How HashMap Works Internally?

To understand this first we should know internal structure of the HashMap.

HashMap stores data in from of key-value pair. Each key-value pair is stored in Object of Entry<K,V> class.

Let's snippet of Entry class-

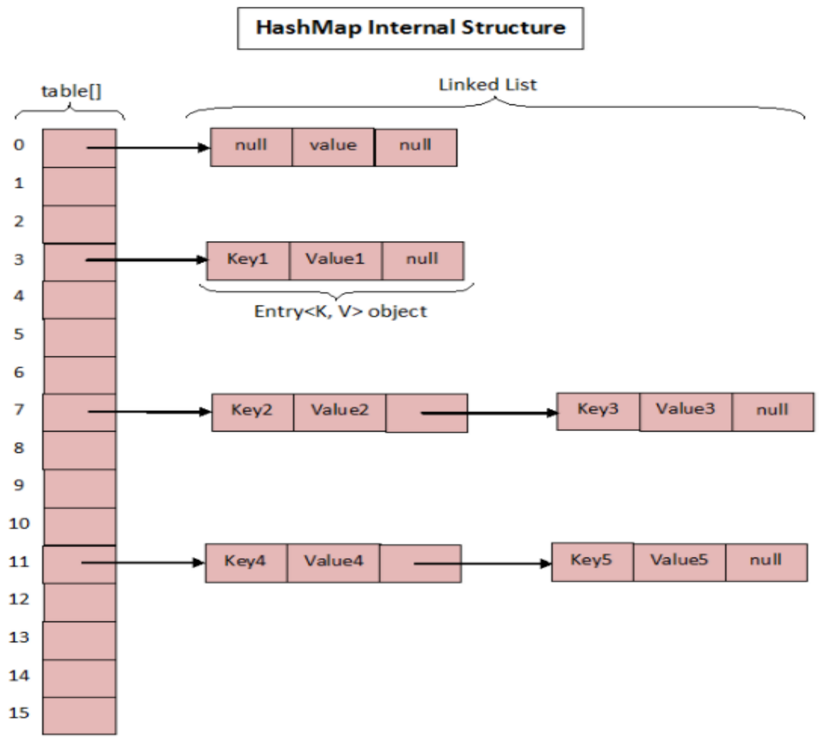
```
static class Entry<K,V> implements Map.Entry<K,V>
             {
                     final K key;
                     V value;
                     Entry<K,V> next;
                     int hash;
             }
```

- Key- It stores key of the Object and it is final.
- Value- It stores value for the key.
- **<u>Next- It holds pointer to next key-value pair. This attribute makes the key-value pairs stored as a linked list.</u>**
- Hash – It stores hash code for the key.

These Entry Objects are stored in an array called table[]. Initial and default size of the array is 16.

Note- The table can be resized but length must be power of 2.

Below image describes the internal structure of the HashMap.



Now Entry will be stored in these 16 buckets/index. But which Entry will be placed where, decided by Hashing Principle.

What is Hashing Principle?

Hashing is an Algorithm to find unique integer value representation of an Object.

- Hashing Algorithm said to be the best if it returns the same hash code each time it is called on the same object. **Two objects can have same hash code in Java**.
- This hash code is passed in indexFor() method to calculate the index in table[] array.
- HashMap has its own method hash() to calculate the hash code. It is different from hashCode() method of Object class.
- For a `null` Object hash() method of HashMap returns 0 while hashCode() method of Object class throws NullPointerException.

```
 String str=null;
             System.out.println(str.hashCode());
```

Throws NullPointerException.

How put() method works?

- First it calculates hash Code for the key using hash() method.
- It calls indexFor() method by passing the hash code and length of the table. This method returns the index where this Entry will be stored.

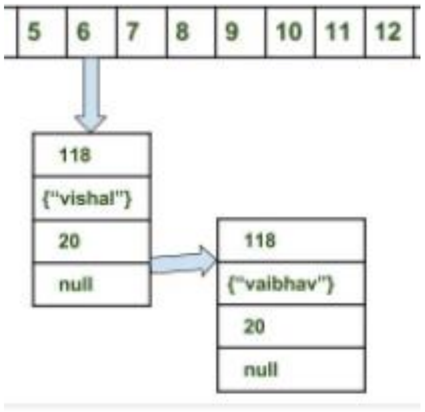
```
static int indexFor(int h, int length) {
 return h & (length-1);
```

- It checks whether the key is null or not. If it is null, calls putForNullKey() method which will return 0. The entry will be stored at table[0] because hash code for `null` is 0.
- If not null calculates the hash code and passes it to indexFor() method to calculate the index where new Entry will be stored.
- After getting the index, it checks all keys (in same bucket there may be more than one key present) present in the linked list at that index (or bucket). If the key is already present in the linked list, it replaces the old value with new value.
- If the key is not present in the linked list, it appends the specified key-value pair at the end of the linked list.

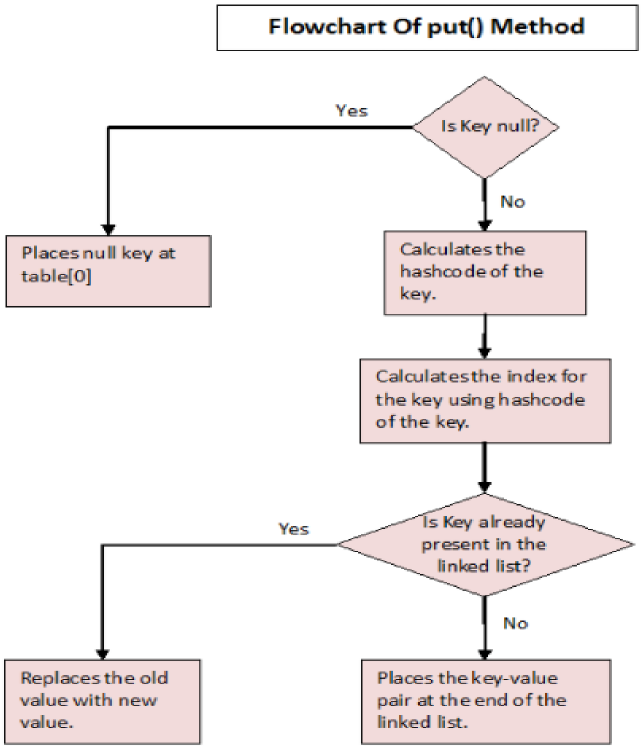
Let's an Example-

```
Map<String, String> hm=new HashMap<>();
             hm.put(null, "sanjay");
             hm.put("01", "sanjay");
             hm.put("01", "Rai");//Since key 01 is already present, sanjay will be
replaced by Rai
             System.out.println("Map Size : "+hm.size());
             System.out.println("Elements in HashMap : "+hm);
             //Output
             Map Size : 2
             Elements in HashMap : {null=sanjay, 01=Rai}
```

- What happens when two Objects have the same hash code. Both will be stored in the same bucket in form of LinkedList. Entry class has attribute next which points the next Object. Below is the diagram. For Vishal and Vaibhav hash code is same.



Let's understand the same using a flowchart.



Flowchart Of put() Method
Is Key null?
Yes
No
Places null key at table[0]
Calculates the hashcode of the key.
Calculates the index for the key using hashcode of the key.
Is Key already present in the linked list?
Yes
No
Replaces the old value with new value.
Places the key-value pair at the end of the linked list.

Note - Capacity of each bucket may differ - capacity = number of buckets * load factor.

How get() method works?

- First checks whether specified key is null or not. If the key is null, it calls getForNullKey() method.
- If the key is not null, hash code of the specified key is calculated.

- indexFor() method is used to find out the index of the specified key in the table[] array.
- After getting index, it will iterate though linked list at that position and checks for the key using equals() method. If the key is found, it returns the value associated with it. otherwise returns null.

What happens during get() method when two objects have the same hash code?

If both the Objects have same hash code then they will be stored at the same index. In our example Vishal and Vaibhav are stored at index 6.

Here Keys.equals() method will be called to identify the correct key and corresponding value will be returned.

How null key is handled in HashMap? Since equals() and hashCode() are used to store and retrieve values, how does it work in case of the null key?

The null key is handled specially in HashMap, there are two separate methods for that putForNullKey(V value) and getForNullKey().

Why do we do Hashing?

Using this a huge string can be represented as small fixed length value which helps in indexing and searching.

What is change in Java 8 for HashMap?

In Java 8 when too many unequal keys produces same hashcode i.e number of elements in a buckets grows beyond a certain limit (TREEIFY_THRESHOLD=8) content of that bucket switches from LinkedList to Balance Tree which improves the performance.