HashMap

* Extends AbstractMap
* Has an array of nodes ( Node has K key ,V value and int hash ,Node next )
* Hashing – process of converting an object into integer form by using the Hashcode method.
* Bucket is an element of HashMap array to store the node
* Single bucket can have more than one node (depending on Hashcode)
* Buckets are different in capacities.
* Capacity = number of Bucket \* capacity
* Index = Hashcode (key) & (N-1)
* Allows null key
* Initial size =16
* Load factor =0.75 (75 % of capacity)
* Threshold = current capacity \* load factor
* The number of entries in HashTable that exceeds the Map is rehashed, twice the number of buckets as before.
* Rehashing 🡪 recalculating the Hashcode value of already stored entries.
* Collision occurs when the hash function returns the same bucket location with 2 different keys.
* While Collision it checks for *Hashcode() and equals()* method that both the key are same.
  + If keys are the same, replace the value with the current value.
  + Otherwise, connect the node object to the previous node object through a linked list so both are stored at the same index.
* Hashmap initially uses the linkedList
  + When the entries crosses certain threshold , it will replace the linkedlist with balanced binary tree(red-black Tree)

Methods :

*Get*

*Put*

*Getor Default*

*Remove*

*Containskey*

*containsValue*

*isEmpty*

*size*

complexity

O(1) for insertion and lookup

**Top 10 HashMap questions in Java — Interview Preparation Guide — Part 2**

Recently a lot of interviewers are asking for the internal working mechanism of the hashmap. This is a continuation of [**my first article**](https://medium.com/@devesh.k.chaubey/top-10-hashmap-questions-in-java-interview-preparation-guide-part-1-342c98099060) on the same topic. With this article we will try to understand and answer below questions:

1. What is HashMap
2. Does it allow Null Keys/Values
3. Does it allow duplicate Keys
4. Does it maintain order of inserted elements
5. Is HashMap thread safe
6. Why it should be avoided to be used in Multi-threaded env, Solution
7. HashMap vs ConcurrentHashMap
8. HashMap vs HashTable
9. What is Capacity and Load Factor
10. Common methods used and its time complexity
11. Internal implementation

**Common methods and Time complexities**

Table

Description automatically generated

1. **HashMap**uses **hashCode()** and **equals()** methods on keys for the **get** and **put**operations. So HashMap key objects should provide a good implementation of these methods.
2. That’s why the **Wrapper** classes like **Integer** and [**String**](https://levelup.gitconnected.com/string-stringbuilder-and-stringbuffer-a-complete-guide-7029c1953f7e) classes are a good choice for keys for HashMapas they are[**immutable**](https://medium.com/javarevisited/confused-with-immutable-class-in-java-why-string-is-immutable-get-it-clear-7edbee856c9)and their object state won’t change over the course of the execution of the program.
3. In a fairly distributed hashMap where the entries go to all the buckets in such a scenario, the hashMap has **O(1)** time for *search, insertion*, and *deletion* operations.
4. In the worst case, where all the entries go to the same bucket and the singly linked list stores these entries, **O (n)**time is required for operations like *search, insert,*and*delete.*
5. In a case where the ***threshold***for converting this linked list to a self-balancing binary search tree(i.e. AVL/Red black) is used then for the operations, *search, insert*and*delete* **O(log(n))** is required as AVL/Red Black tree has a max length of **log(n)** in the worst case.

**Internal implementation**

Before moving to internal implementation we have to understand few basic questions:

**1) What is bucket in hashMap?**

*HashMap contains an array of Nodes, which is called a*bucket *and each node is represented as a class having the four objects i.e int hash, K key, V value, Node next.*

**2) What is hash/hashing?**

Hashing is the process of transforming any given key or a string of [characters](https://www.techtarget.com/whatis/definition/character) into another value. This is usually represented by a shorter, fixed-length value or key that represents and makes it easier to find or employ the original string.In Java hashing converts the object into an integer form by using the method *hashCode()*.

**3) What is hashCode()?**

HashMap works on the basis of hashing which uses the *hashCode()* from Object class. You can override this *method* to write your own Hashing algorithm to get unique integer value, which will be used for deciding which bucket (the index)an object should be placed into.

**4) What is equals()?**

The default implementation of *equals()* in the class *Object* says that equality is the same as object identity. This method is provided by the Object class. You can override this in your class to provide your implementation. HashMap uses equals() to compare the key to whether they are equal or not. If the equals() method return true, they are equal otherwise not equal.

**5) What is Initial Capacity?**

Capacity is the number of buckets in the *HashMap*. The initial capacity is the capacity at the time the *Map*is created. Finally, the default initial capacity of the *HashMap* is 2⁴ i.e.16. As the number of elements in the *HashMap* increases, the capacity is expanded i.e. 2⁵, 2⁶

**6) What is Load Factor?**

The Load factor is a measure that decides when to increase the HashMap capacity to maintain the insertion (i.e put()) and retrieval(i.e. get()) operation complexity of O(1). The default load factor of HashMap is 0.75f (75% of the map size).

The threshold of a *HashMap* is approximately the product of current capacity and load factor. There are two way to determine when to increase the HashMap bucket size.

***Threshold = initial capacity \* Load factor***

1. The initial capacity of hashmap is =16, The default load factor of hashmap=0.75. So when 16\*0.75 =12. So, 12th index key-value pair of hashmap will keep its size to 16. As soon as 13th element (key-value pair) will come into the HashMap, it will increase its size from default 16 buckets to 32 buckets following 2^n.

***m/n > 0.75*** // increases the HashMap bucket size/capacity.  
where m = number of entries in a hashmap  
n = total size of hashmap/bucket size

2. After inserting the first element by checking the hash code of key, it checks whether increase of the hashmap capacity required or not by using the formula m/n.

Now we comfortable with core concepts lets understand internal working.

**Before Java8:**

* Internally HashMap uses a **hashCode**of the key Object and this hashCode is further used by the **hash function** to find the index of the bucket where the new entry can be added.
* HashMap uses multiple buckets and each bucket points to a **Singly Linked List** where the entries (nodes) are stored.
* Once the bucket is identified by the hash function using hashcode, then hashCode is used to check if there is already a key with the same hashCode or not in the bucket(singly linked list).
* If there already exists a key with the **same** **hashCode**, then the **equals()** method is used on the **keys**. If the equals method returns **true**, that means there is already a node with the same key and hence the value against that key is **overwritten**in the entry(node), otherwise, a new node is created and added to this Singly Linked List of that bucket.
* If there is no key with the same hashCode in the bucket found by the hash function then the new Node is added into the bucket found.

Diagram, timeline

Description automatically generated

**After Java8:**

Before java 8, singly-linked lists were used for storing the nodes. But this implementation has changed to self-balancing BST after a thresold is crossed (**static final int TREEIFY\_THRESHOLD = 8;**). The motive behind this change is that HashMap buckets normally use linked lists, but for the linked lists the worst-case time is O(n) for lookup. Also note that Ordinary binary search trees have pathological cases where they become O(n) [basically BST becomes skewed], but red-black/AVL trees are specifically designed to prevent these cases. In a HashMap with linked lists, if we have a really an awful hash function, we could end up with all the items hashing to the same bucket and get O(n) lookup, But it seems like with this red-black/AVL tree scheme, even if all the items hashed into the same bucket, we would get O(log⁡n) lookup in worst of worst scenario.

Diagram

Description automatically generated

**Conclusion**

Hash is one of the important and frequently used data structure. With this article I had tried to cover all the aspects of it. Hope it makes you more comfortable in using it and clearing interviews.