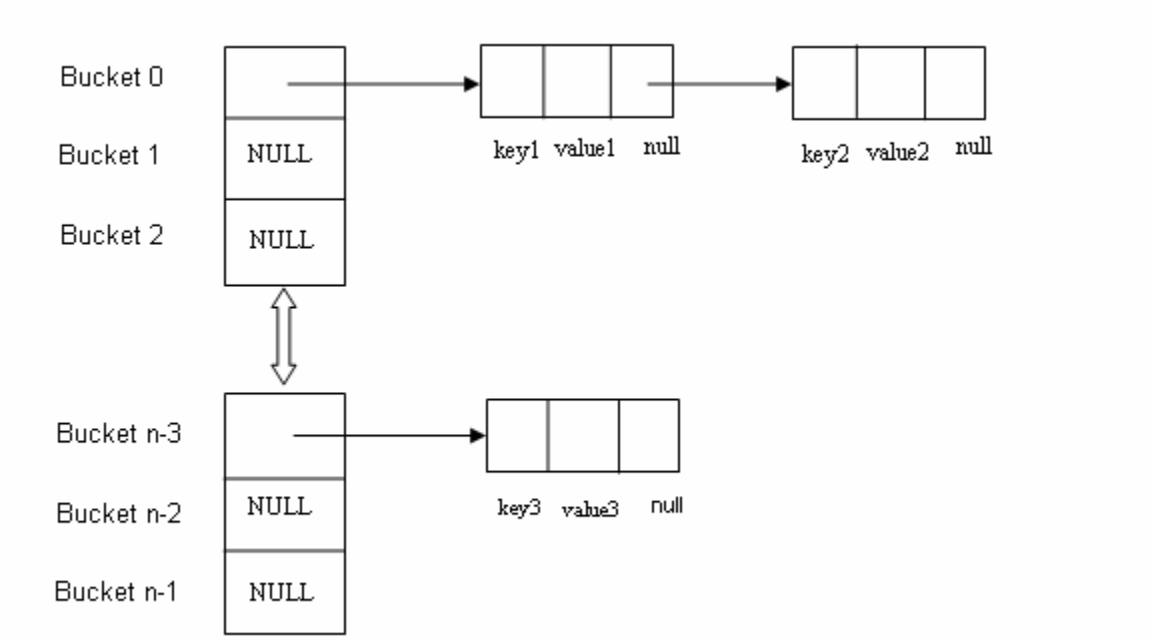
How HashMap internally works in Java

There are four things we should know about before going into the internals of how does **HashMap** work in Java -

- **HashMap** works on the principal of hashing.
- **Map.Entry interface** - This interface gives a map entry (key-value pair). HashMap in Java stores both key and value object, in bucket, as an object of Entry class which implements this nested interface Map.Entry.
- **hashCode()** -HashMap provides put(key, value) for **storing** and get(key) method for**retrieving** Values from HashMap. When put() method is used to store (Key, Value) pair, HashMap implementation **calls hashcode** on Key object to calculate a hash that is used to find a bucket where Entry object will be stored. When get() method is used to retrieve value, again key object is used to calculate a hash which is used then to find a bucket where that particular key is stored.
- **equals()** - equals() method is used to **compare objects for equality**. In case of HashMap key object is used for comparison, also using equals() method Map knows how to handle**hashing collision** (hashing collision means more than one key having the same hash value, thus assigned to the same bucket. In that case objects are stored in a linked list, refer figure for more clarity. Where hashCode method helps in finding the bucket where that key is stored, equals method helps in finding the right key as there may be more than one key-value pair stored in a single bucket.

** Bucket term used here is actually an index of array, that array is called table in HashMap implementation. Thus table[0] is referred as bucket0, table[1] as bucket1 and so on.

- Refer Overriding hashCode() and equals() method in Java to know more about hashCode() and equals() method



HashMap changes in Java 8

Though HashMap implementation provides constant time performance O(1) for get() and put() method but that is in the ideal case when the Hash function distributes the objects evenly among the buckets.

But the performance may worsen in the case hashCode() used is not proper and there are lots of hash collisions. As we know now that in case of hash collision entry objects are stored as a node in a linked-list and equals() method is used to compare keys. That comparison to find

the correct key with in a linked-list is a linear operation so in a worst case scenario the complexity becomes O(n).

To address this issue in Java 8 hash elements use balanced trees instead of linked lists after a certain threshold is reached. Which means HashMap starts with storing Entry objects in linked list but after the number of items in a hash becomes larger than a certain threshold, the hash will change from using a linked list to a balanced tree, this will improve the worst case performance from O(n) to O(log n).