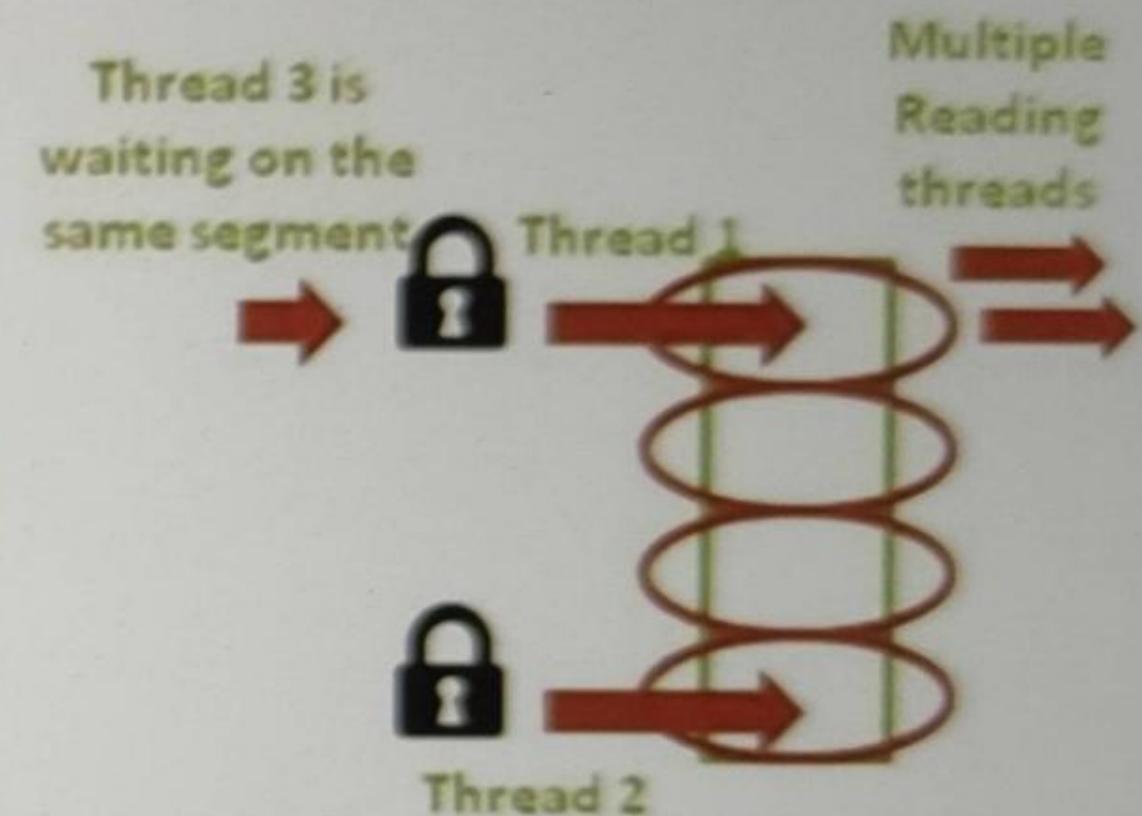
Thread-Safe.
Slow Performance.
null key and values
are not allowed

SynchronizedMap



Thread-Safe.
Slow Performance.
null key and
multiple null values
are allowed

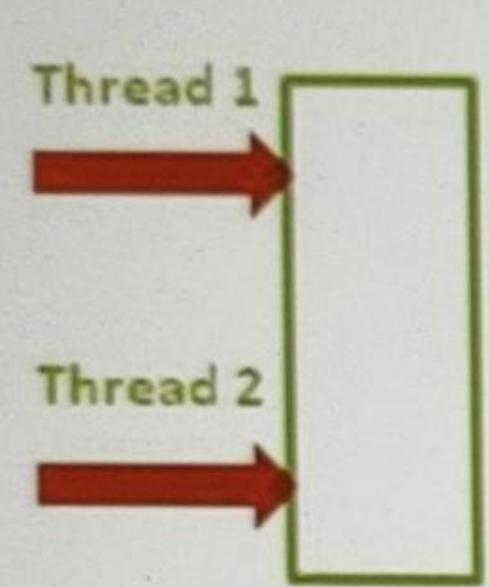
ConcurrentHashMap



Thread-Safe.
Fast Performance.
null key and values
are not allowed

```
8 public class HashMapSync {
9
10    public static void main(String[] args) {
11        //synchronizedMap method in collections class:
12        Map<String, String> map1 = new HashMap<String, String>();
13
14        map1.put("1", "Naveen");
15        map1.put("2", "Tom");
16        map1.put("3", "Lisa");
17
18
19
20        //create synchronizedMap:
21        Map<String, String> syncMap = Collections.synchronizedMap(map1);
22
23        System.out.println(syncMap);
24
25        //concurrentHashMap: it does not throw any ConcurrentModificationException
26
27        ConcurrentHashMap<String, String> concurrentMap = new ConcurrentHashMap<>();
28
29        concurrentMap.put("A", "Java");
30        concurrentMap.put("B", "Python");
31        concurrentMap.put("C", "Ruby");
32
33        System.out.println(concurrentMap.get("A"));
34
35
36
37    }
38
39 }
```

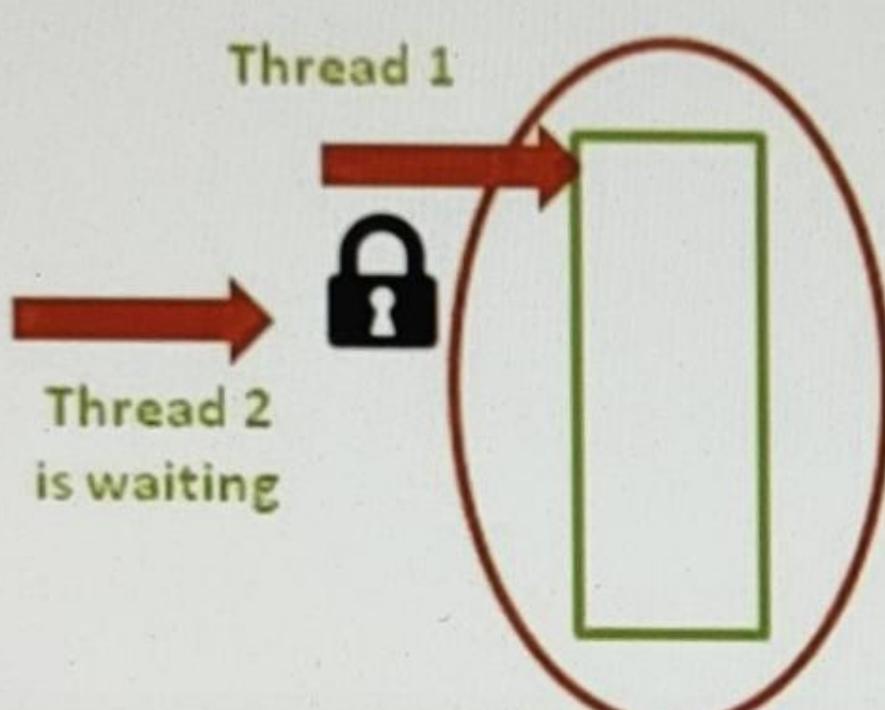
HashMap



Not Thread-Safe.
Can have one null

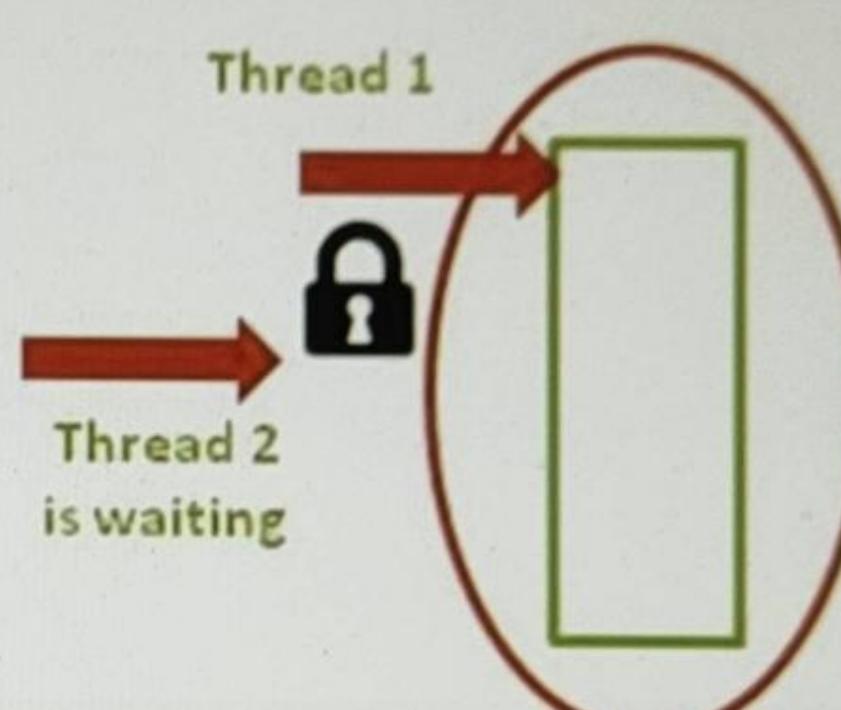
Can Multiple thread write in the same segment?

Hashtable



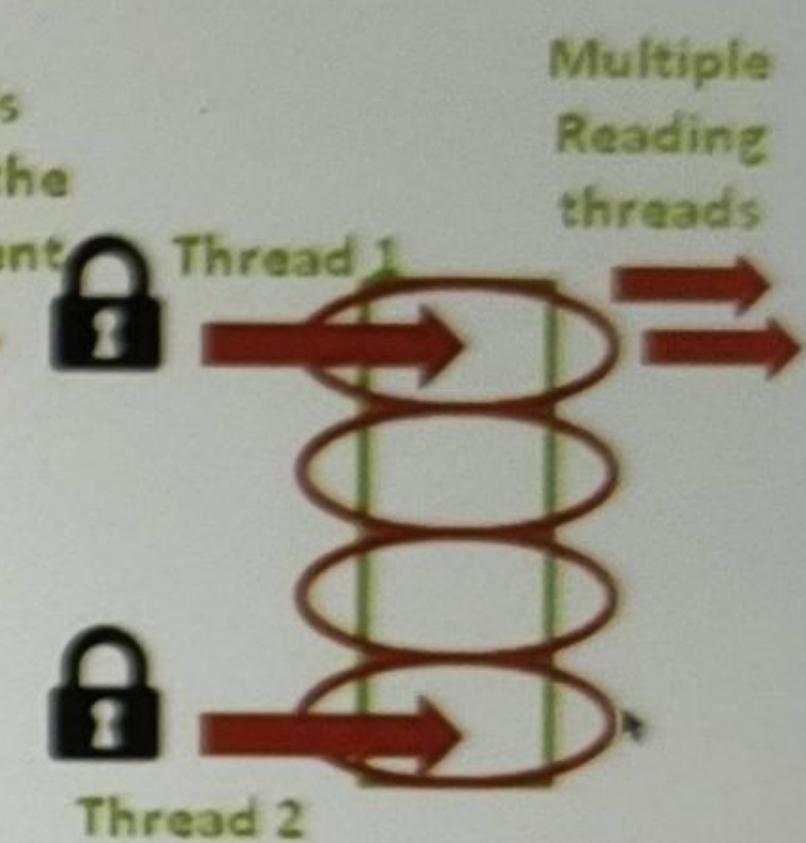
Thread-Safe.
Slow Performance.

SynchronizedMap



Thread-Safe.
Slow Performance.

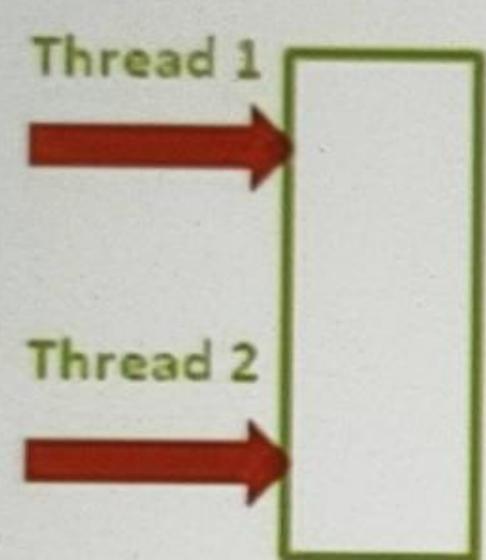
ConcurrentHashMap



Thread-Safe.
Fast Performance.

No. Thread acquires a lock on segment in put() operation and at a time only one thread can write in that segment.

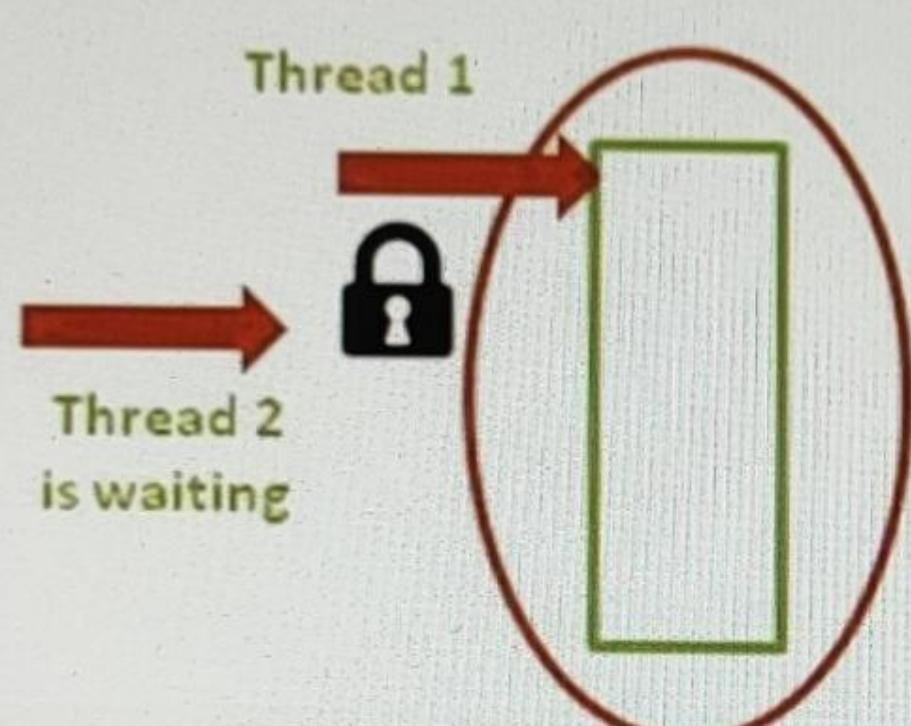
HashMap



Not Thread-Safe.

Can have or
key and mu
null value

Hashtable

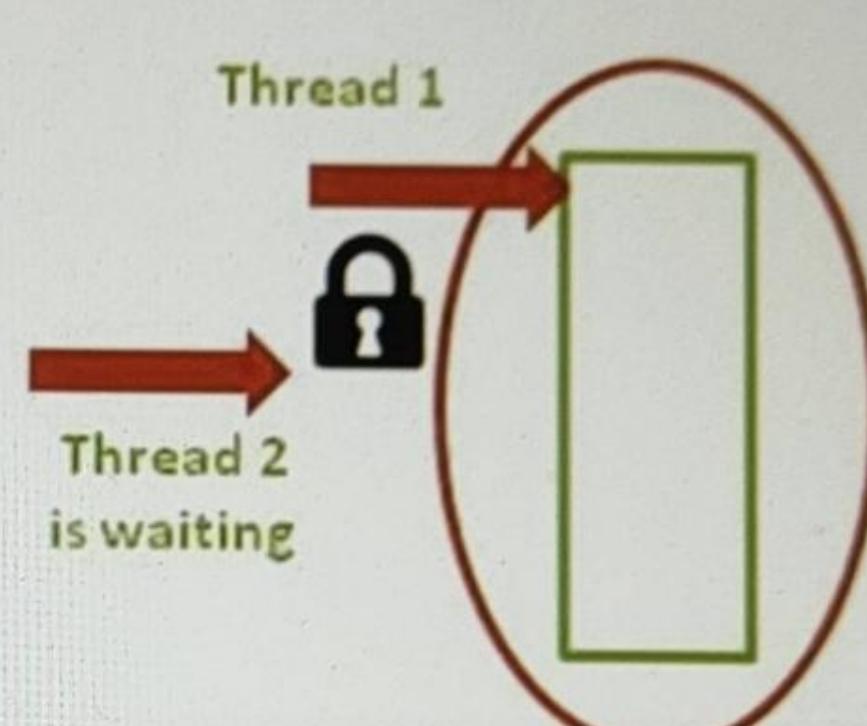


Thread-Safe.

Can two threads write in the different segment?

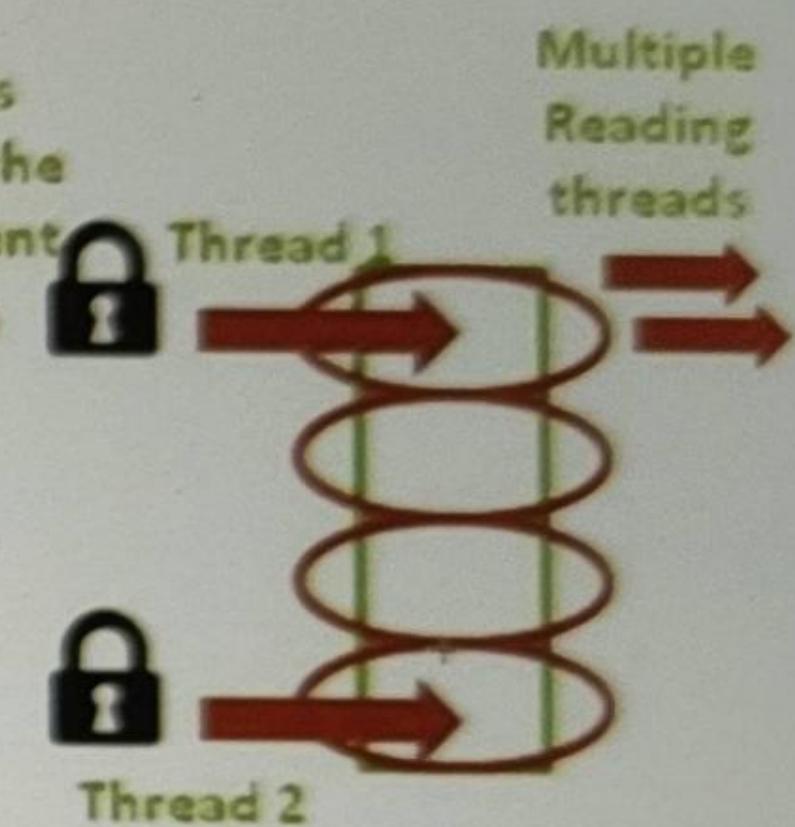
Yes. Two threads are allowed to write concurrently in different segments.

SynchronizedMap



Thread-Safe.

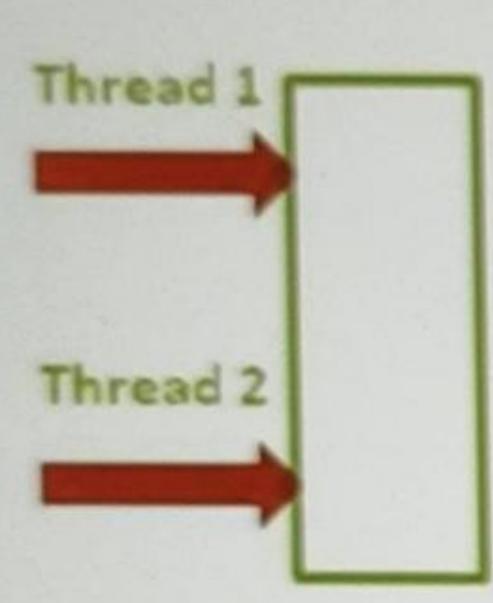
Thread 3 is
waiting on the
same segment



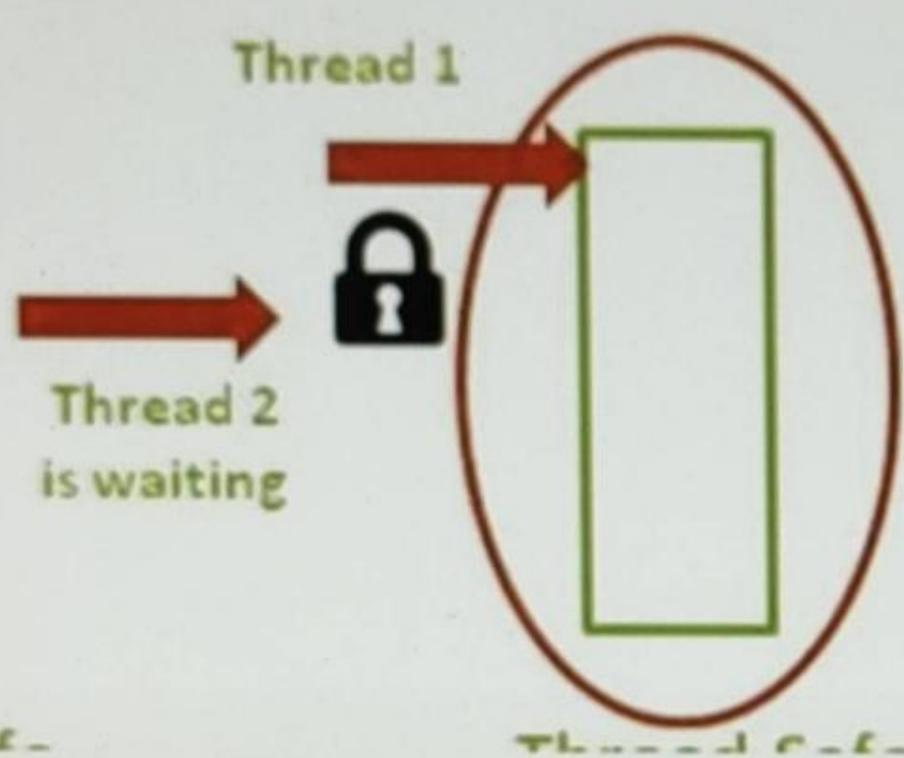
Thread-Safe.

Fast Performance,
null key and values
are not allowed

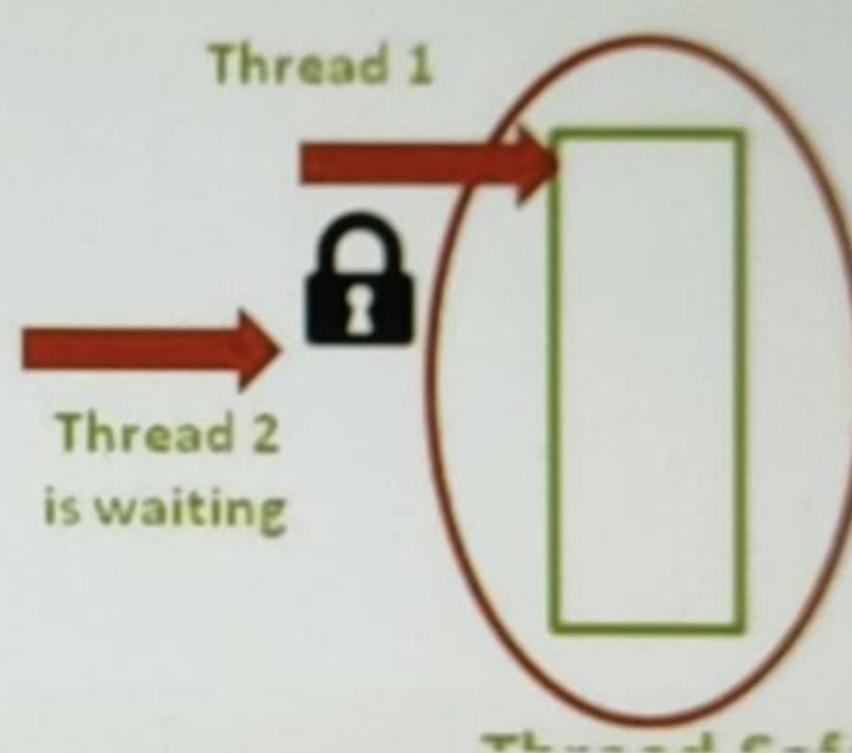
HashMap



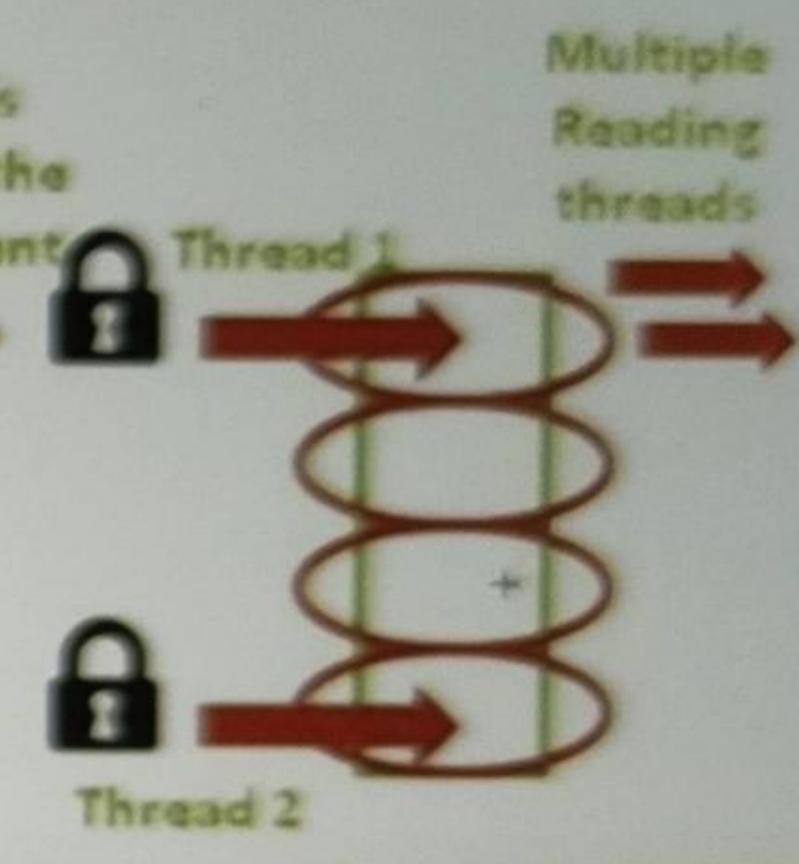
Hashtable



SynchronizedMap

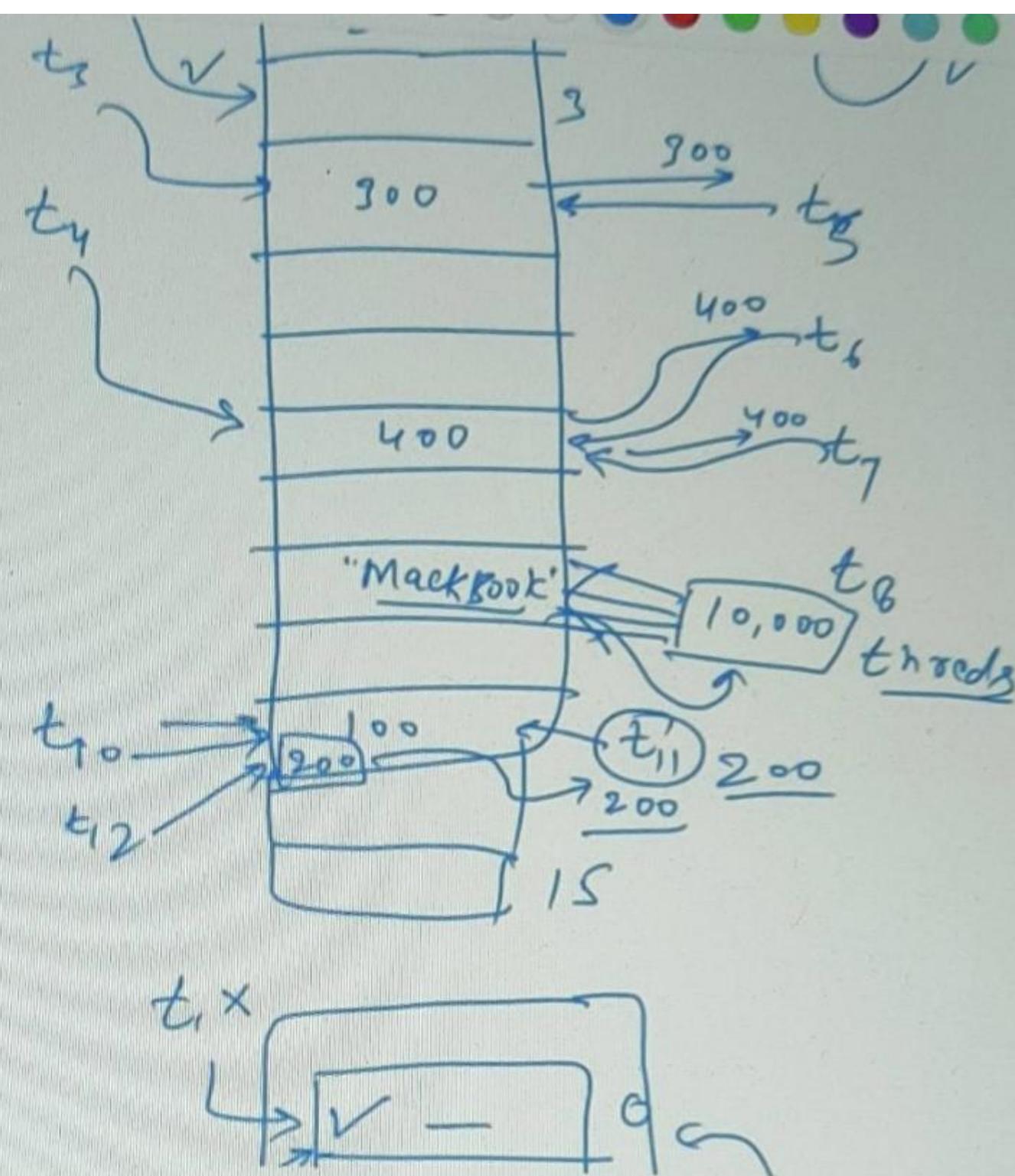


ConcurrentHashMap



Can Multiple thread read from the same segment?

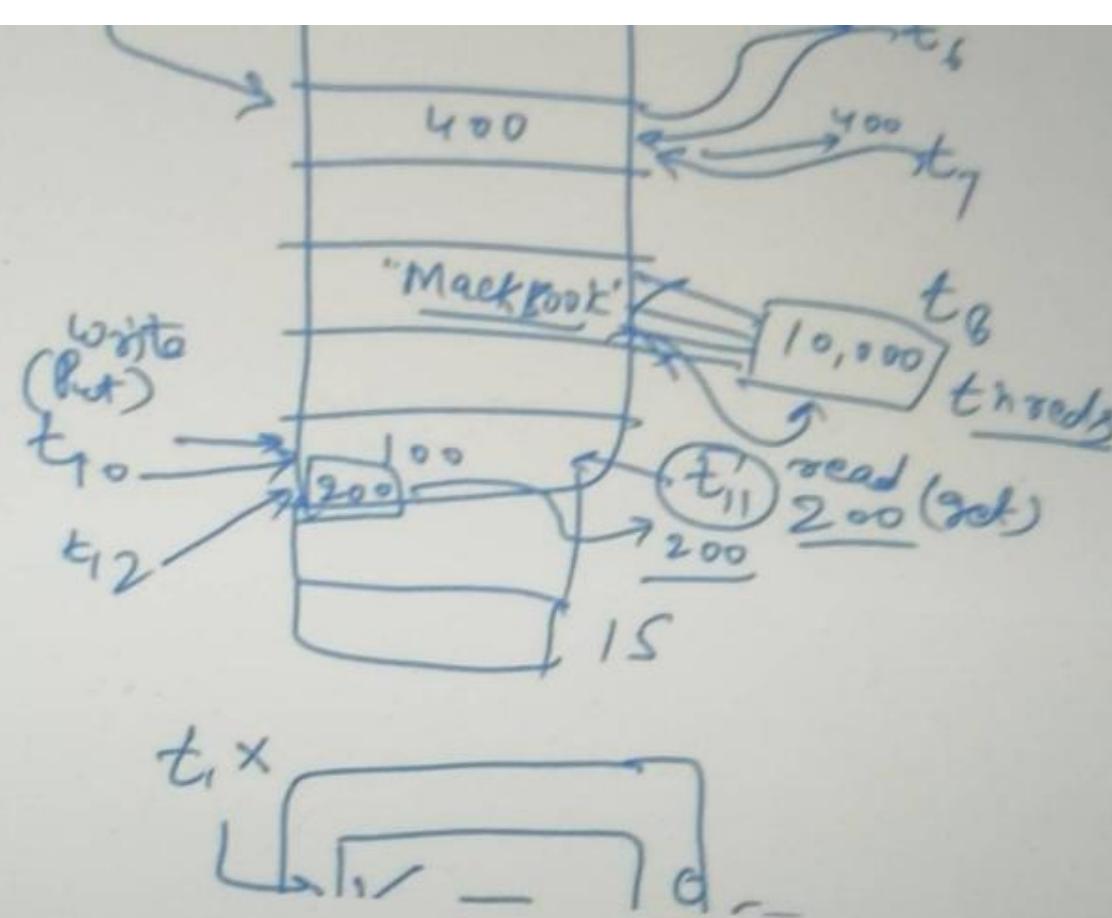
Yes. Thread doesn't acquire a lock on segment in get() operation and any number of threads can read from the same segment.



If one thread is writing in a segment, can another thread read from that segment(?)?

Yes. but in this case last updated value will be seen by the reading thread.

DELL



Null keys and values

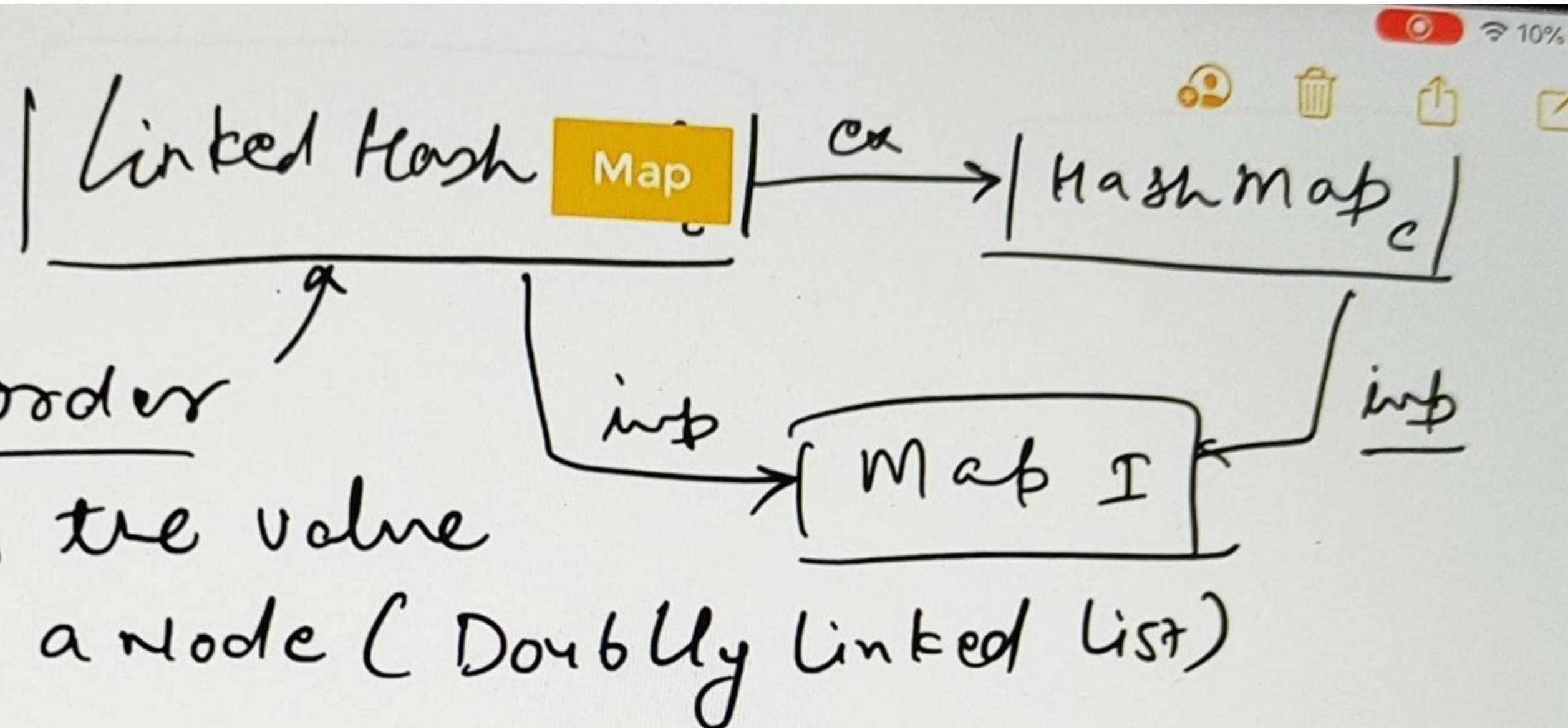
ConcurrentHashMap doesn't allow null keys and null values.

◀ ▶ ⏪ 21:35 / 22:10

```
12  
13     HashMap<String, Integer> compMap = new HashMap<String, Integer>();  
14     compMap.put("Google", 10000);  
15     compMap.put("Walmart", 20000);  
16     compMap.put("Amazon", 30000);  
17     compMap.put("Facebook", 5000);  
18     compMap.put("Cisco", 15000);  
19  
20     System.out.println("comp map size: " + compMap.size());  
21  
22     Iterator it = compMap.entrySet().iterator();  
23  
24     while(it.hasNext()) {  
25         Map.Entry pairs = (Map.Entry)it.next();  
26         System.out.println(pairs.getKey() + " = " + pairs.getValue());  
27     }  
28  
29     //convert hashmap keys into ArrayList:  
30     List<String> compNamesList = new ArrayList<String>(compMap.keySet());  
31     System.out.println(compNamesList);  
32     for(String t : compNamesList) {  
33         System.out.println(t);  
34     }  
35  
36     //convert hashmap values into ArrayList:  
37     List<Integer> EmpValuesList = new ArrayList<Integer>(compMap.values());  
38  
39  
40  
41 }  
42  
43 }  
44
```

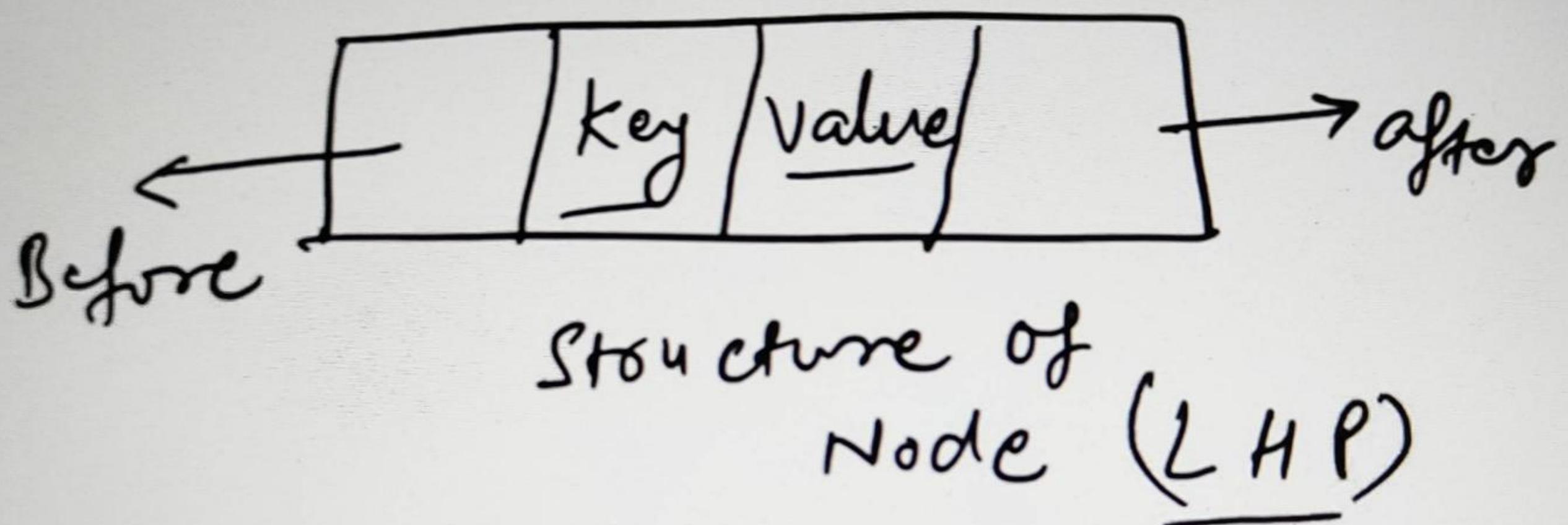
Type mismatch: cannot convert from ArrayList<String> to List<Integer>

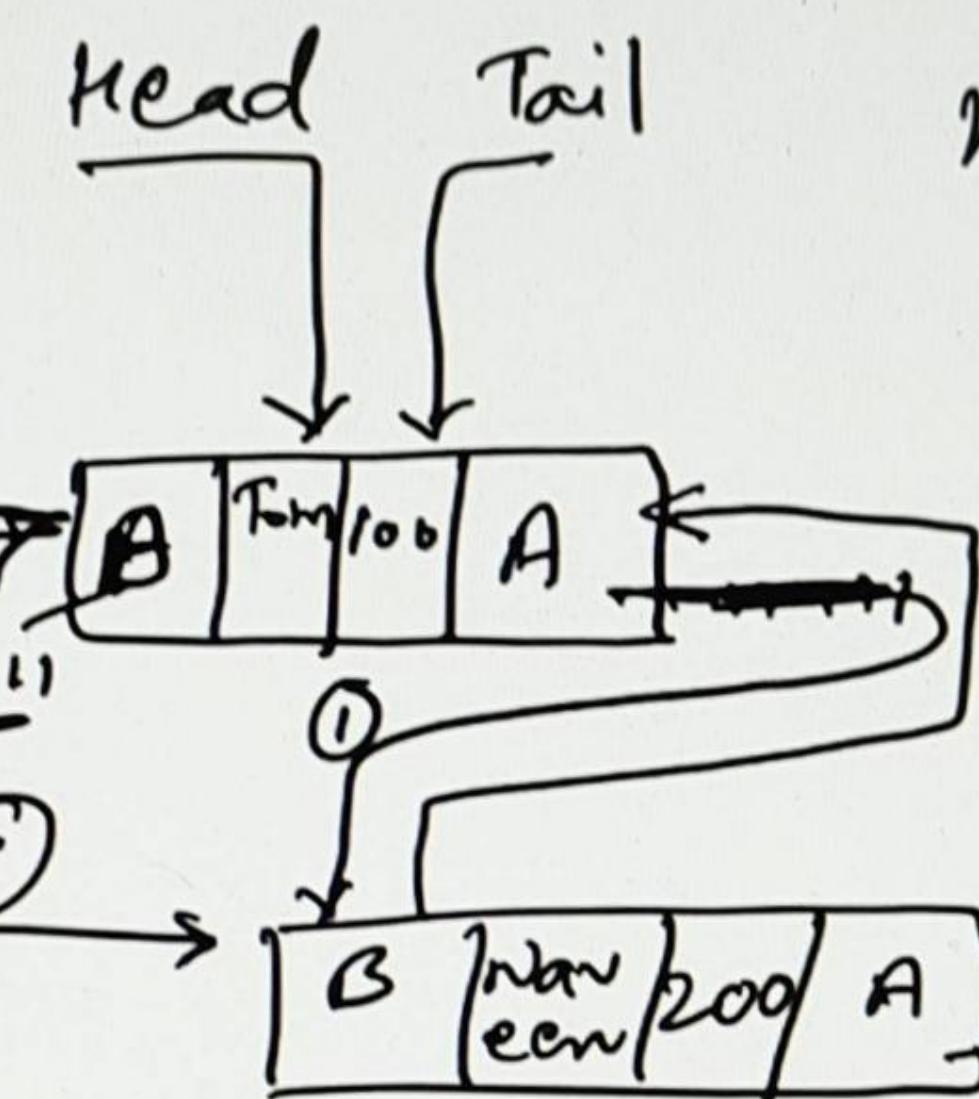
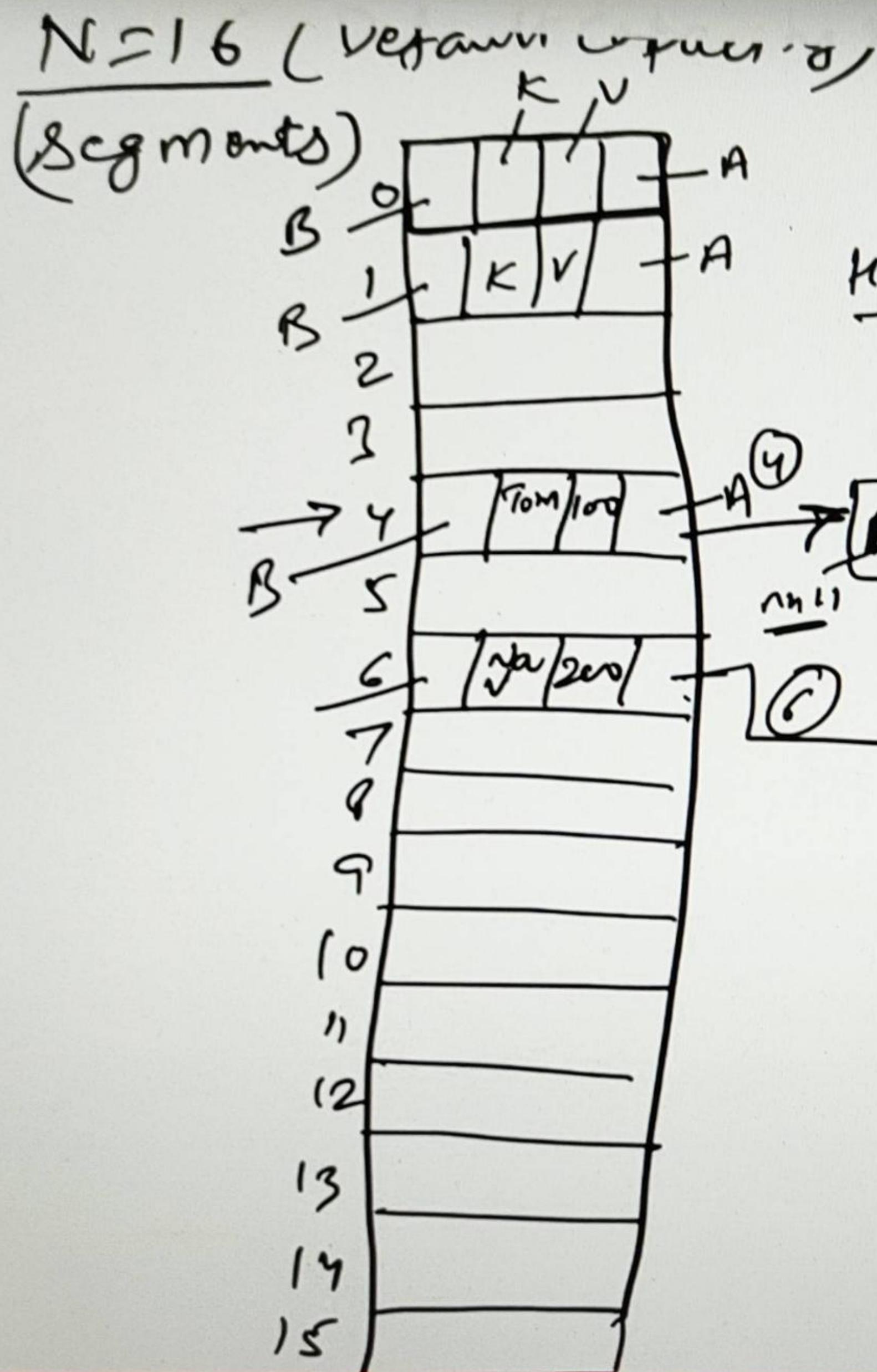
```
12
13     HashMap<String, Integer> compMap = new HashMap<String, Integer>();
14     compMap.put("Google", 10000);
15     compMap.put("Walmart", 20000);
16     compMap.put("Amazon", 30000);
17     compMap.put("Facebook", 5000);
18     compMap.put("Cisco", 15000);
19
20     System.out.println("comp map size: " + compMap.size());
21
22     Iterator it = compMap.entrySet().iterator();
23
24     while(it.hasNext()) {
25         Map.Entry pairs = (Map.Entry)it.next();
26         System.out.println(pairs.getKey() + " = " + pairs.getValue());
27     }
28
29     System.out.println("-----");
30
31     compMap.forEach((k,v) -> System.out.println("key =" + k + "value = " + v));
32
33     //convert hashmap keys into ArrayList:
34     List<String> compNamesList = new ArrayList<String>(compMap.keySet());
35     System.out.println("total comp count = " + compNamesList.size());
36     for(String t : compNamesList) {
37         System.out.println(t);
38     }
39
40     System.out.println("-----");
41
42     //convert hashmap values into ArrayList:
43     List<Integer> EmpValuesList = new ArrayList<Integer>(compMap.values());
44     System.out.println("Total emp values count = " + EmpValuesList.size());
```



- ① Insertion order
↳ stores the value
in a node (Doubly linked list)
- ②. <k-v>
- ③. not Sync (not Thread-Safe)
- ④. one null key & multiple null values.

$$\underline{i/p} \left\{ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \right\} \rightarrow \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \underline{o/p}$$





```

int h = hashCode(key);
int index = m % h;
hashCode("Tom");
h = 170268
index = 4
hashCode("Naveen")
h = 170280
index = 6
    
```



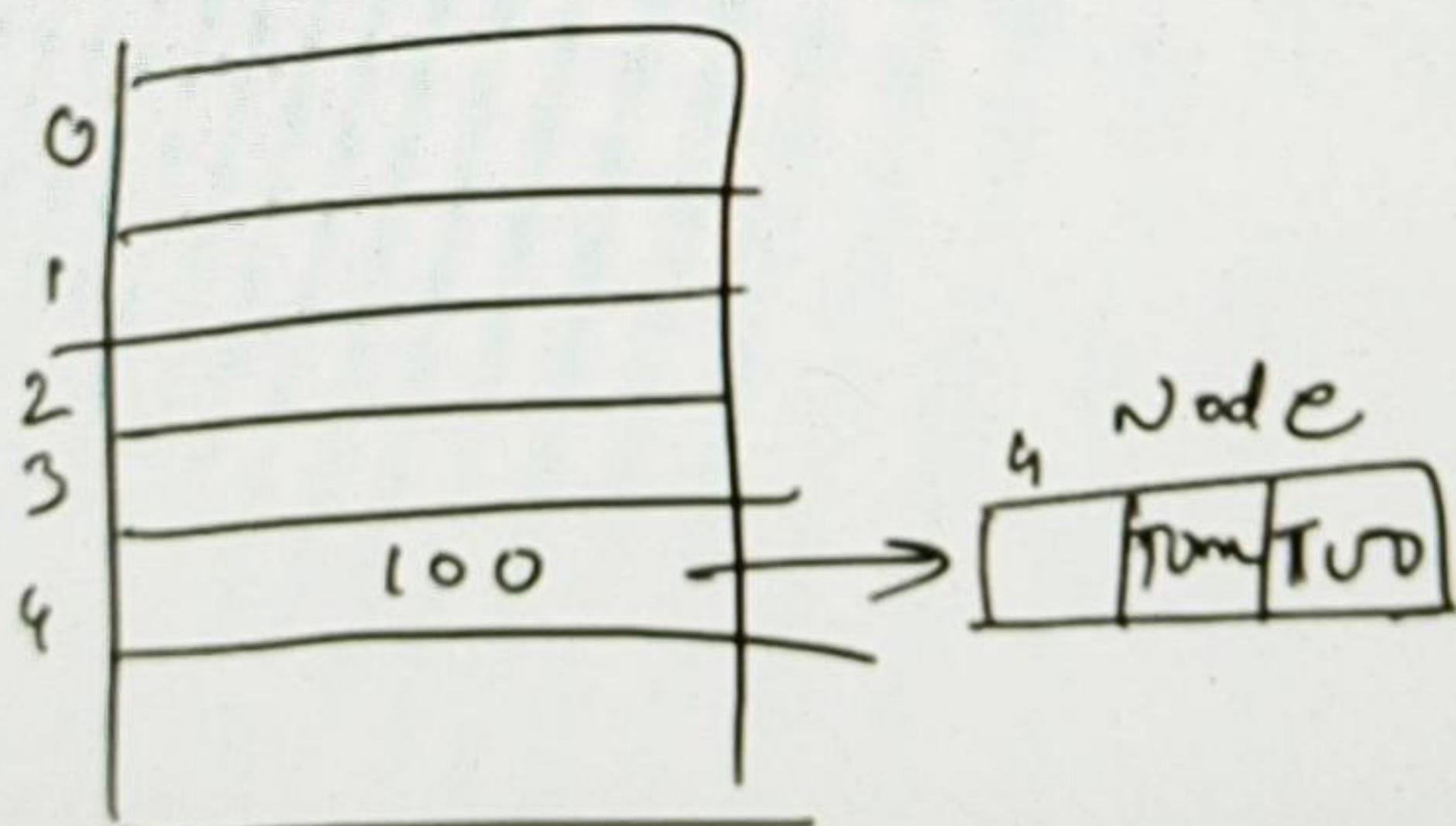
HashMap Top Interview Questions

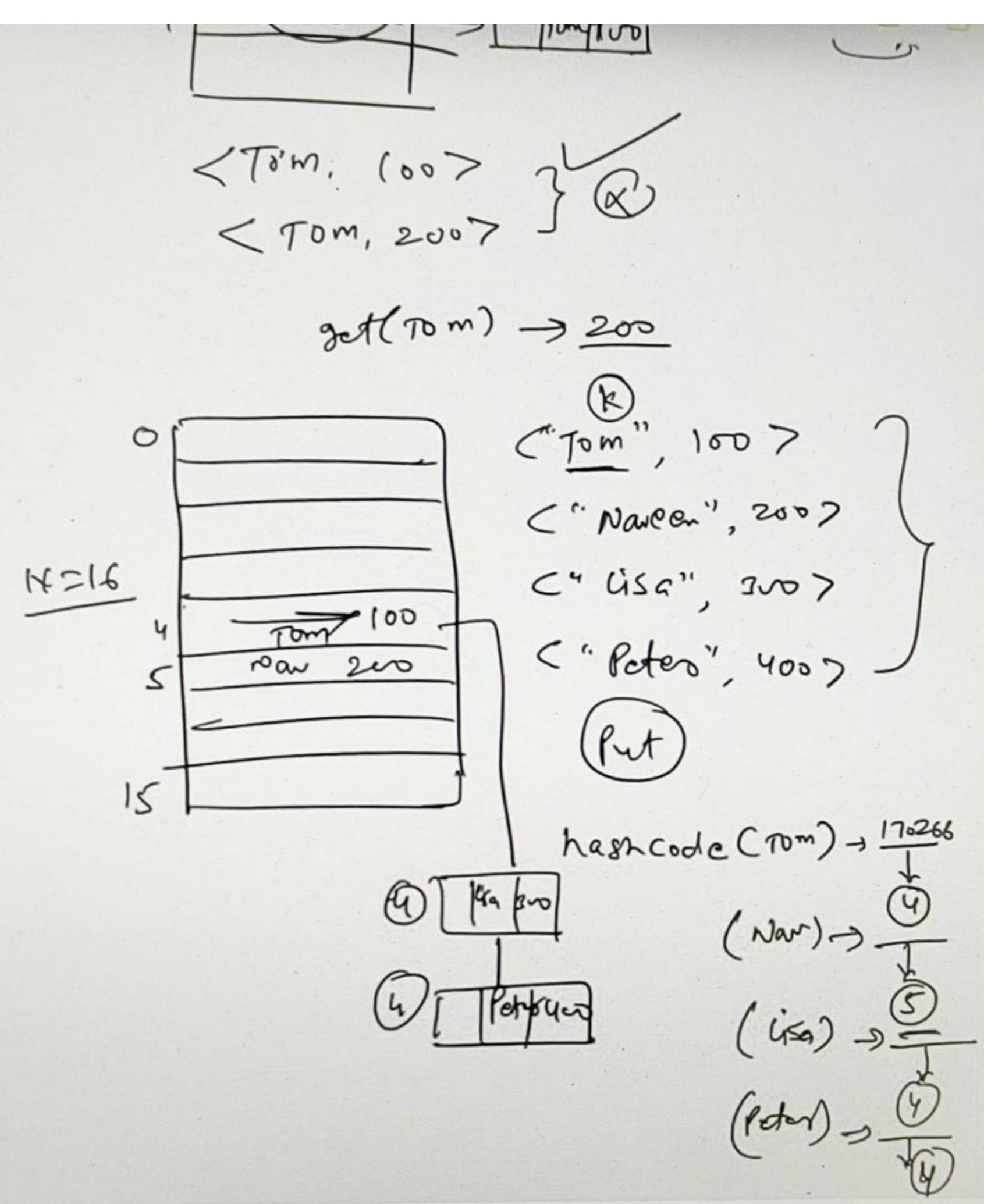
<Key, Value>

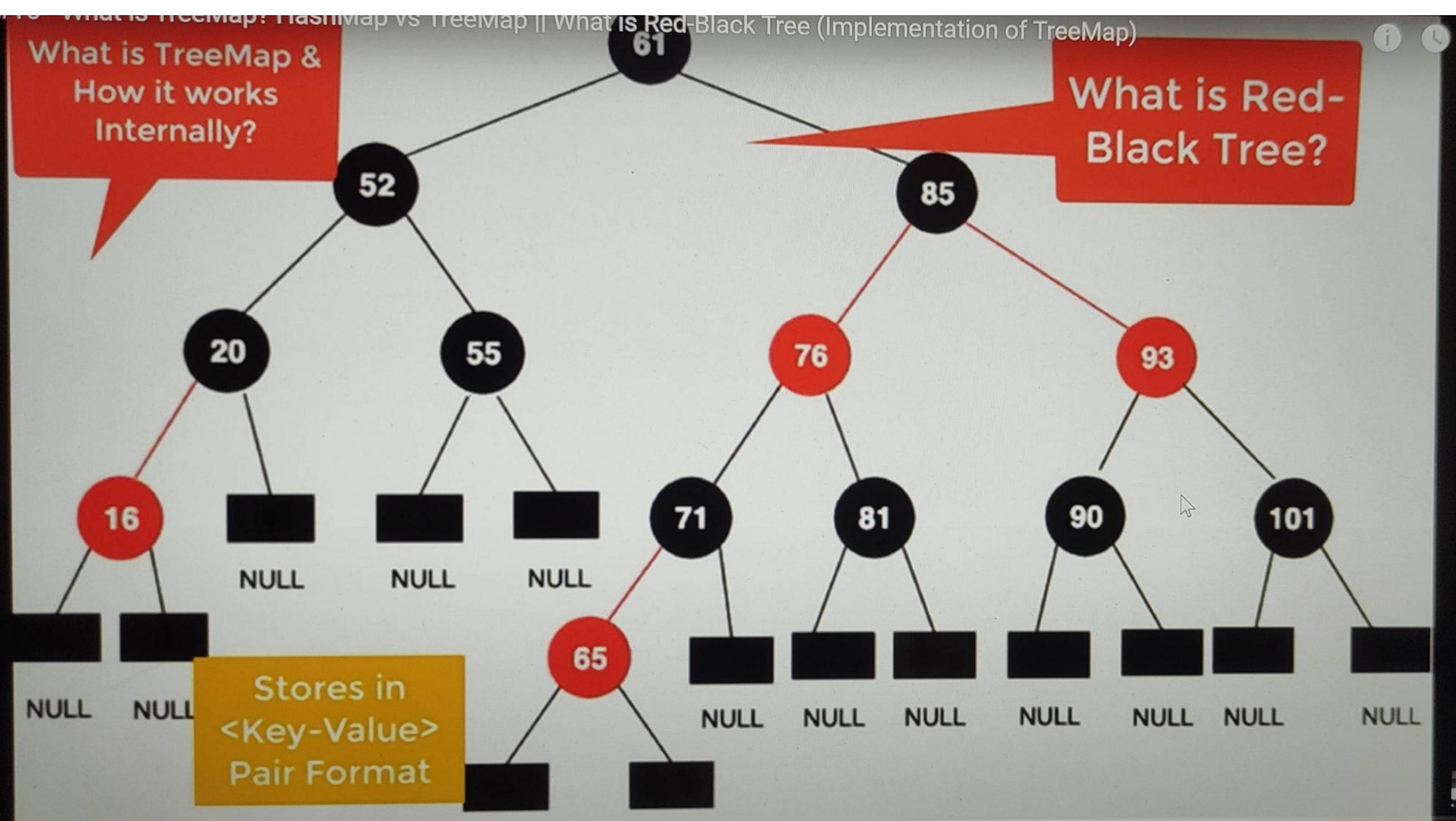
<"Tom", 100>

int h = hashCode("Tom") → 170268

int id = mod(h) → ④







The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with the following packages:
 - demo [boot]
 - DemoSpring
 - SecureApp [boot]
 - src/main/java
 - com.telusko.secureapp
 - AppSecurityConfig.java
 - HomeController.java
 - SecureAppApplication.java
 - src/main/resources
 - src/test/java
 - JRE System Library [JavaSE-1.8]
 - Maven Dependencies
 - src
 - target
 - mvnw
 - mvnw.cmd
 - pom.xml
- Open Editors:** SecureAppApplication.java, HomeController.java, home.jsp, SecureApp/pom.xml, AppSecurityConfig.java.
- Code View:** The code for AppSecurityConfig.java is displayed, which configures Spring Security to use an in-memory user details manager. The code includes imports for various Spring and security annotations and classes, and defines a UserDetailsService bean.

```
1 package com.telusko.secureapp;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import org.springframework.context.annotation.Bean;
7 import org.springframework.context.annotation.Configuration;
8 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
9 import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
10 import org.springframework.security.core.userdetails.User;
11 import org.springframework.security.core.userdetails.UserDetails;
12 import org.springframework.security.core.userdetails.UserDetailsService;
13 import org.springframework.security.provisioning.InMemoryUserDetailsManager;
14
15 @Configuration
16 @EnableWebSecurity
17 public class AppSecurityConfig extends WebSecurityConfigurerAdapter
18 {
19     @Bean
20     @Override
21     protected UserDetailsService userDetailsService() {
22
23
24         List<UserDetails> users = new ArrayList<>();
25         users.add(User.withDefaultPasswordEncoder().username("navin").password("1234").roles("USER").build());
26
27
28         return new InMemoryUserDetailsManager(users);
29     }
30 }
```

```
6
7     // its not synchronized
8     // its a highly performance java collection member
9     // faster than HashSet
10    // all the methods are implemented using bitwise arithmetic operations.
11
12- enum Lang {
13     JAVA, CSHARP, JAVASCRIPT, PYTHON, RUBY
14 }
15
16- public static void main(String[] args) {
17
18     //created a new enumset using enum
19     EnumSet<Lang> langs = EnumSet.allOf(Lang.class);
20     System.out.println(langs);
21
22     //empty enum set:
23     EnumSet<Lang> l = EnumSet.noneOf(Lang.class);
24     System.out.println(l);
25
26     //range(e1,e2):
27     EnumSet<Lang> enumRange = EnumSet.range(Lang.JAVA, Lang.JAVASCRIPT);
28     System.out.println(enumRange);
29
30     //of:
31     EnumSet.of(e)
32
33 }
34
35 }
```

```
27 // range(e1,e2):
28 EnumSet<Lang> enumRange = EnumSet.range(Lang.JAVA, Lang.JAVASCRIPT);
29 System.out.println(enumRange);
30
31 //of:
32 EnumSet<Lang> CSharpEnum = EnumSet.of(Lang.CSHARP);
33 System.out.println(CSharpEnum);
34
35 EnumSet<Lang> multipleEnum = EnumSet.of(Lang.JAVA, Lang.RUBY);
36 System.out.println(multipleEnum);
37
38 //add and addAll():
39 EnumSet<Lang> lang1 = EnumSet.allOf(Lang.class);
40 EnumSet<Lang> lang2 = EnumSet.noneOf(Lang.class);
41 lang2.add(Lang.JAVASCRIPT);
42 lang2.addAll(lang1);
43 System.out.println(lang2);
44
45 //how to iterate EnumSet: iterator:
46 EnumSet<Lang> fullLang = EnumSet.allOf(Lang.class);
47
48 Iterator<Lang> it = fullLang.iterator();
49
50 while(it.hasNext()) {
51     System.out.print(it.next());
52     System.out.print(" , ");
53 }
54
55 //remov
56
57
```

```
47  
48     Iterator<Lang> it = fullLang.iterator();  
49  
50     while(it.hasNext()) {  
51         System.out.print(it.next());  
52         System.out.println(" , ");  
53     }  
54  
55     //remove() and removeAll():  
56     EnumSet<Lang> newLang = EnumSet.allOf(Lang.class),  
57     System.out.println(newLang);  
58  
59     boolean b = newLang.remove(Lang.CSHARP);  
60     System.out.println(b);  
61     System.out.println(newLang);  
62  
63     boolean b1 = newLang.removeAll(newLang);  
64     System.out.println(b1);  
65     System.out.println(newLang);  
66  
67  
68  
69  
70  
71     }  
72  
73  
74 }
```



Please subscribe
like & share