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

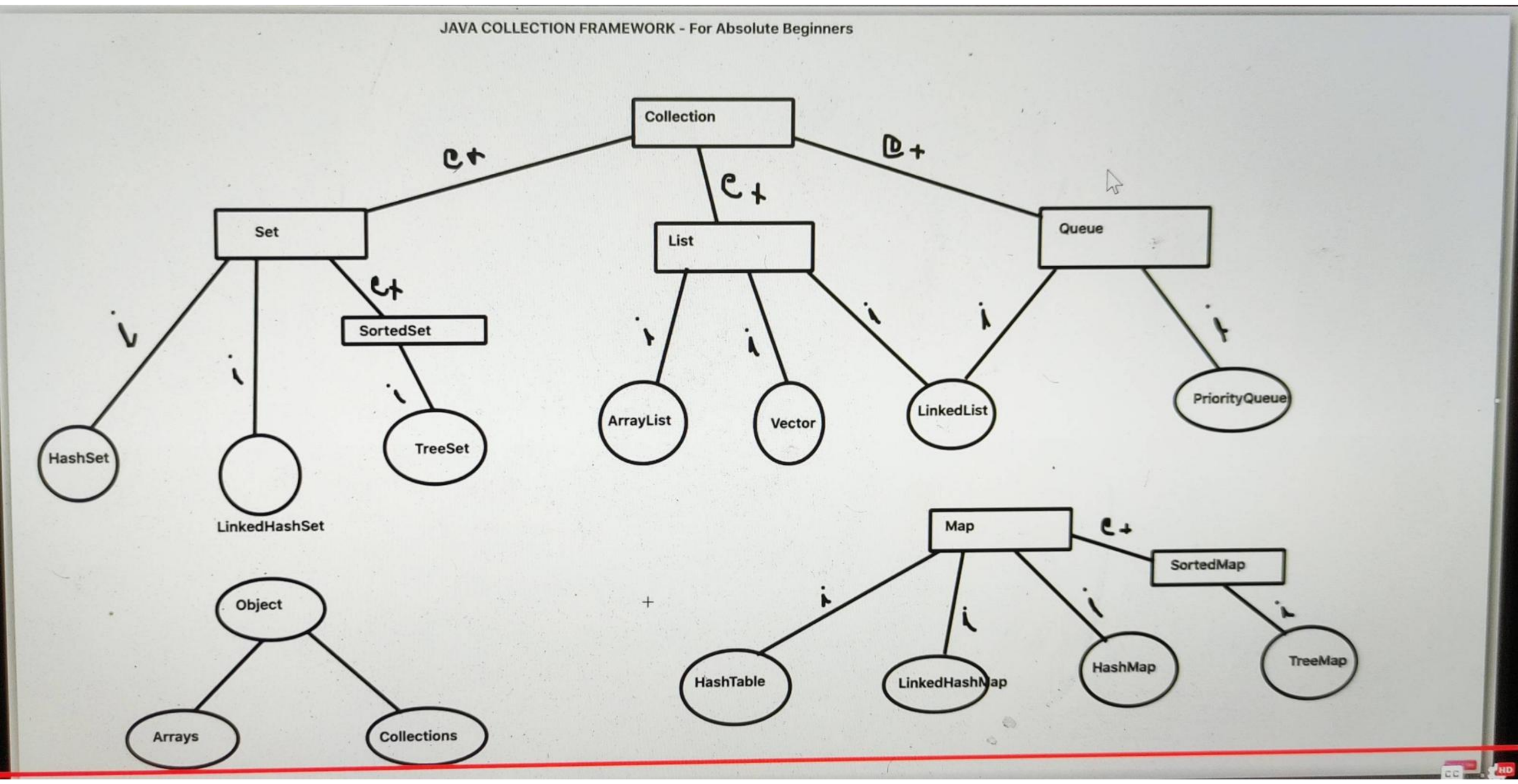
Iterator:
to iterate the elements from a collection

Advantages of Collections?

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

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
IllegalArgumentExcetion
IllegalStateException
UnSupportedOperationException



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	❌	❌	❌	❌	✅	✅

```
public static void main(String[] args) {


    ArrayList<String> studentList = new ArrayList<String>();

    studentList.add("Tom");//0
    studentList.add("Naveen");//1
    studentList.add("Steve");//2
    studentList.add("Lisa");//3

    //typical for loop:
    for(int i=0; i<studentList.size(); i++) {
        System.out.println(studentList.get(i));
    }

    System.out.println("----");

    //for each loop:
    for(String s : studentList) {
        System.out.println(s);
    }

    System.out.println("----");

    //JDK 8 - streams with lambda:
    studentList.stream().forEach(ele -> System.out.println(ele));

```

```
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
```

*ArrayListMethods.java

```
27 //
28 //        ar1.clear();
29 //        System.out.println(ar1);
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", "
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);
55
56
```

How to synchronize ArrayList in Java:

1. Collections.synchronizedList() - method - returns synchronized list

2. copyOnWriteArrayList - class - Thred Safe variant of ArrayList

SynchronizedArayList.java

```java
public class SynchronizedArayList {

    public static void main(String[] args) {

        List<String> namesList = Collections.synchronizedList(new ArrayList<String>());

        namesList.add("Java");
        namesList.add("Python");
        namesList.add("Ruby");

        //add, remove - we dont need explicit synchronization

        //to fetch/traverse the values from this list -- we have to use explicit synchronization

        synchronized (namesList) {

            Iterator<String> it = namesList.iterator();

            while(it.hasNext()) {

                System.out.println(it.next());
            }


        }



    }

}
```

```
CopyOnWriteArrayList<String> empList = new CopyOnWriteArrayList<String>();
empList.add("Tom");
empList.add("Steve");
empList.add("Naveen");

//we dont need explicit synchronization for any operation: add/remove/traverse

Iterator<String> it = empList.iterator();
while(it.hasNext()) {

    System.out.println(it.next());
}

}
```

Writable Smart Insert 38 : 30 : 98

ArrayListCompare.java | RemoveDuplicateElements.java

```java
package ArrayListConcept;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedHashSet;

public class RemoveDuplicateElements {

    public static void main(String[] args) {

        ArrayList<Integer> numbers = new ArrayList<Integer>(Arrays.asList(1,2,3,2,2,3,1,4,5,6,1,7,8,9,7));

        //1. LinkedHashSet

        LinkedHashSet<Integer> linkedHashSet = new LinkedHashSet<Integer>(numbers);

        ArrayList<Integer> numbersListWithoutDuplicates = new ArrayList<Integer>(linkedHashSet);

        System.out.println(numbersListWithoutDuplicates);


    }

}
```

```java
        ArrayList<Integer> numbers = new ArrayList<Integer>(Arrays.asList(1,2,3,2,2,3,1,4,5,6,1,7,8,9,7));

        //1. LinkedHashSet

        LinkedHashSet<Integer> linkedHashSet = new LinkedHashSet<Integer>(numbers);

        ArrayList<Integer> numbersListWithoutDuplicates = new ArrayList<Integer>(linkedHashSet);

        System.out.println(numbersListWithoutDuplicates);

        //2. JDK 8 - stream:

        ArrayList<Integer> marksList = new ArrayList<Integer>(Arrays.asList(1,2,3,2,2,3,1,4,5,6,1,7,8,9,7));
        List<Integer> marksListUnique = marksList.stream().distinct().collect(Collectors.toList());
        System.out.println(marksListUnique);

    }

}
```

ArrayListCompare.java

```java
public class ArrayListCompare {

    public static void main(String[] args) {

        //1. sort and then equals:

        ArrayList<String> l1 = new ArrayList<String>(Arrays.asList("A", "B", "C", "D", "F"));

        ArrayList<String> l2 = new ArrayList<String>(Arrays.asList("A", "B", "C", "D", "E"));

        ArrayList<String> l3 = new ArrayList<String>(Arrays.asList("B", "A", "C", "D", "F"));

        Collections.sort(l1);
        Collections.sort(l2);

        System.out.println(l1.equals(l2));//false

        Collections.sort(l3);
        System.out.println(l1.equals(l3));//true


        //2. compare two list - find out the additional elements:
        ArrayList<String> l4 = new ArrayList<String>(Arrays.asList("A", "B", "C", "D", "F"));

        ArrayList<String> l5 = new ArrayList<String>(Arrays.asList("A", "B", "C", "D", "E"));

        l4.removeAll(l5);
        System.out.println(l4);



    }
```

Writable Smart Insert 30 : 9 : 811

ArrayListCompare.java

```
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);//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 }
```

```java

        //no order - no indexing
        //stores values -- Key-Value <k,v>
        //key can not be duplicate
        //can store n number of null values but only one null key
        //hashmap is not thread-safe -  unsynchronized

        HashMap<String, String> capitalMap = new HashMap<String, String>();
        capitalMap.put("India", "New Delhi");
        capitalMap.put("USA", "Washington DC");
        capitalMap.put("UK", "London");
        capitalMap.put("UK", "London11");
        capitalMap.put(null, "Berlin");
        capitalMap.put(null, "LA");
        capitalMap.put("Russia", null);
        capitalMap.put("France", null);


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

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

```

```
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);
}
```

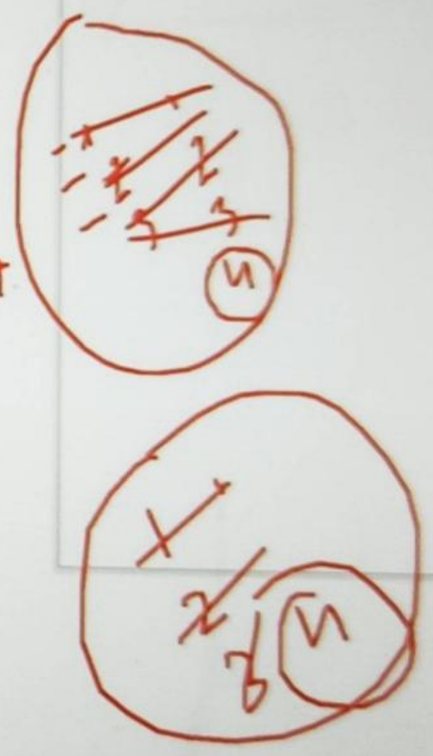
```
        System.out.println(capitalMap.get(null));
        System.out.println(capitalMap.get("France"));

        //iterator: over the keys: by using keySet()
        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);
        }

        System.out.println("----------");
        //iterator: over the set (pair): by using entrySet

        Iterator<Entry<String, String>> it1 = capitalMap.entrySet().iterator();

        while(it1.hasNext()) {
            Entry<String, String> entry = it1.next();
            System.out.println("key = " + entry.getKey() + " and value = " + entry.getValue());
        }


    }

```

```
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  }
```

```java

        HashMap<Integer, String> map1 = new HashMap<Integer, String>();

        map1.put(1, "A");
        map1.put(2, "B");
        map1.put(3, "C");

        HashMap<Integer, String> map2 = new HashMap<Integer, String>();

        map2.put(3, "C");
        map2.put(1, "A");
        map2.put(2, "B");

        HashMap<Integer, String> map3 = new HashMap<Integer, String>();

        map3.put(1, "A");
        map3.put(2, "B");
        map3.put(3, "C");
        map3.put(3, "D");


        //1. on the basis of key-value: use equals method:
        System.out.println(map1.equals(map2));//true
        System.out.println(map1.equals(map3));//false


        //2. compare hashmaps for the same keys: keySet():
        System.out.println(map1.keySet().equals(map2.keySet()));//true
        System.out.println(map1.keySet().equals(map3.keySet()));

```

```java
        map3.put(1, "A");
        map3.put(2, "B");
        map3.put(3, "C");
        map3.put(3, "D");


        //1. on the basis of key-value: use equals method:
        System.out.println(map1.equals(map2));//true
        System.out.println(map1.equals(map3));//false


        //2. compare hashmaps for the same keys: keySet():
        System.out.println(map1.keySet().equals(map2.keySet()));//true
        System.out.println(map1.keySet().equals(map3.keySet()));//true

        //3. find out the extra keys:
        HashMap<Integer, String> map4 = new HashMap<Integer, String>();

        map4.put(1, "A");
        map4.put(2, "B");
        map4.put(3, "C");
        map4.put(4, "D");

        //combine the keys from both the maps: using HashSet:
        HashSet<Integer> combineKeys = new HashSet<>(map1.keySet());
        //add the keyset from map4:
        combineKeys.addAll(map4.keySet());
        combineKeys.removeAll(map1.keySet());
        System.out.println(combineKeys);
```

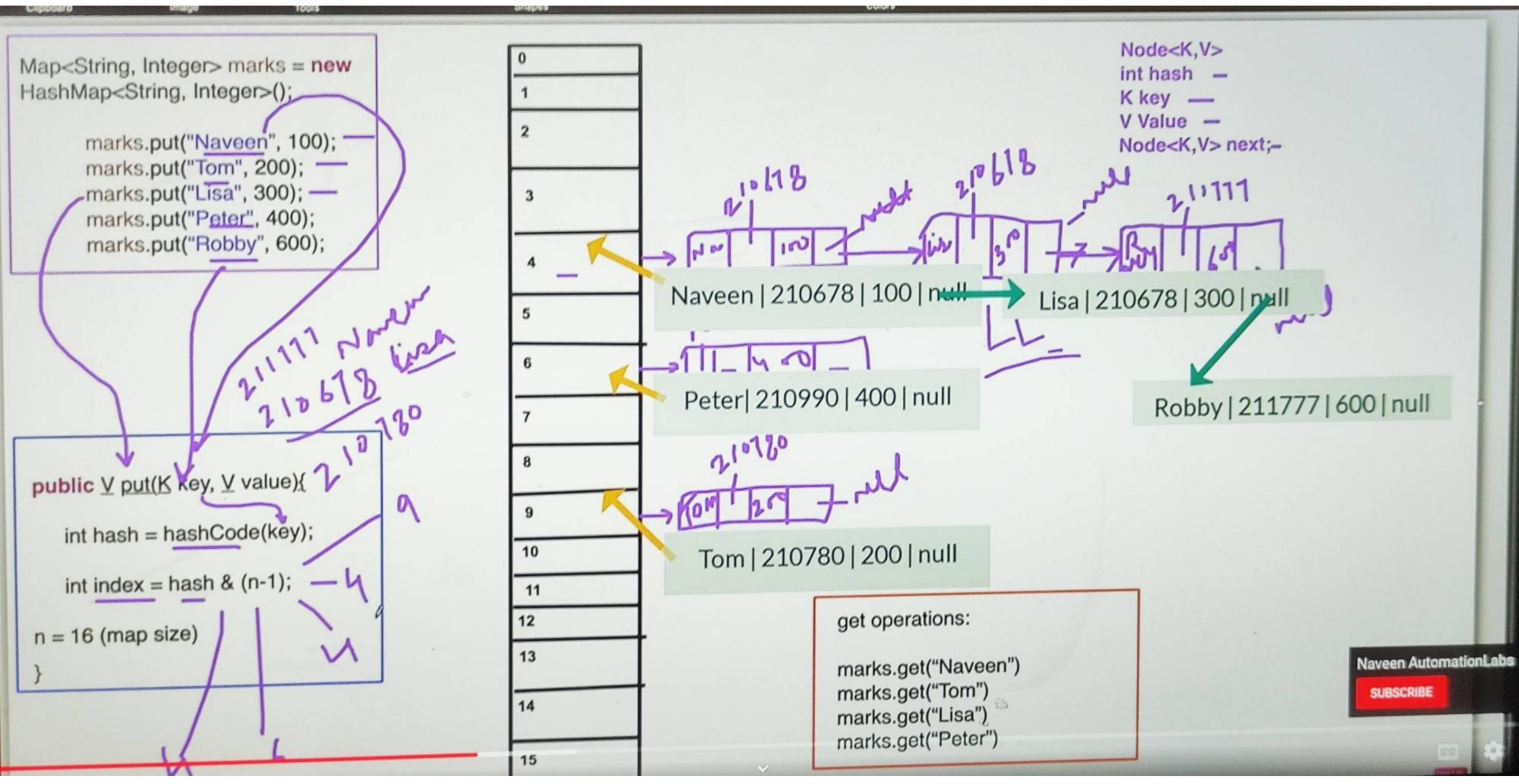
```
<terminated> HashMapCompare [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_261.jdk/Con
true
false
true
true
[4]
```

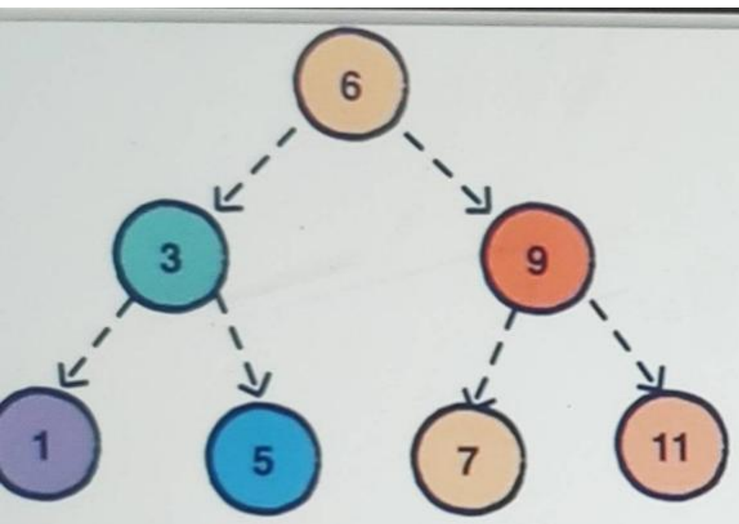


```
52          combineKeys.removeAll(map4.keySet());
53          combineKeys.removeAll(map1.keySet());
54          System.out.println(combineKeys);
55
56          //4. compare maps by values:
57          HashMap<Integer, String> map5 = new HashMap<Integer, String>();
58
59          map5.put(1, "A");
60          map5.put(2, "B");
61          map5.put(3, "C");
62
63          HashMap<Integer, String> map6 = new HashMap<Integer, String>();
64
65          map6.put(4, "A");
66          map6.put(5, "B");
67          map6.put(6, "C");
68
69          HashMap<Integer, String> map7 = new HashMap<Integer, String>();
70
71          map7.put(1, "A");
72          map7.put(2, "B");
73          map7.put(3, "C");
74          map7.put(4, "C");
75
76          //1. duplicates are not allowed:
77          System.out.println(new ArrayList<>(map5.values()).equals(new ArrayList<>(map6.values())));//true
78          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:Iusing 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

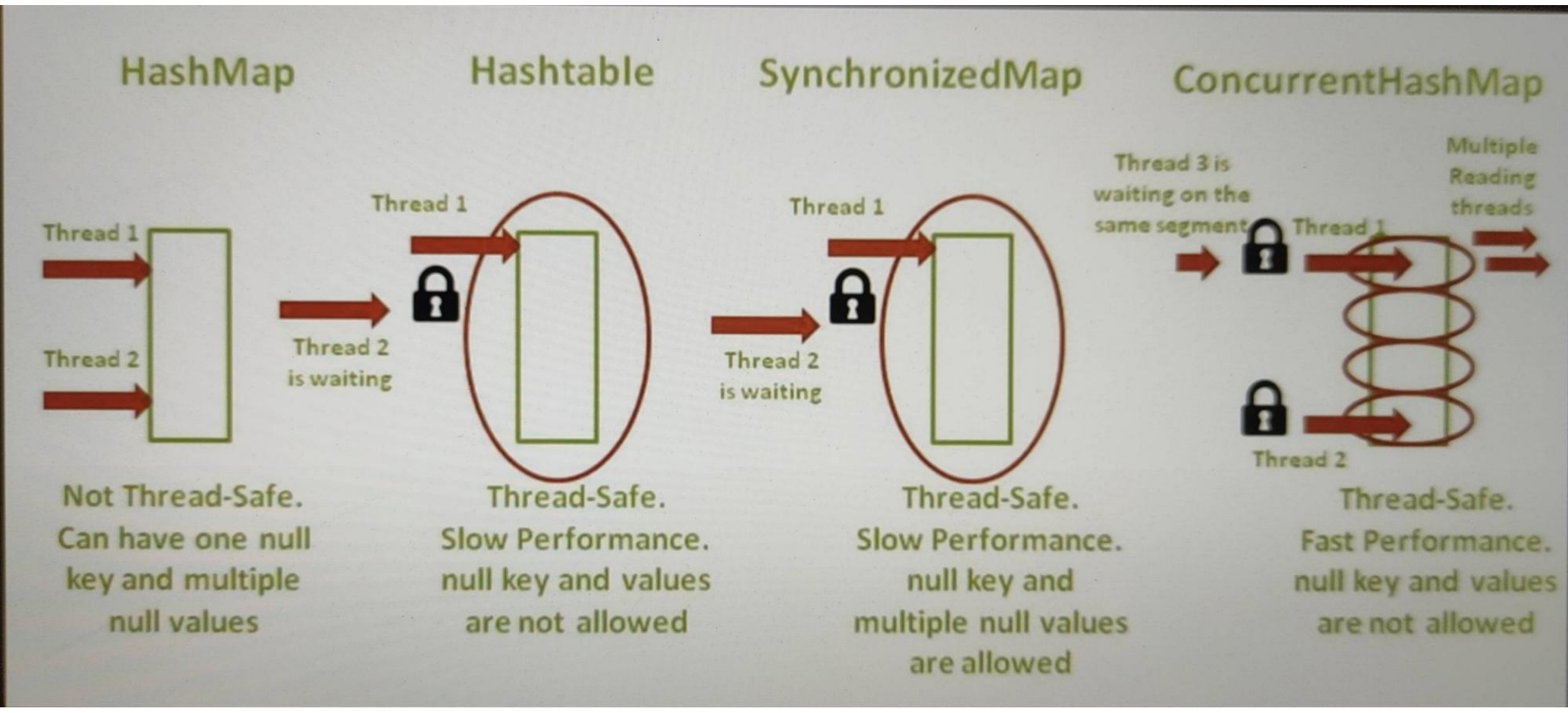
```
24          //1. using HashMap class
25
26          HashMap<String, String> map1 = new HashMap<>();
27          Map<String, String> map2 = new HashMap<>();
28
29          //2. static way : static hashmap:
30          System.out.println(HashMapInitialization.marksMap.get("A"));
31
32          //3. 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
55                  ).collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue));
56
57          System.out.println(map5.get("Naveen"));
```

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

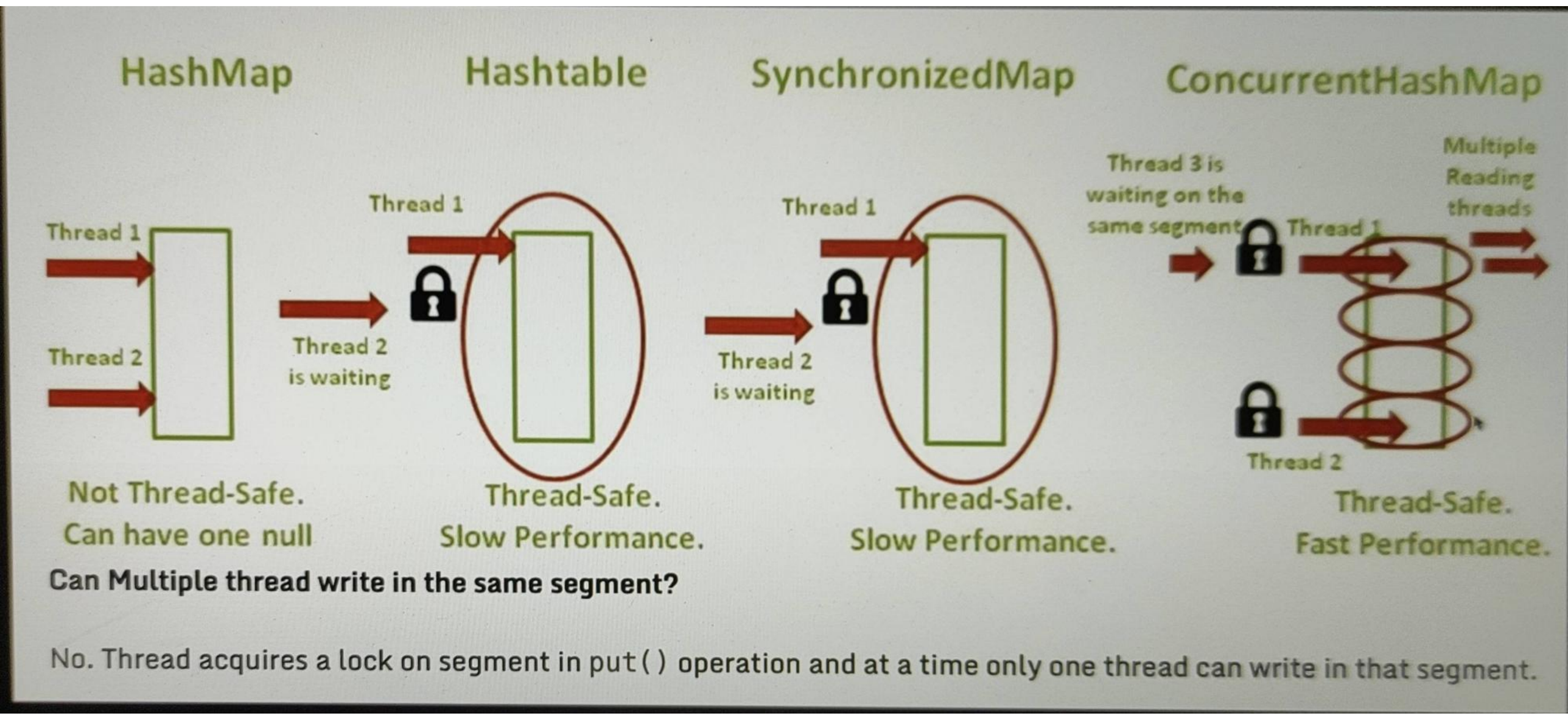
#11 - Different ways of creating HashMap in Java

```
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          //singletom 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
92          //ofEntries method: no limitations on pairs (key-value)
93          //Immutable Maps:
94          Map<String, Integer> map7 = Map.ofEntries(
95                  new AbstractMap.SimpleEntry<>("A", 100),
96                  new AbstractMap.SimpleEntry<>("B", 200),
97                  new AbstractMap.SimpleEntry<>("C", 300)
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
```

24:03 / 24:59

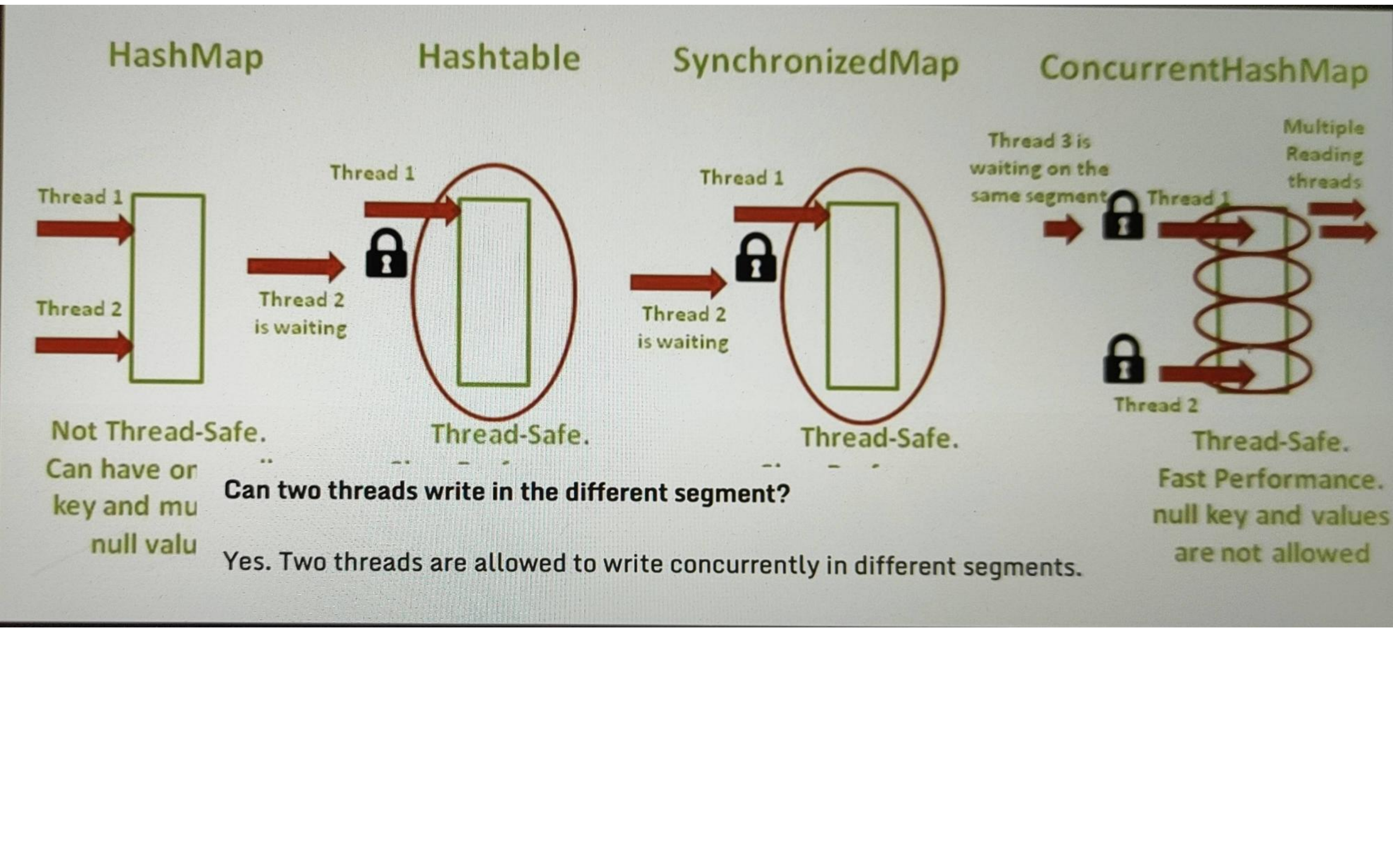


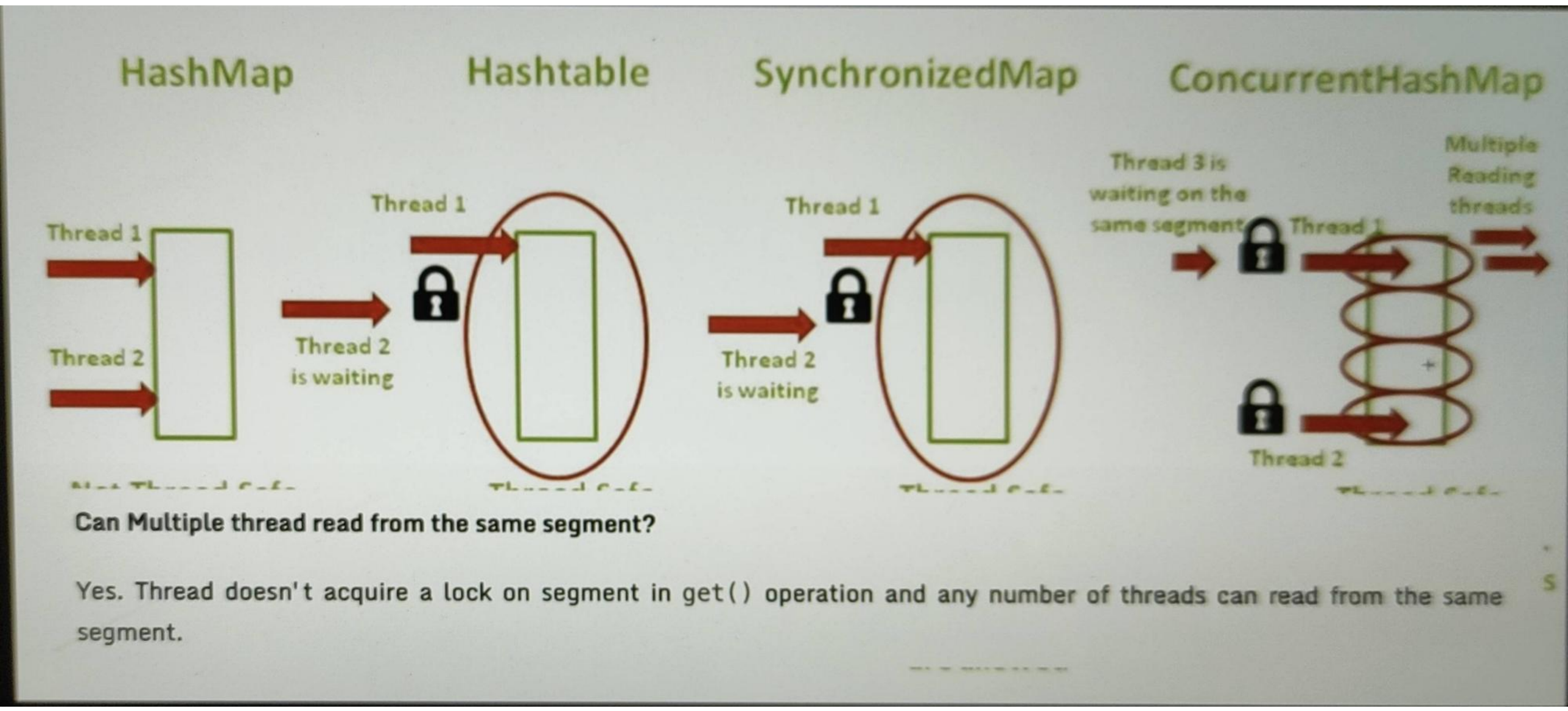
```java
public class HashMapSync {

    public static void main(String[] args) {

        //synchronizedMap method in collections class:

        Map<String, String> map1 = new HashMap<String, String>();

        map1.put("1", "Naveen");
        map1.put("2", "Tom");
        map1.put("3", "Lisa");


        //create synchronizedMap:
        Map<String, String> syncMap = Collections.synchronizedMap(map1);

        System.out.println(syncMap);

        //concurrenthashMap: it does not throw any CocurrentModificationException

        ConcurrentHashMap<String, String> concurrentMap = new ConcurrentHashMap<>();

        concurrentMap.put("A", "Java");
        concurrentMap.put("B", "Python");
        concurrentMap.put("C", "Ruby");

        System.out.println(concurrentMap.get("A"));


    }

}
```



Can Multiple thread write in the same segment?

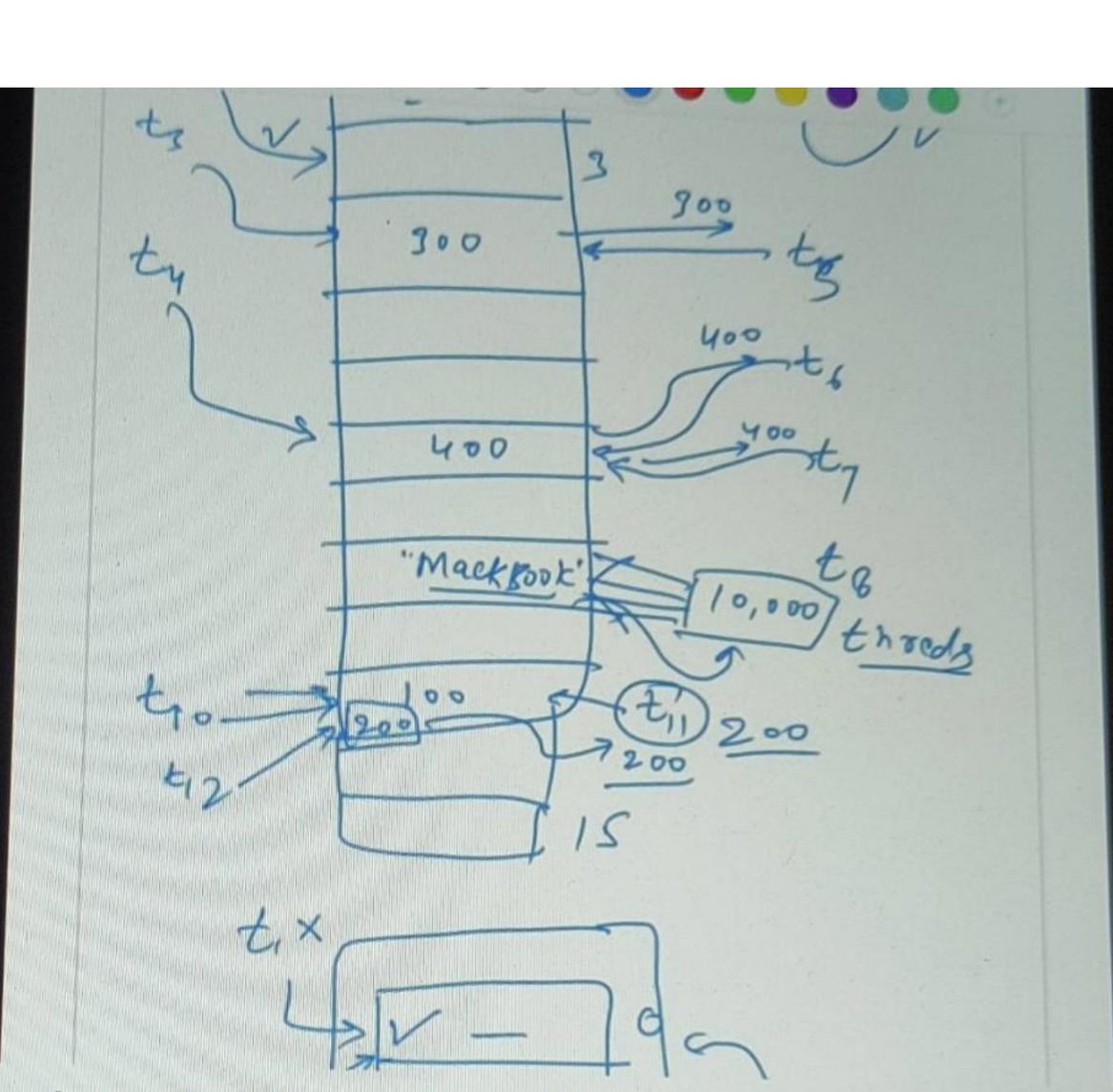
No. Thread acquires a lock on segment in `put()` operation and at a time only one thread can write in that segment.





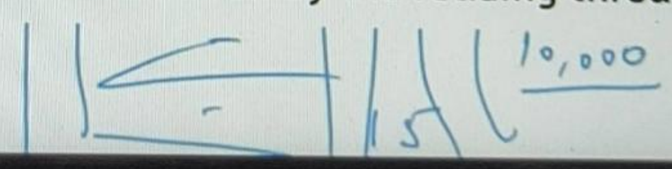
Can Multiple thread read from the same segment?

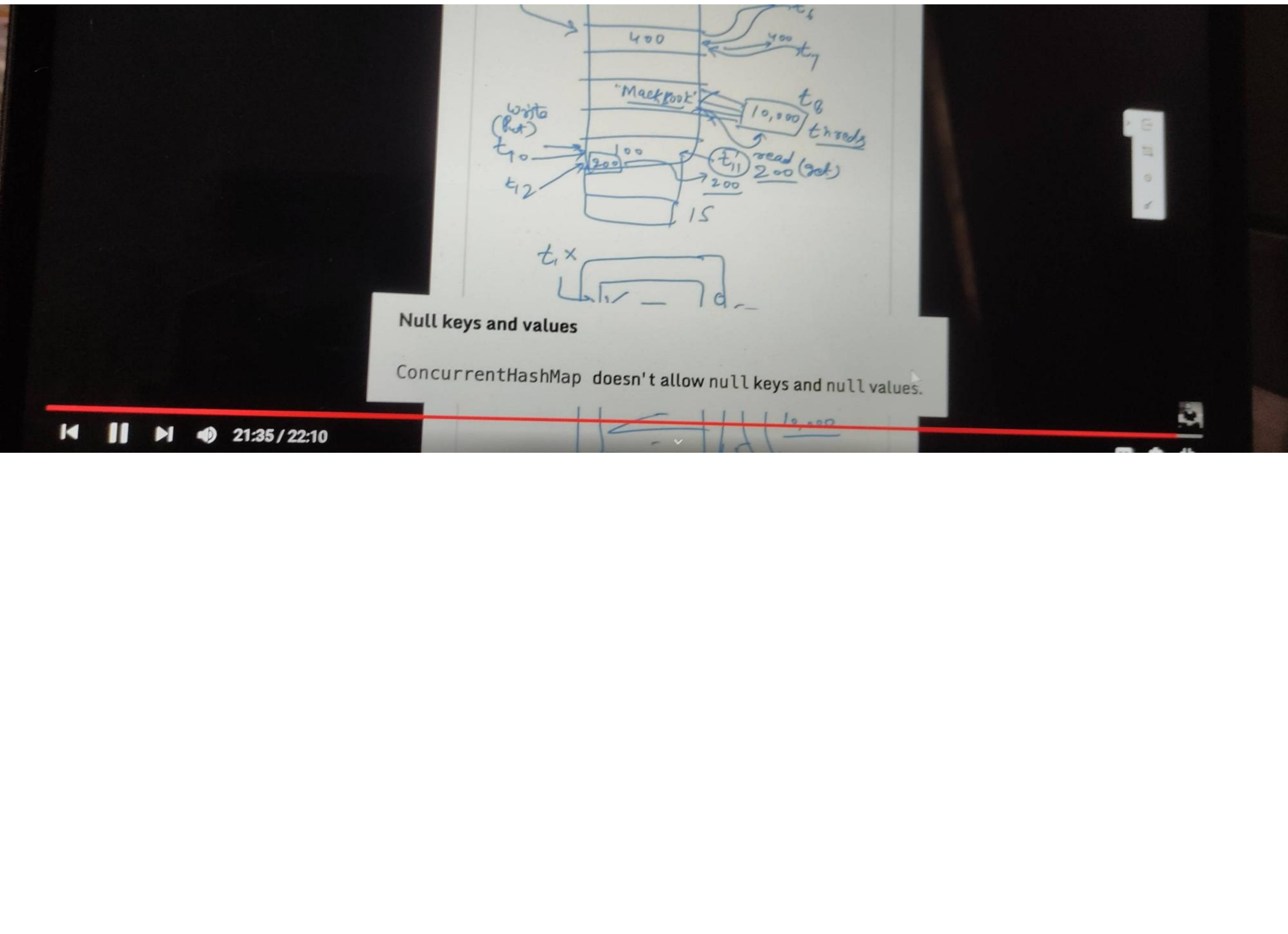
Yes. Thread doesn't acquire a lock on segment in `get()` operation and any number of threads can read from the same segment.

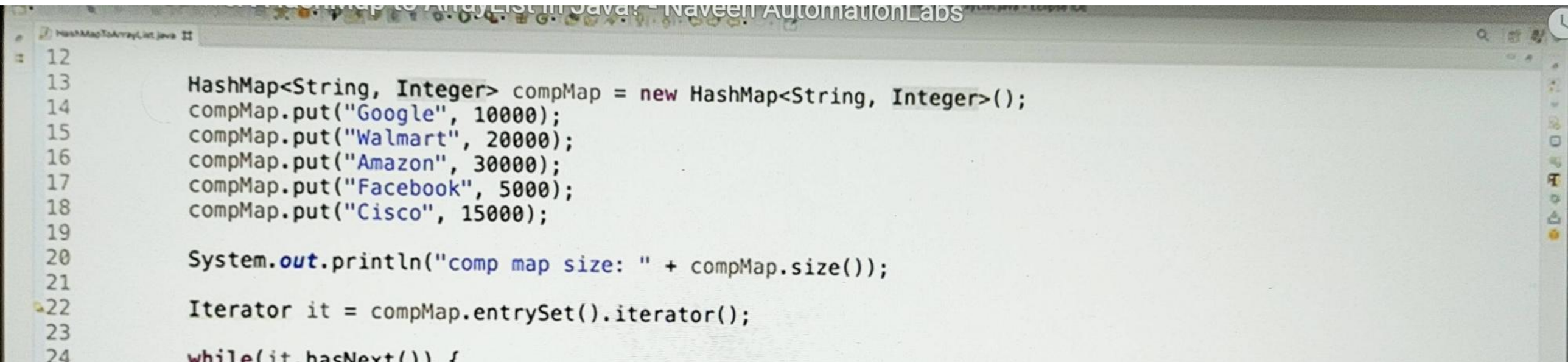


If one thread is writing in a segment, can another thread read from that segment()?

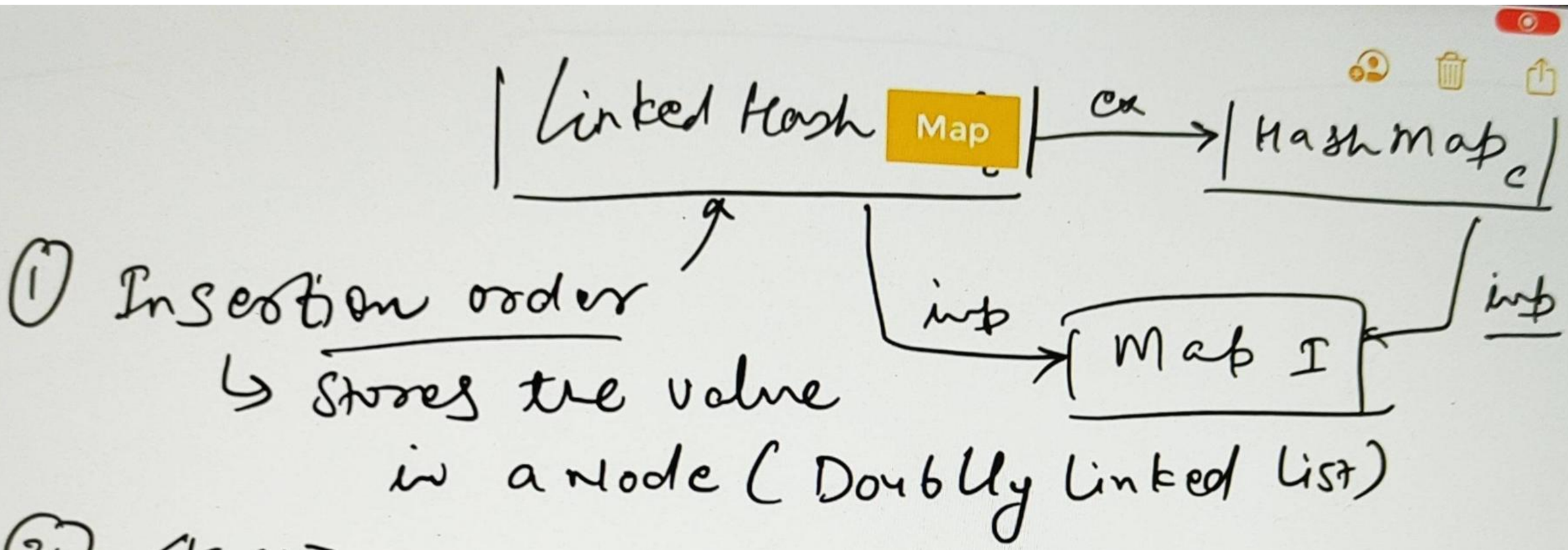
Yes. but in this case last updated value will be seen by the reading thread.







```java
        HashMap<String, Integer> compMap = new HashMap<String, Integer>();
        compMap.put("Google", 10000);
        compMap.put("Walmart", 20000);
        compMap.put("Amazon", 30000);
        compMap.put("Facebook", 5000);
        compMap.put("Cisco", 15000);

        System.out.println("comp map size: " + compMap.size());

        Iterator it = compMap.entrySet().iterator();

        while(it.hasNext()) {
            Map.Entry pairs = (Map.Entry)it.next();
            System.out.println(pairs.getKey() + " = " + pairs.getValue());
        }

        System.out.println("--------");

        compMap.forEach((k,v) -> System.out.println("key =" + k + "value = " + v));

        //convert hashmap keys into ArrayList:
        List<String> compNamesList = new ArrayList<String>(compMap.keySet());
        System.out.println("total comp count = " + compNamesList.size());
        for(String t : compNamesList) {
            System.out.println(t);
        }

        System.out.println("--------");

        //convert hashmap values into ArrayList:
        List<Integer> EmpValuesList = new ArrayList<Integer>(compMap.values());
        System.out.println("Total emp values count = "+ EmpValuesList.size());
```

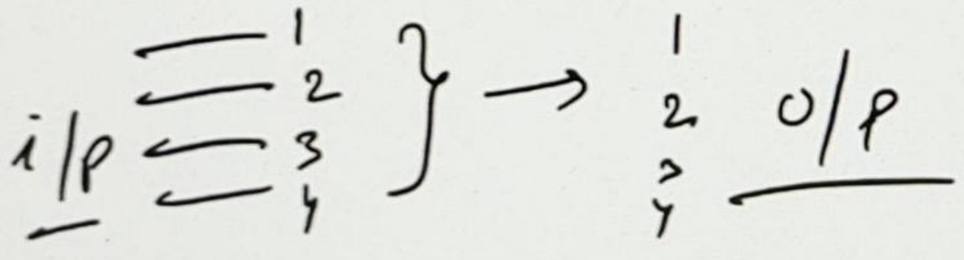


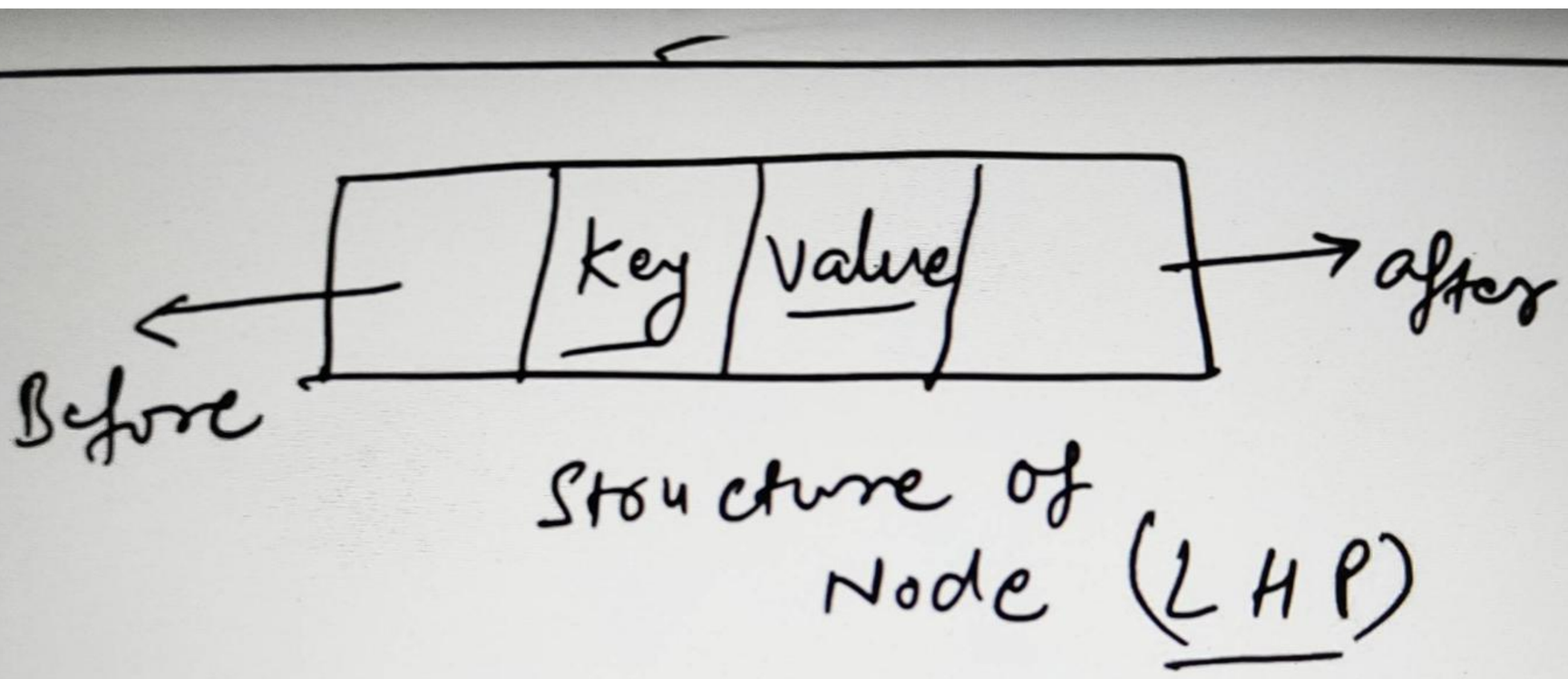
① Insertion order
↳ Stores the value in a Node (Doubly Linked List)

② . <K-V>

③ . not Sync (not Thread-Safe)

④ . one null key & Mutliple Null Values.

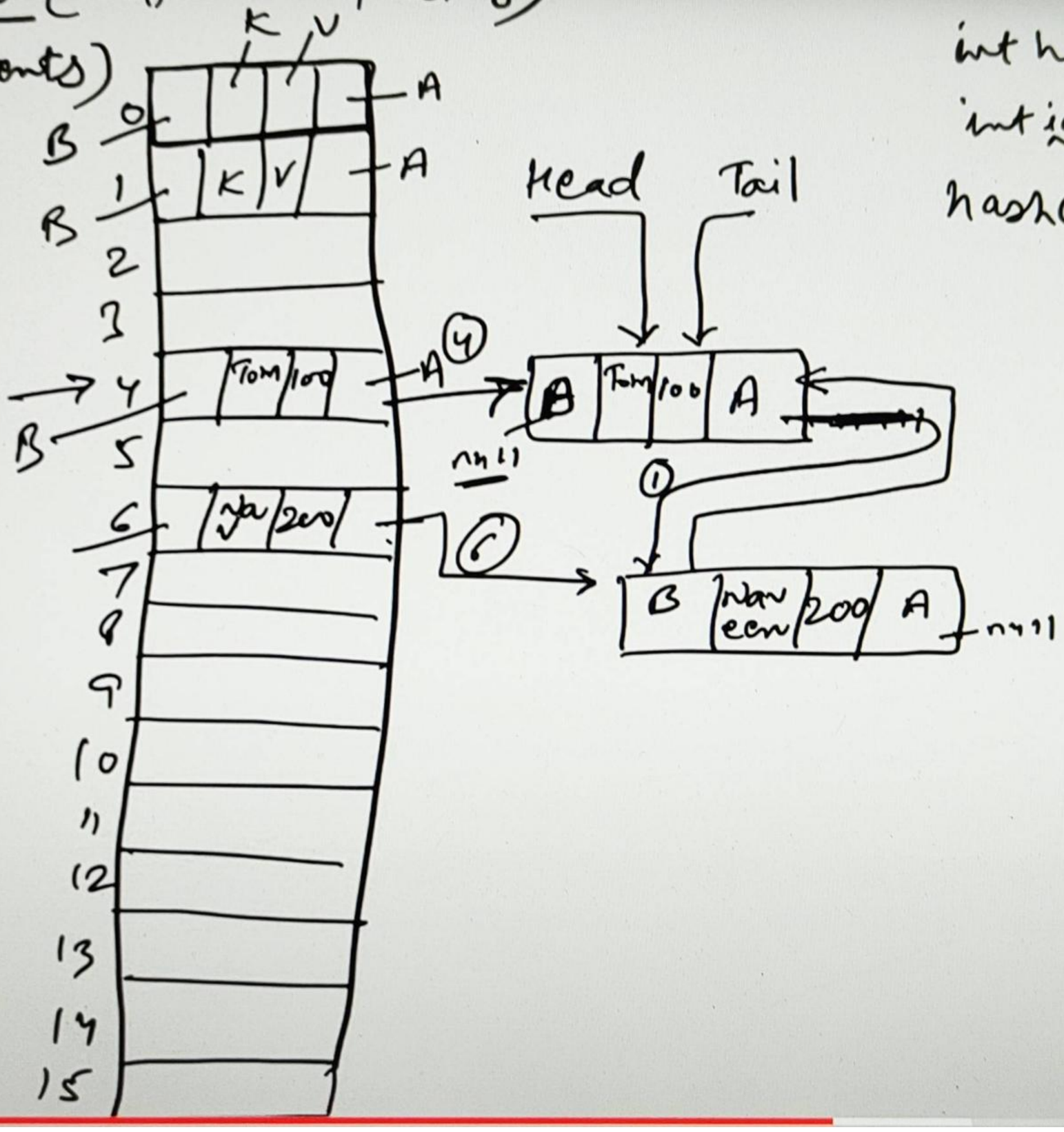




Structure of Node (LHP)

N=16 (Default capacity)

(Segments)



int h = hashCode(key);

int index = mod(h);

hashCode("Tom");

h = 170268

index = 4

hashCode("Naveen")

h = 170280

index = 6

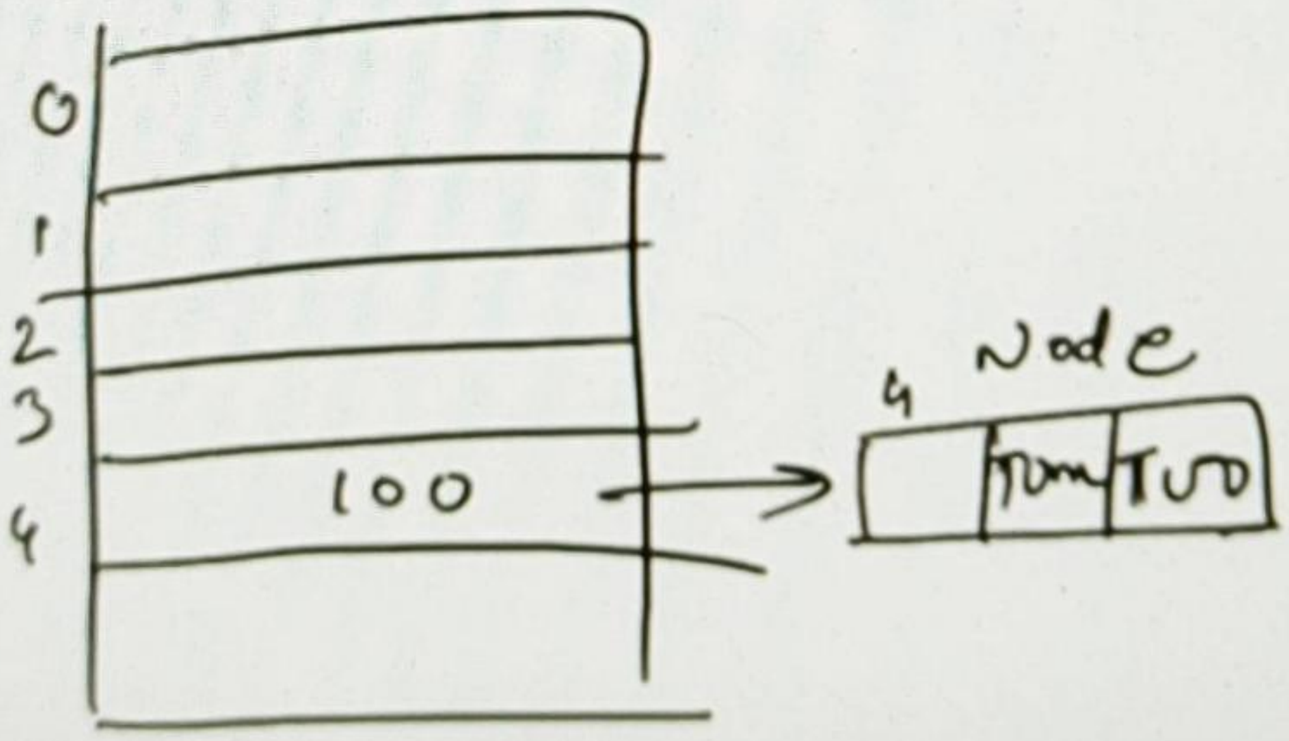
Hash map top Interview Questions

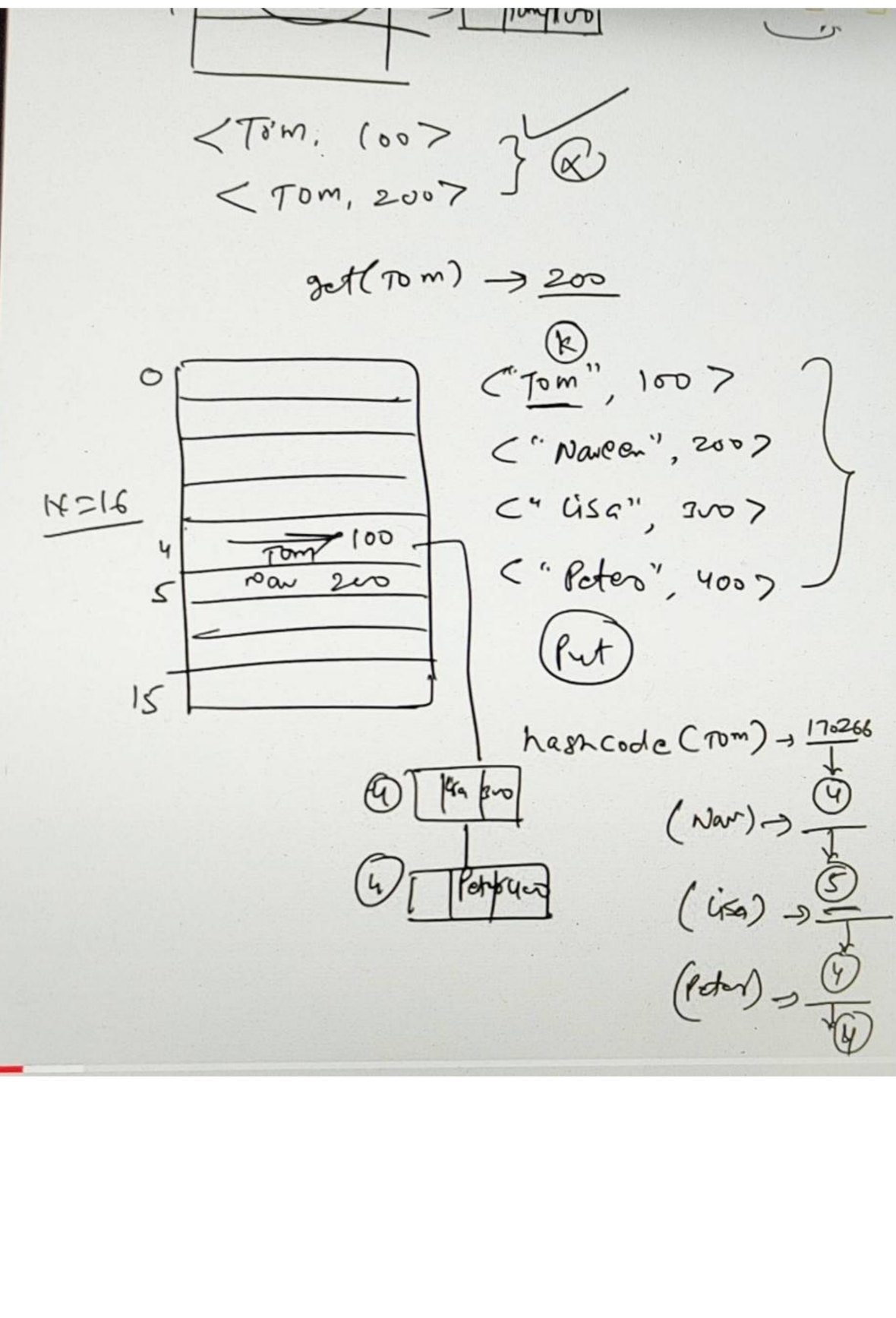
<Key, Value>

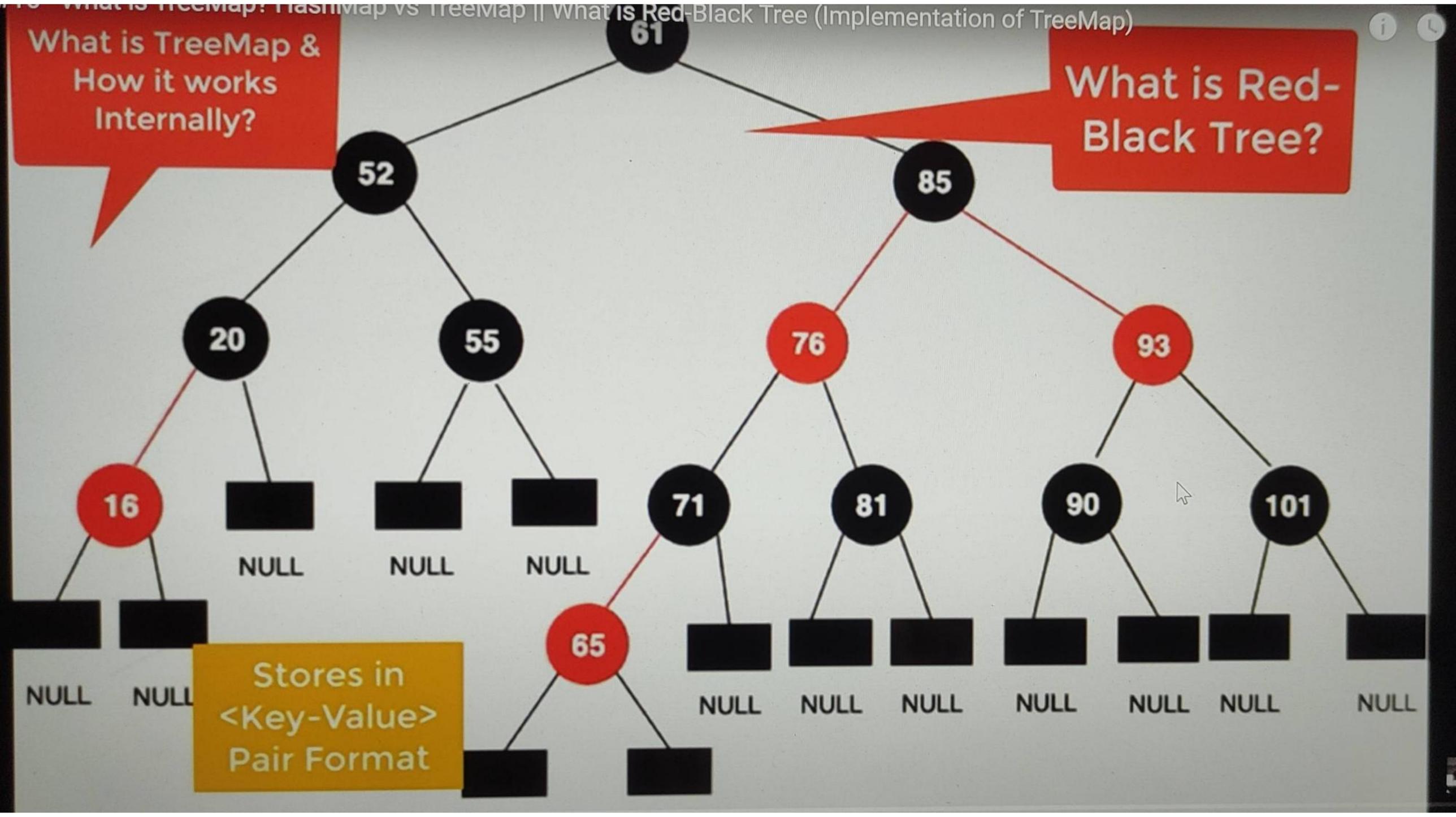
<"Tom", 100>

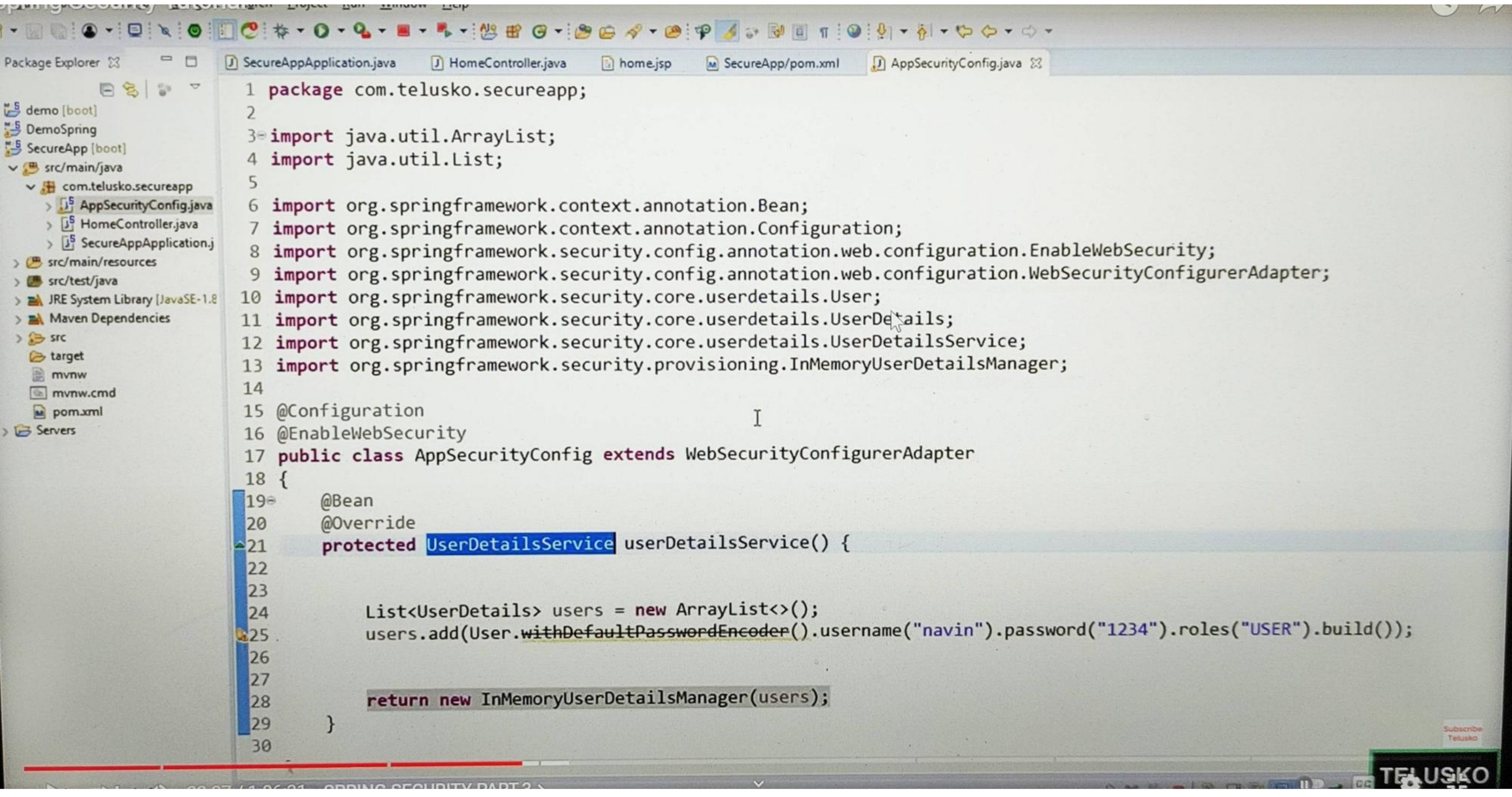
int h = hashCode("Tom") → 170268

int id = mod(h) → 4

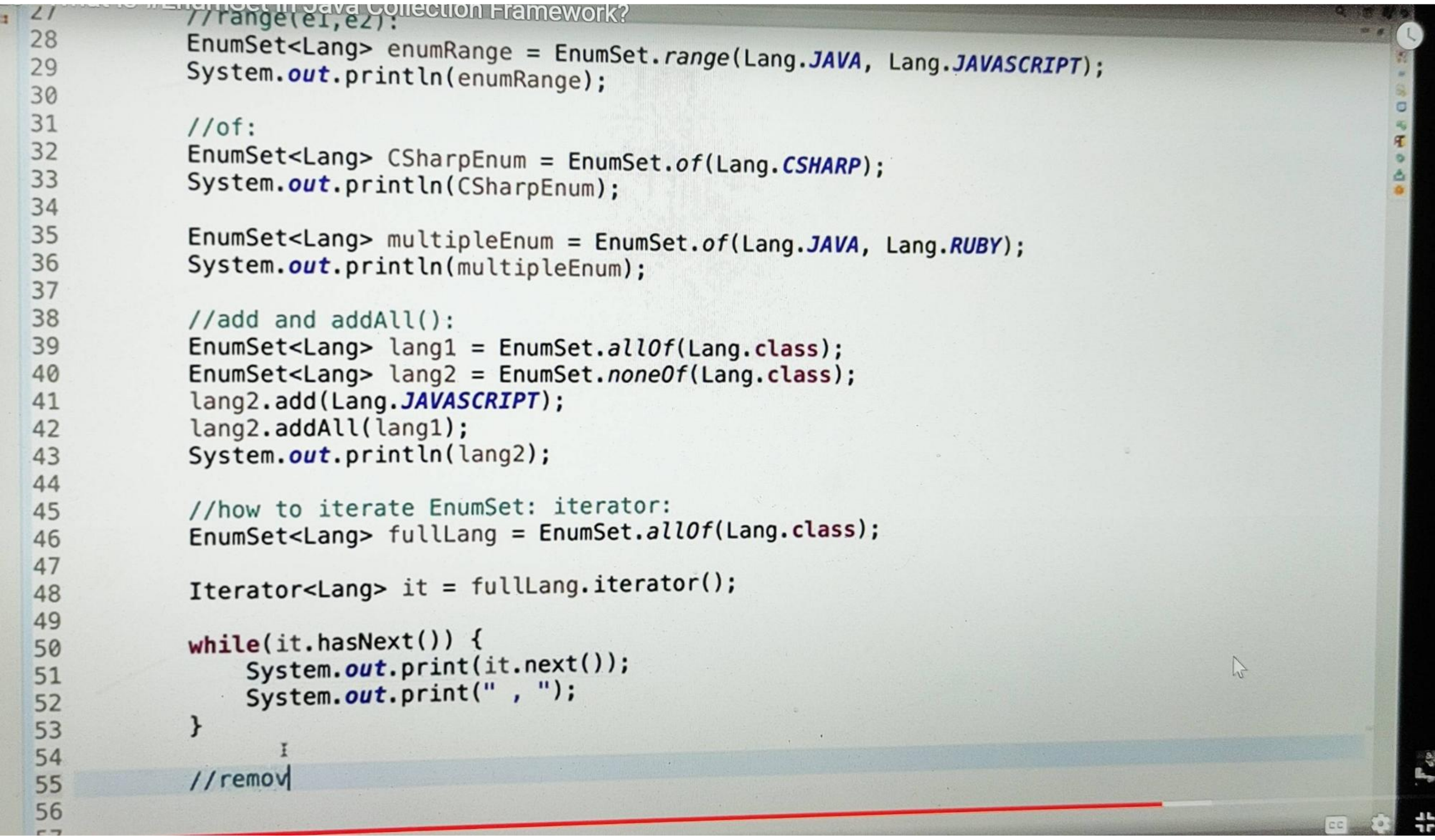








```java
// its not synchronized
// its a highly performance java collection member
// faster than HashSet
// all the methods are implemented using bitwise arithmetic operations.

enum Lang {
    JAVA, CSHARP, JAVASCRIPT, PYTHON, RUBY
}

public static void main(String[] args) {

    //created a new enumset using enum
    EnumSet<Lang> langs = EnumSet.allOf(Lang.class);
    System.out.println(langs);

    //empty enum set:
    EnumSet<Lang> l = EnumSet.noneOf(Lang.class);
    System.out.println(l);

    //range(e1,e2):
    EnumSet<Lang> enumRange = EnumSet.range(Lang.JAVA, Lang.JAVASCRIPT);
    System.out.println(enumRange);

    //of:
    EnumSet.of(e)

}

}
```



```
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  }
```