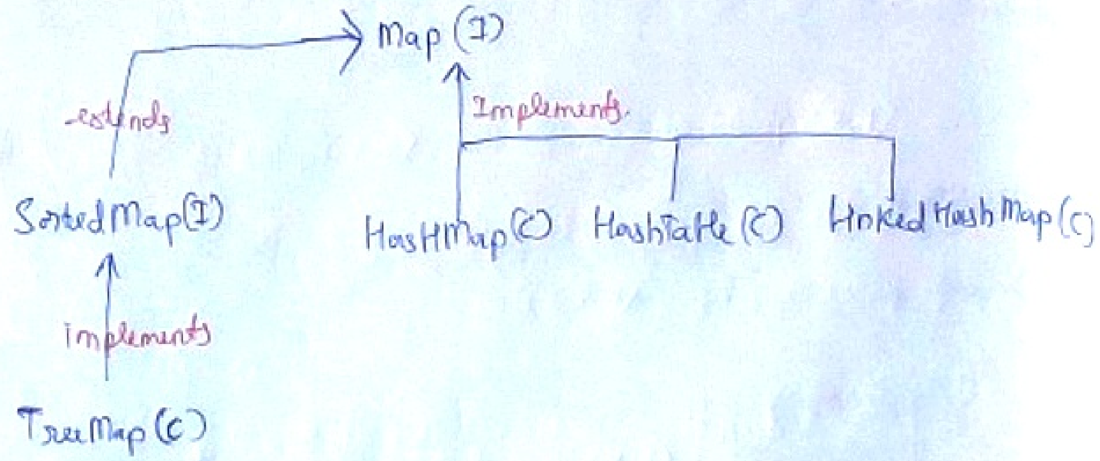
25. Collections in Java - Part 4

HashMap Internal Working in Java



Q) Why is map not associated with Collection?

A) All the interfaces / concrete classes in Collection are dealing with value / values. That is one entry has value only.

Wherease in Map it is associated with Key - Value. So we need total different methods. So collection methods don't have much value.

Map Properties:

- HashMap → does not maintain order
- HashTable → Synchronised version of HashMap
- LinkedHashMap → Maintains the insertion order
- TreeMap: sorts the data internally.
- Cannot contain duplicate key → (Value is overwritten if same key is inserted)

HashMap Internal Design.

- Load factor
- Entry<K,V> interface
- re-hashing
- performance

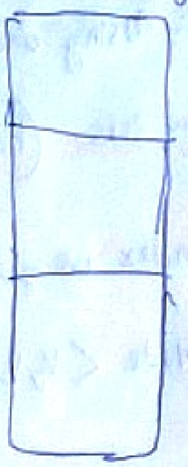
Entry <K,V> interface

This is a sub-interface, inside the Map Interface, Map <K,V>

```
interface Map<K,V>{

    interface Entry <K,V> {
        ⇓
          Sub/Nested
              Interface

    }

}
```

How Hashmap Stores

array of Entry<K,V>



P.T.O

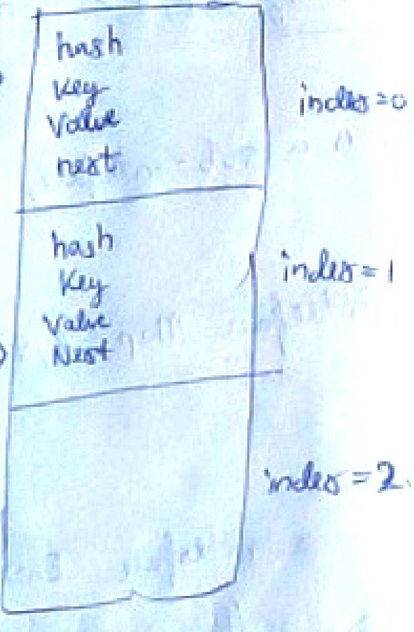
How the HashMap is implementing the Entry<K,V> interface.

public class HashMap<K,V> extends AbstractMap<K,V>
implements Map<K,V> , Cloneable, Serializable {

Node<K,V> implements array of Entry<K,V>

Map<String,Integer> mp = new HashMap<>();
DEFAULT_INITIAL_CAPACITY =16

Node<K,V> has



Lets say we create a HashMap, Map<Integer,String>

Steps:

- Create a Hash: Using any hash Function,

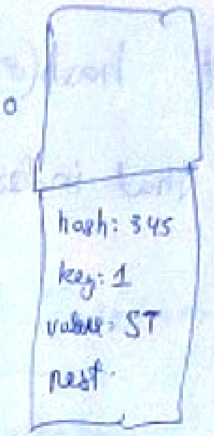
Now this (hashValue % sizeof Hashmap) = index.

Whatever index we get, we try to insert it into that index of the Array of Node <K,V>

- ~~What happens~~

What happens in case of a collission?

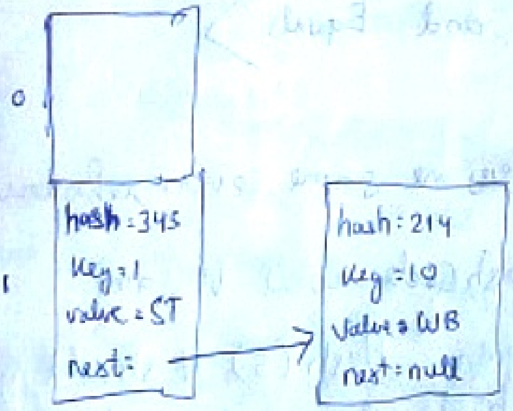
Lets say mp.put(1, "ST")
was inside index 1



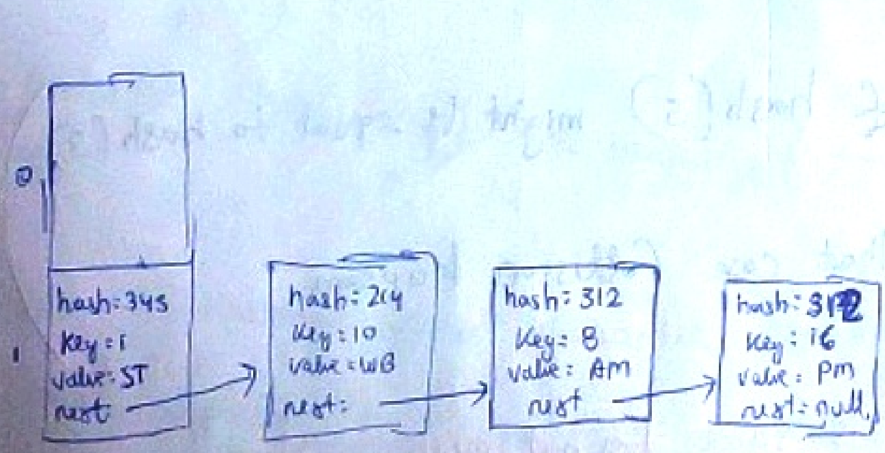
Now we do a mp.put (10, "WB"),
and after hash calculation we get
the index 10 again. what happens then,

[Collision]

It starts creating a LinkedList on the same index.



Upon more Collisions, it will keep adding to the LinkedList

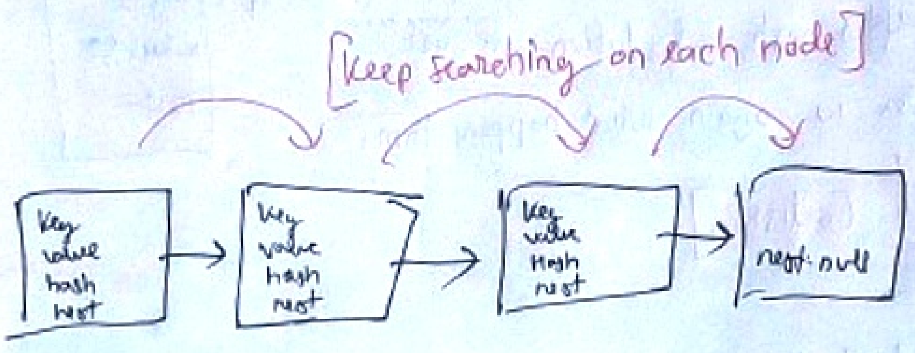


Now lets say on we did a mp.get(8);

if first hash(8) = somvalue will be generated. and from that

From that index will be generated

[Keep searching on each node]



⟨Contract b/w hashCode and Equals⟩

* if (obj1 = = obj2) all valeus are same, even reference is same, then their hash Code will be same.

[hash(5) = x again Hash (5) it should give x]

* if 2 objects have the same hashcode, It does not mean that the objects will be same.

(~~get~~ if hash(5) might be equal to hash(3)

In that case Collision happens.)

If hashValue is same does not mean

Objects are same.

Based on the previous LinkedList. Ideally the complexity should be O(N).

So worst time Complexity is O(N).

But how is that average Complexity. Insertion, Deletion & find in O(1) time complexity.

(How is this handled)

There is a default value in Java ~~called~~ Hashmap called.

LoadFactor : 0.75f, Initial Size of Map = 16

(Step 1) Threshold = Size of Hashmap $\times$ LoadFactor.
= 12.

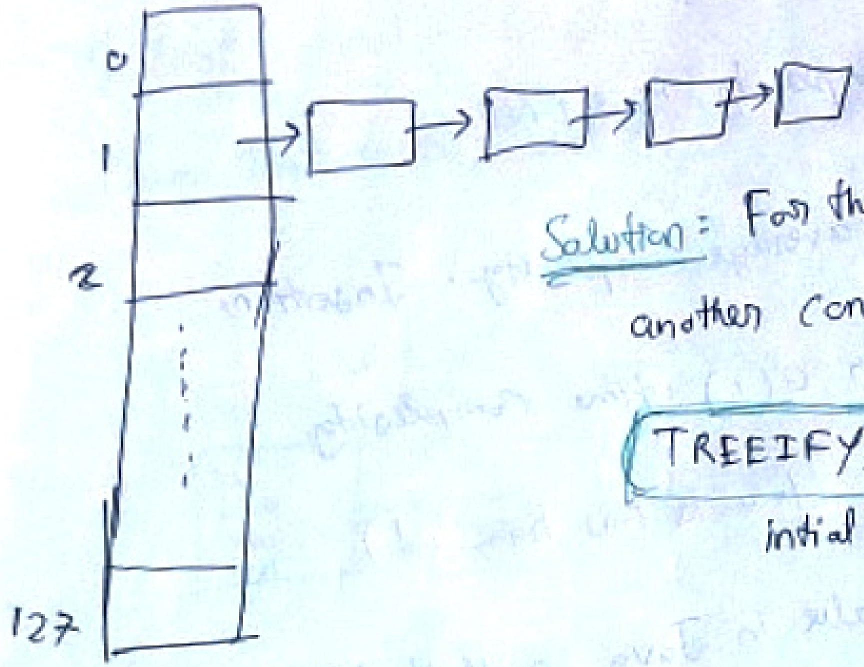
* As soon as the Hashmap has more elements than the threshold. The size is doubled and ~~[illegible]~~ Re-Hashing is done.

index = {hash (Key) % new_size}

****

Above Re-Hashing does not gurantee removing Linear Complexity.

lets say size of HashMap is 128



Solution: For this HashMap has another constant variable

TREEIFY_THRESHOLD

intial value = 8

- Whenever the LinkedList size reaches the TREEIFY_Threshold then the LinkedList is converted into (Balanced Binary Search Tree (AVL Tree))

- Now Time Complexity becomes O(logN)

ConcurrentHashMap / HashTable is the thread-safe version of HashMap

```
Hashmap<Integer, String> mp = new HashMap<>();
mp.put(null, "Test");
mp.put( 0, null);
```

You can have key and value both as null in Hashmap