HashMaps (Detail Understanding and Related Interview Questions)

How does HashMap work? :- It's a MAP implementation

⇒ A Map is an associative array data structure.

"Key1" → value , "Key2" → value , and so on.

(Key are probably string or char.)

stores key value pair. so later you can see value using its key.

⇒ Hashing :- transformation of a string of characters (Text) to a shorted fixed-length value that represents original string. A shorter value helps in indexing and faster searches.

⇒ You can imagine Hashmap or Map as a dictionary.

⇒ So 'hashMap' is based on the technique called hashing.

⇒ So by hashing (converting large string to small integer value) will help us in faster indexing.

10 ★★★★

⇒ In Java every object has a method `public int hashcode()` that will return a hash value for given object.

e.g. if you call hashcode() method of API in a String object you will get an int value of it.

⇒ There is a 'equals' and 'hashcode' contract (Method) that exists in Java that basically goes in following way:-

If two objects are equal they should have the same hash code as well.

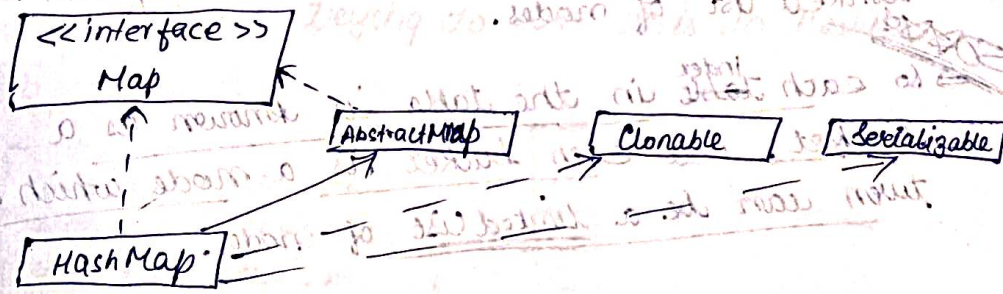
→ So it's very important to have robust hash code implementation in your class and if you've difficulty in implementing your own hash code for example you are implementing a account class, and you are not able to provide a very good hash code implementation; then you can always let the IDE generate for you and they do a good job in generating hashcode implementations.

⇒ Why is it important?

It is important because the hashcode is used in storing values into the hashmap; so when we look up values corresponding to a given key & if the hashcode is not consistent, you won't be able to look up the corresponding value ever.

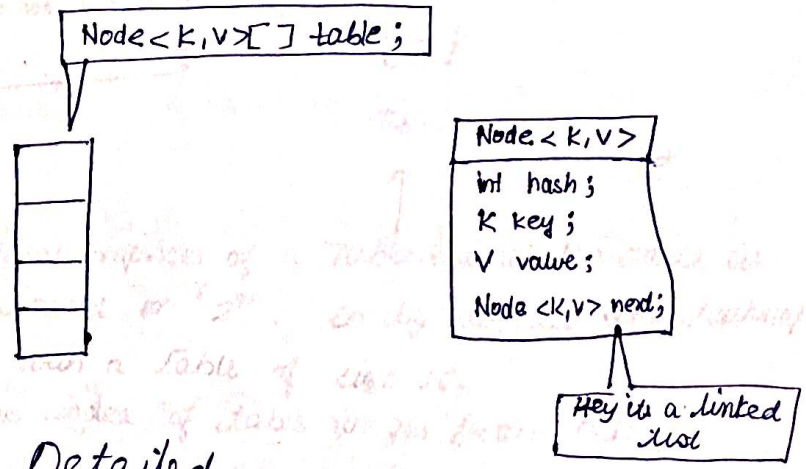
Implementation details:-

HASHMAP



High Level

⇒ At low-level:- HashMap has a table an array of nodes and nodes a basically int hash key, the key you sent to the hash map, the value you will add to the table and a pointer to the next node; so basically the node itself is a linked list inside the tail.

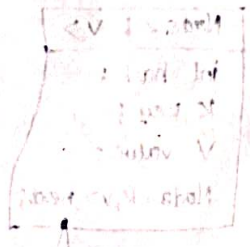


Detailed

⇒ So each index in this array that you can imagine as a node which actually can be linked list of nodes.

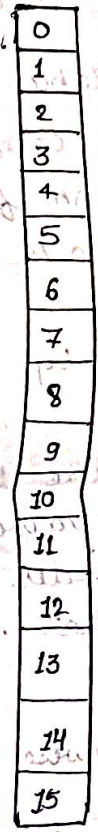
So each index in the table is known as a bucket, and each bucket is a node which in turn can be a linked list of nodes.

§ Let's see how this implementation works in terms of the put operation.



§ e.g. We are trying to put a person name and the score he got in some Game. We are trying to store this in HashMap.

```
scores.put("KING", 100);
scores.put("CLARK", 90);
scores.put("BLAKE", 10);
"  "  .("FORD", 110);
"  "  ("SMITH", 10);
"  "  ("WARD", 99);
"  "  ("JONES", 99);
```




```
put(K k, V v)
    hash(K)
    index = hash & (n-1)
```

n=16

Table size = 16

So HashMap comprises of a Table; and the table is sized based on '2^m'. So by default the hashmap comes with a table of size 16.
So the index of table ranges from 0 to 15.

⇒ So we need put the entries in the table and see how it works. When the put operation happens.

for

⇒ "KING" in put method

hash ("KING") = 2306996

↳ Now we cannot have array of this size in java.

So hashmap table is size to 2^n and now run a index compilation to find out where exactly we can fit this hashcode in our table of range 0 to 15.

So we do this by using modulo operation to get index.

⇒ So we divide hashcode using the in a maximum value of the range and you get the remainder. And remainder value will be value that you can fit within the range.

So we use bitwise operation as:-

index = hash & (n-1)

It's like

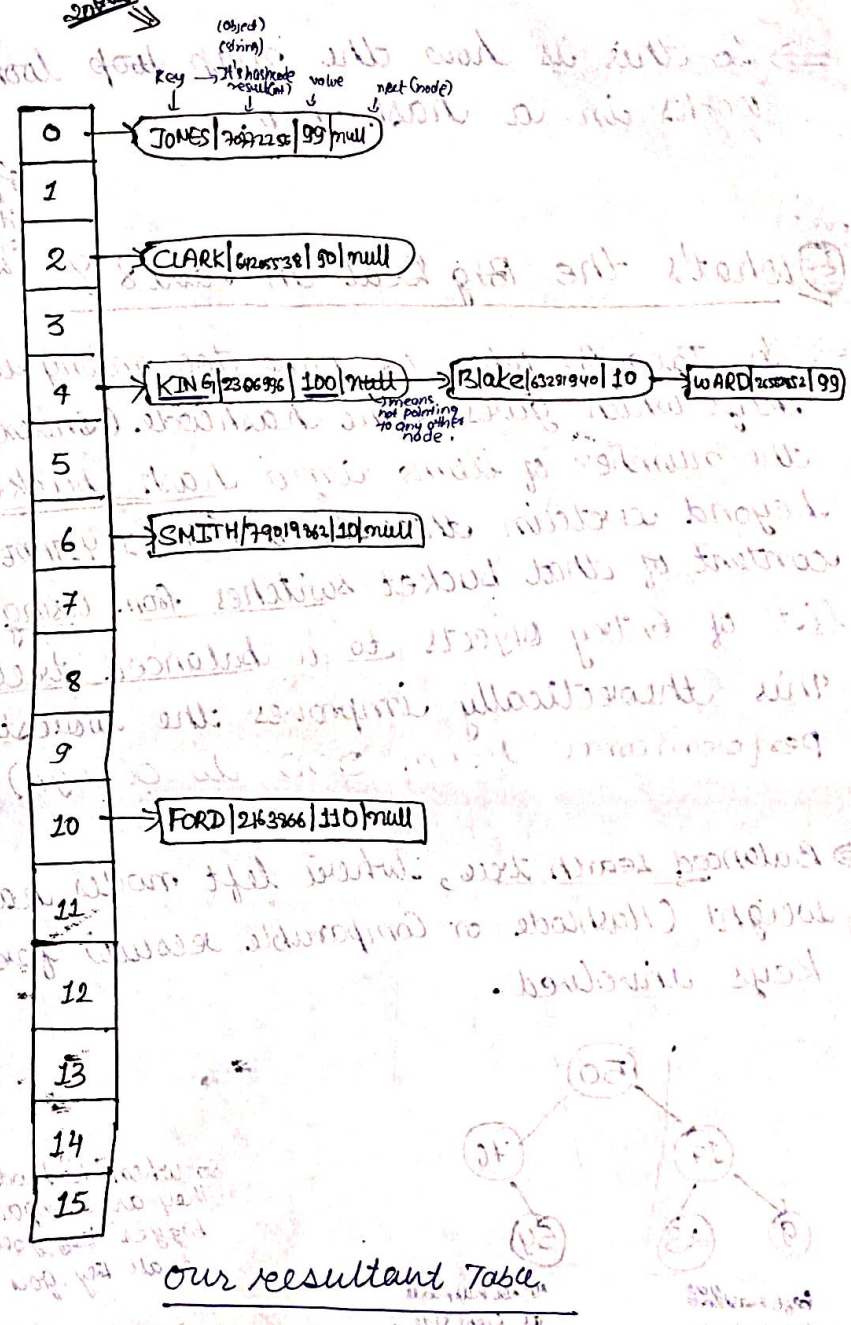
index = hash % 15

So on doing this operation & on our hash (2306996) we get index = 4

or 2306996 % 16 ⇒ 4.

(since 16 box (bucket) in table. 0-15 total 16)

That means entry will go to index '4' as new node.



Our resultant Table

⇒ Similarly we do 'put' method and get index for each.

⇒ Whenever there is collision (means index same for multiple key) we will just add it as next node of the already existing node.

⇒ So this is how the map ~~loop~~ look up works in a hash map.

5***

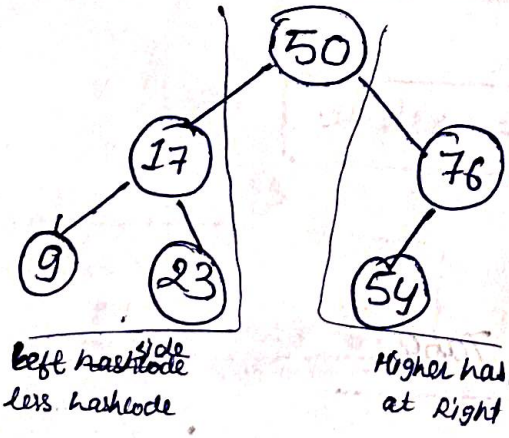
(8) What's the Big Deal in Java 8 ?

5***

bucket means it will have similar hashes in it

⇒ In Java 8, when we have too many unequal keys which gives same hashcode (Index) - where the number of items in a hash bucket grows beyond certain threshold (TREEIFY_THRESHOLD = 8), content of that bucket switches from using a linked list of Entry objects to a balanced tree. This theoretically improves the worst-case performance from O(n) to O(log n).

⇒ Balanced search tree, where left nodes have lesser weight (Hashcode or Comparable result) for the keys involved.



So when hashcode some then they are compared & bigger key goes to Right & small key goes to left.

⇒ So it's increases the tree efficiency.

⇒ So this is the enhancement in Java 8.