hashmap internal working

==========================

Collections:

- Hashsetinternally uses hashmap
- LinkedHashSet internally uses LinkedHashMap
- TreeSet internally uses TreeMap

so unitil map is not clear we cannot know how set is working

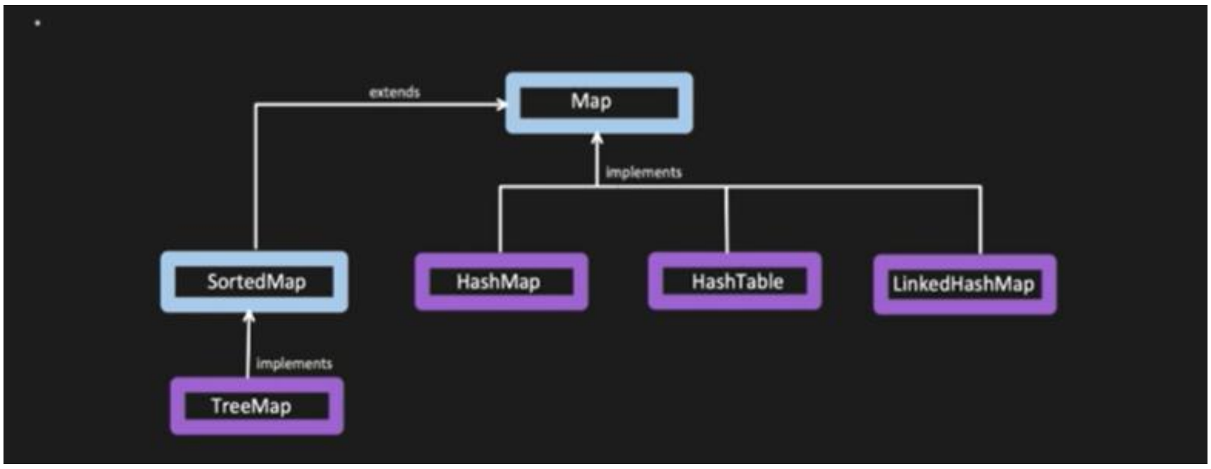
- im map, we have key and values
- all values will be mapped to a pirticualr key

Q) why Map is not part of collection?

Because all Collection work on values and not a key value pair

So it requires different methods

All available meths in collection is of no use use for map , so there is no point of bringing map under the collection



Map Properties:

- Map is an interface and HAshmap, Hashtable, LinkedHashMap implemnets Map interface
- We can store data in key value form
- Eg: roll no1, name Java, roll no 2, name scratch
- Duplicate key cannot be existed in map

Some methods:

Size method:

- It returns how many key value pair are there
- Lets say 1, java, 2, scratch then size will be 2

isEmpty():

- If map is empty, it will return true

Get(key):

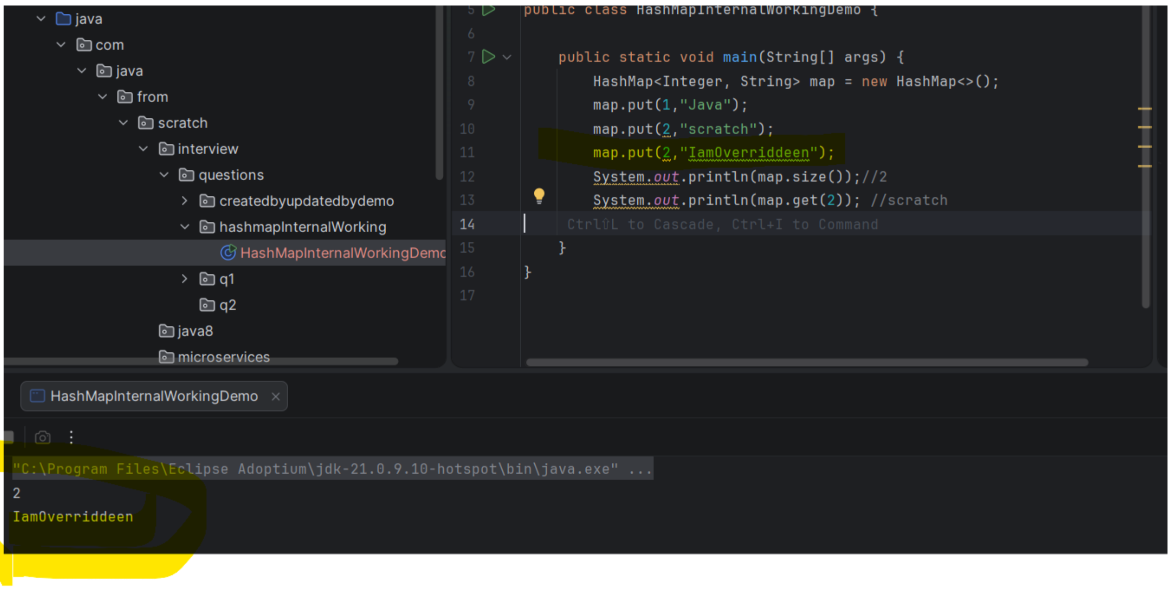
- It will return the value which is present for key 2

Put():

- It is used to insert the value in the map
- Example:

```
map.put(1,"Java");
map.put(2,"scratch");
```

- if you try to put the same key again, then the value will be overridden see in below example



```
public class HashMapInternalWorkingDemo {

    public static void main(String[] args) {
        HashMap<Integer, String> map = new HashMap<>();
        map.put(1,"Java");
        map.put(2,"scratch");
        map.put(2,"IamOverriddeen");
        System.out.println(map.size());//2
        System.out.println(map.get(2)); //scratch
        Ctrl⇧L to Cascade, Ctrl+I to Command
    }
}
```

```
"C:\Program Files\Eclipse Adoptium\jdk-21.0.9.10-hotspot\bin\java.exe" ...
2
IamOverriddeen
```

Q)how actual get and put method works here?

We should know first below 4 things

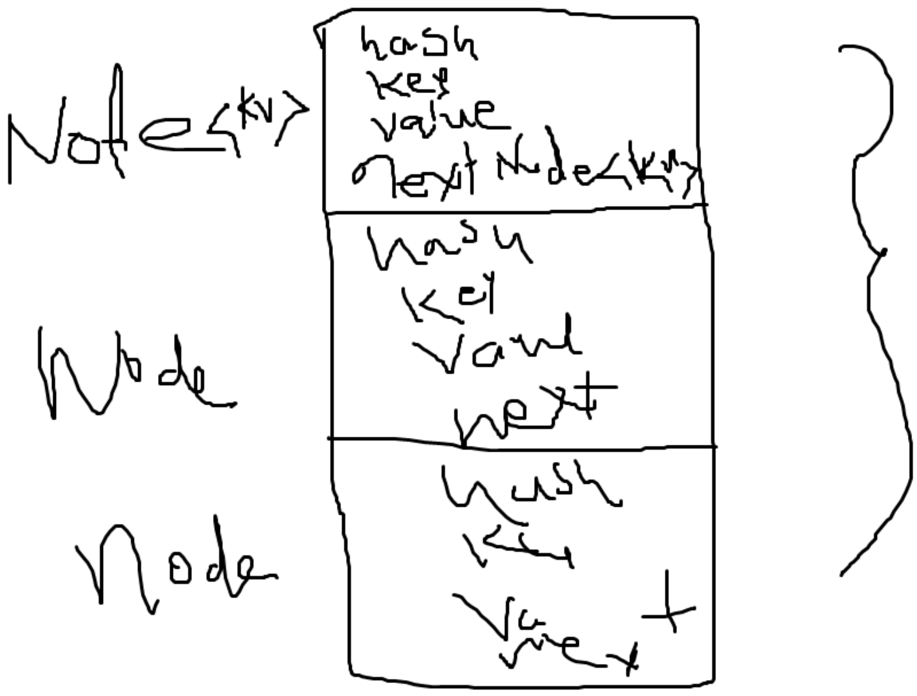
1) **Load factor**
2) **Entry<K,V>**
3) **Re-hashing**
4) **performance**

- inside Map interface, Entry is a sub Interface

- in implementation of this Map interface let's say HashMap is the implementation of this Map interface, inside this HashMap, class, there is another static class called Node, and this static Node class implements the Entry interface

- we have array of Node<K,V> [] → this Node is the static class and it implements Entry interface

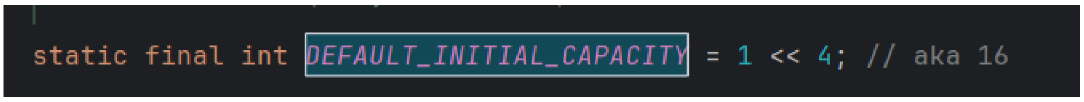
- So HashMap stores data in an array of Nodes



Q) so what will be the initial/default size of a map?

Ans: Map<Integer,String> map = new HashMap();

- when we create object like this, then default initial size will be 16



```
static final int DEFAULT_INITIAL_CAPACITY = 1 << 4; // aka 16
```

- starts fro 0 to 15 is total 16
-

Q) how data is inserted?

- when we do map.put, then what happens?

Putting first element

- eg: map.put(1,"Java");

- as soon as we do map.put then it will do first tep is hash calculation → in hashing it will pass the key and in our case, key is 1→ so for 1, it will compute the hash → there are so many hashing algorithms like MD5, SHA256 etc → so it will use any hashing algorithm and computes a hash→ lets say it has given hash as 1234567→ then it will do hash%size of the table → so in our cse size od the table is 3 → means 1234567%3 → so the calculation wil be like below and the value comes 1 →

$$1234567 \bmod 3 = \boxed{1}$$

Quick check:

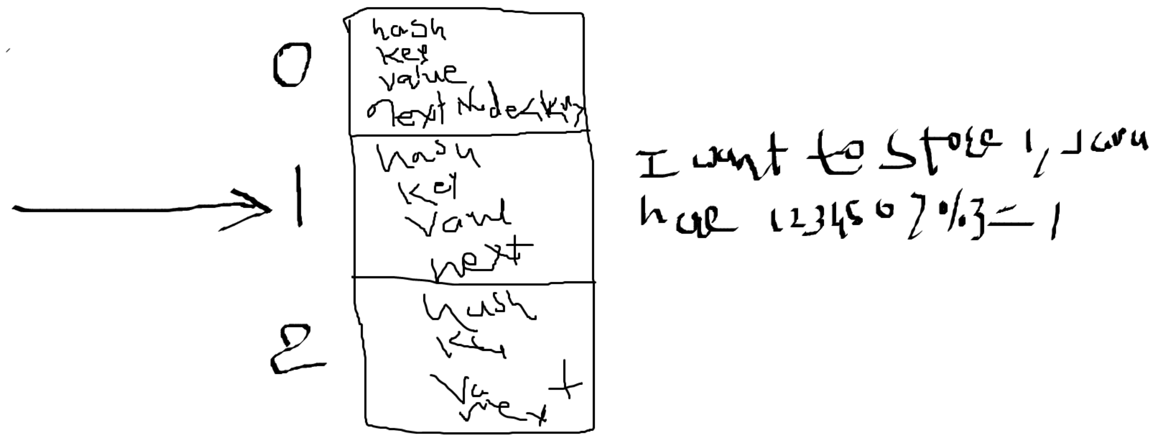
For mod 3, add the digits:

$$1+2+3+4+5+6+7=28$$

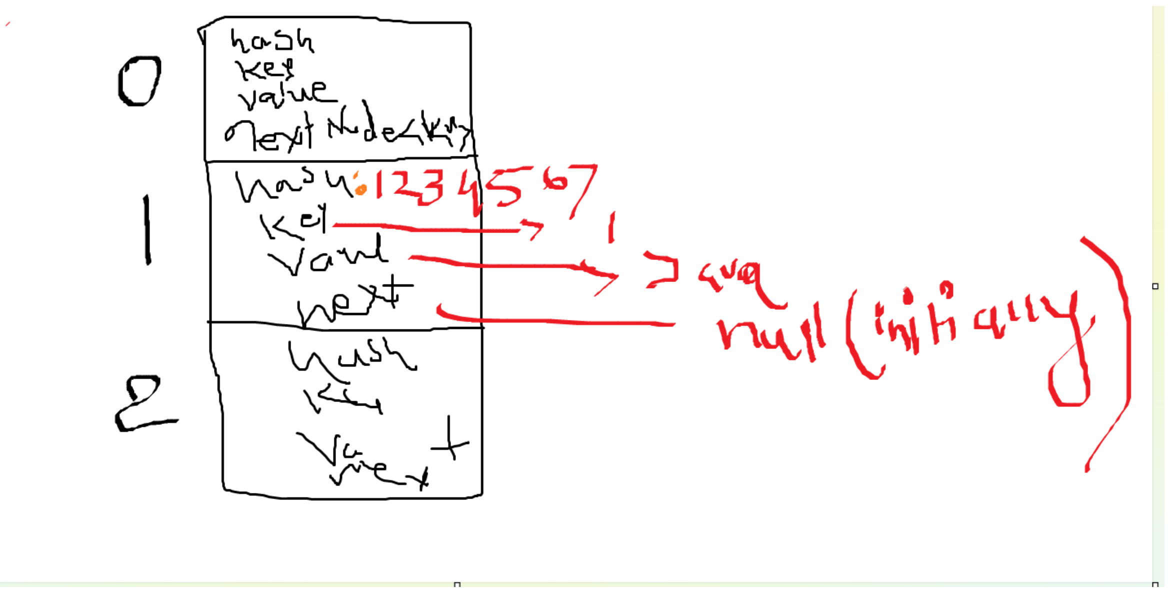
$$28 \bmod 3 = 1$$

So, **1234567 mod 3 = 1** ✅

- ➔ So which means I have to store : map.put(1,"Java"); at position 1



So it stores it like below



Putting second element

- Now, you tries to put another element

```
- map.put(5,"scratch");
```

- now, again it will calculate the hash for key 5 – depending upon algorithm used for hash calculation
- and say hash is 982410 then it will do mod 3 so value comes 0

To calculate $982410 \bmod 3$:

Rule: A number is divisible by 3 if the sum of its digits is divisible by 3.

Sum of digits:

$9 + 8 + 2 + 4 + 1 + 0 = 24$

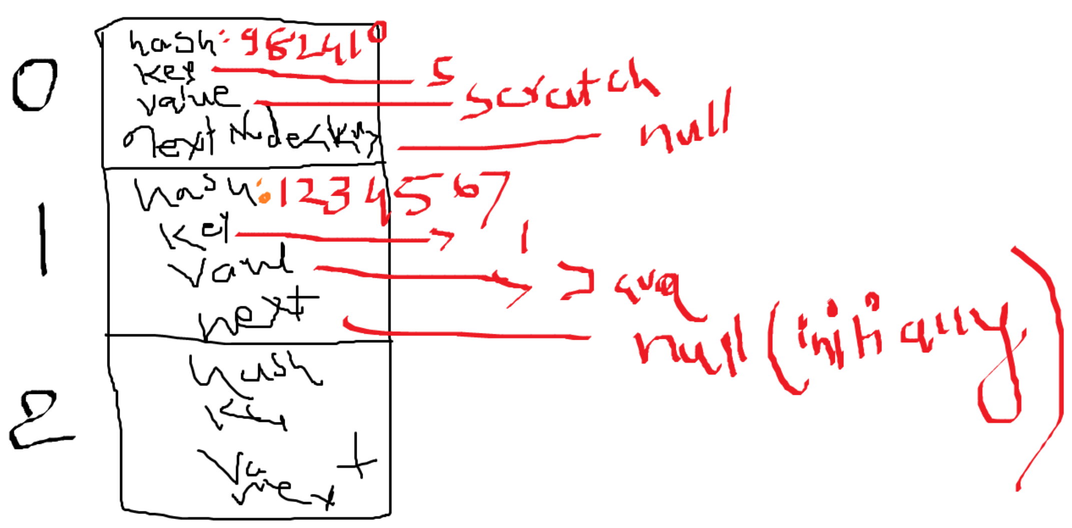
Now,

$24 \bmod 3 = 0$

✅ **Final Answer:**

$$982410 \bmod 3 = \boxed{0}$$

- so it will store at the 0th index



HASH_COLLISION

Q) what if collision happens

- let's say we have to put map.put(10,"javaValue");

- and let's say hash is 31 mod 3 = 1

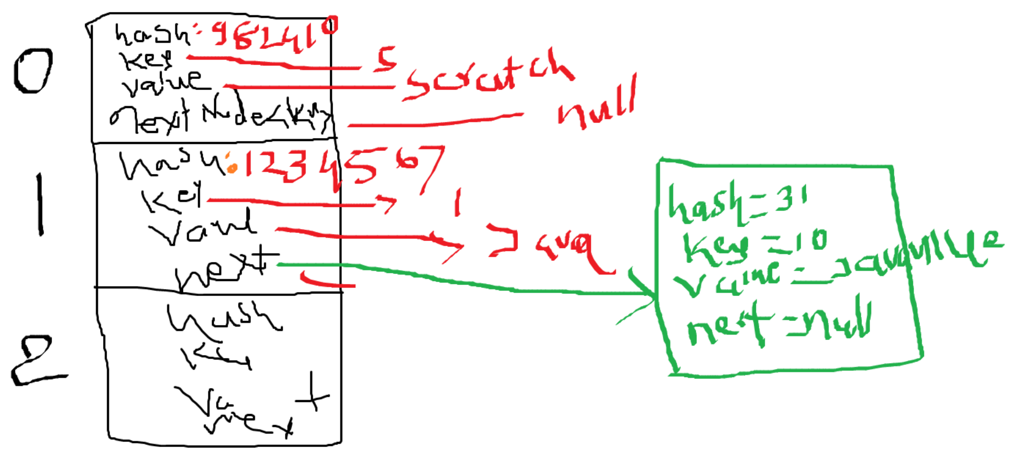
To calculate $31 \bmod 3$:

$$31 = 3 \times 10 + 1$$

✅ **Final Answer:**

$$31 \bmod 3 = \boxed{1}$$

- so, it found location as 1 to store the element
- but it will say, at index 1, there is already 1 element present
- then hashmap internally will check if the key is same? which is already present? → so in our case, earlier key was 1 and now key is 10, so key is not same → so by using next, it will create a new node using a LinkedList → lookup time in worst case is here O(n) → which means If many keys collide, performance degrades badly. →



- if there are lot of collations, this list can grow → since this is using LinkedList till java 7, → If many keys collide, performance degrades badly.
- Java 8 introduced **Treeification → Initially — still LinkedList → When collisions happen, nodes are stored as a LinkedList (same as before). → When LinkedList becomes too long → Tree →**

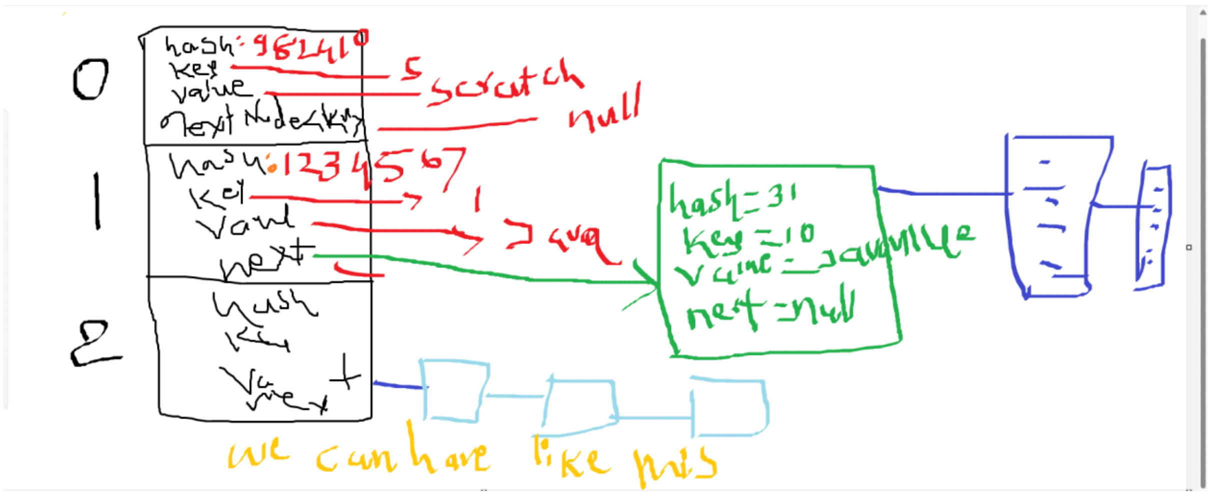
If **both** conditions are met:

Condition	Value
Number of nodes in bucket	≥ **8**
Table capacity	≥ **64**

➡️ The linked list is converted into a **Red-Black Tree**.

```mathematica
bucket[i] → Red-Black Tree
```

- Worst case llookup → O(log n) when we use redblack tree → and performance is much better



Get operation details in hashmap :

- Now, what happens when we do get operation
- Lets say get(1);
- Then it first internally performs hash then it will get same hash which we got while putting the data → that is → 1234567 → hash function should generate the same hash
- So 1234567%3 will become 1

$$1234567 \text{ mod } 3 = \boxed{1}$$

Quick check:

For mod 3, add the digits:

$$1 + 2 + 3 + 4 + 5 + 6 + 7 = 28$$

$$28 \text{ mod } 3 = 1$$

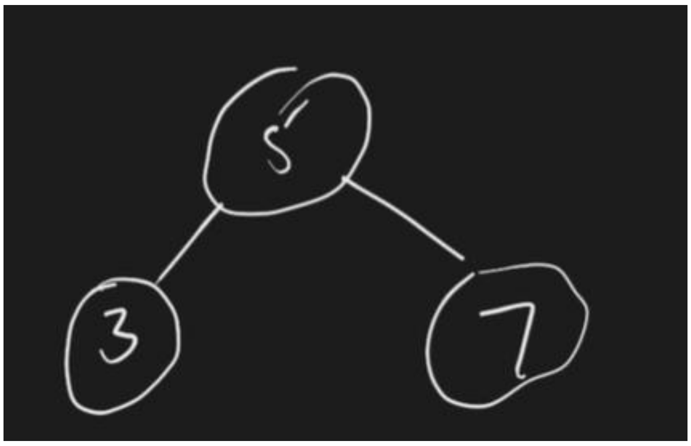
So, **1234567 mod 3 = 1** ✅

-
- So, it will check if key is equals to 1, and if it found that, it will return its corresponding value
- If key is not found, then next node will be searched, and next and next node and son on until it finds the key

Contract between hashcode and equals:

- **If obj 1 and obj2 are same, then their hash should also be same**
- **But is 2 objects's hash is same, dosent mean those 2 objects are same**
- Whenever get method is tries to find the hash of a particular key, it should get the same key which is used while inserting the value → because there is contract between hashcode and equals method → hashcode is the method which helps us to generate the hash → equals is the method in which we are doing comparison in get method

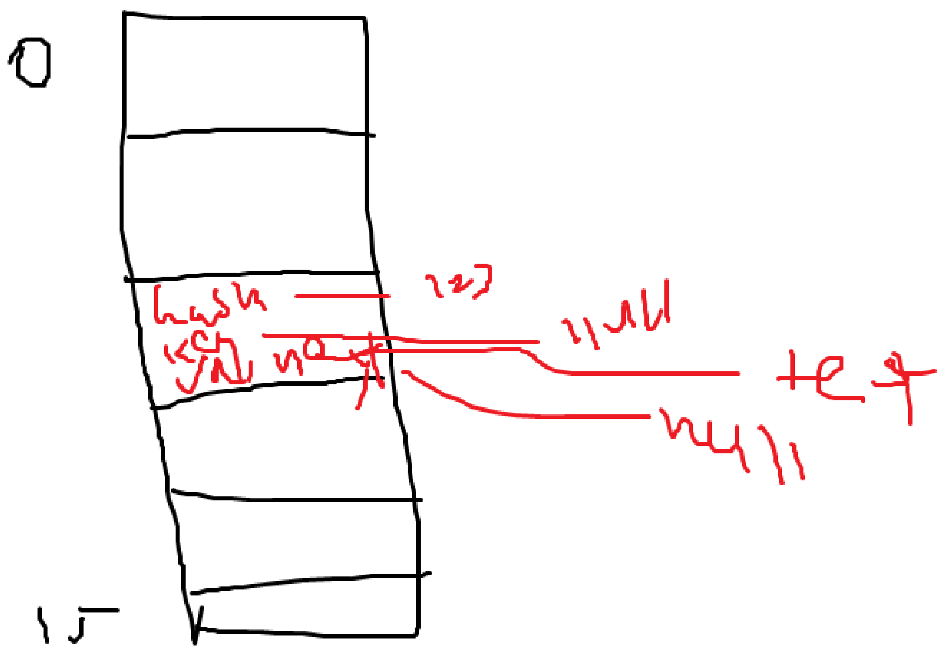
Load factor in hashmap:

- Default load factor is 0.75
- Lets say initial size is 16 0 to 15
- 16×0.75=12
- Means, whenever 12 key value mappings are inserted into this table(hashmap), then it will do rehash → means it will increase the size → or simply it will double the size → so earlier it was 16, now it will become 32 → then again 32*0/.75=23→ so when hashmap size reaches 24, again it will double the size and size becomes 64→ and so on →
- So in this rehash, earlier data will be rehashed again , which means earlier, lets say data was present at bucket location 2, thn it can move to any other bucet location as per re hashing → so in this way, it don't let your linked list grow much →
- Treefy in java 8 → lets say in worst case scenario, you have added some elements and which went in same bucket location lets say 0th location → and as soon as it reaches 8 → means total key value pair are 8 → and now, you want to insert 9th value → and again lets say index comes 0, → as you reached treefy threashhold 8, → then it will convert linked list into tree → which means left side will be lessa than aprent node and right side will be greater than parent node →
  

-
- So that searching will be o(log n)

Q) what if null is the key in hashmap

- for null, it will take as a key and it will calculate hash of the null → null(hash)%16=lets say 2→ so it got the bucket index as 2 → so it will put null, test



- In hashmap, we can have key as null as well value as null as well

Q) what is the return type of EntrySet?

- it will return array of Entry<K,V>
- which means basically it will return all the nodes
- and nodes, we know we have, hash, key, value, and next
- so we have specific method available to get the key and value

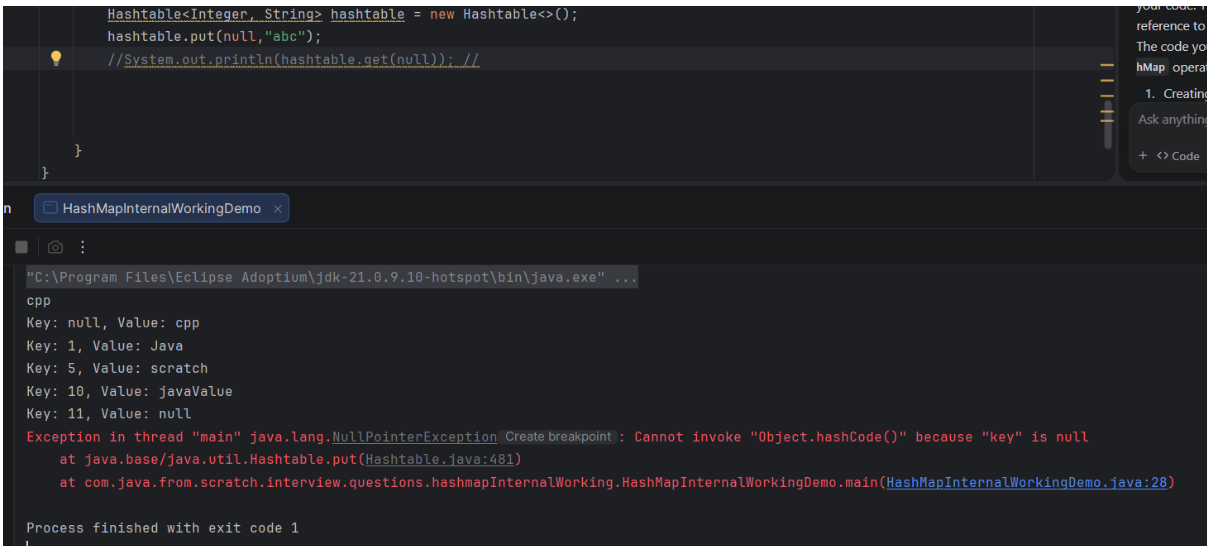
Q) is hashmap Thread safe?

- No , hashmap is not thread safe ,

Q) what is hash table?

hashtable is thread safe, and its synchronized version of HashMap

- hashtable do not contain null key or null value
- whenever you tries to put the null key in hash table, you will get below error at runtime
- hash map does not main tin insertion order,



```
        Hashtable<Integer, String> hashtable = new Hashtable<>();
        hashtable.put(null,"abc");
        //System.out.println(hashtable.get(null)); //

    }
}
```

HashMapInternalWorkingDemo

```
"C:\Program Files\Eclipse Adoptium\jdk-21.0.9.10-hotspot\bin\java.exe" ...
cpp
Key: null, Value: cpp
Key: 1, Value: Java
Key: 5, Value: scratch
Key: 10, Value: javaValue
Key: 11, Value: null
Exception in thread "main" java.lang.NullPointerException Create breakpoint : Cannot invoke "Object.hashCode()" because "key" is null
    at java.base/java.util.Hashtable.put(Hashtable.java:481)
    at com.java.from.scratch.interview.questions.hashmapInternalWorking.HashMapInternalWorkingDemo.main(HashMapInternalWorkingDemo.java:28)

Process finished with exit code 1
```

-