**Internal Implementation of HashMap**.

**HashMap** in Java works on hashing principle. It is a data structure which allows us to store object and retrieve it in constant time O(1)provided we know the key. In hashing, hash functions are used to link key and value in HashMap. Objects are stored by calling put(key, value) method of HashMap and retrieved by calling get(key) method. When put() method is called to store (key, value ) pair , HashMap implementation calls HashCode() method on key obect to find bucket ,which is actually an index of the internal array, where entry object will be stored. When get() method is used to retrieve value , again key object is used to calculate hash which is used then to find a bucket where that particular key is stored. HashMap internally stores mapping in the form of **Map.Entry** object which contains both key and value object.

Since HashMap is backed up by an array the internal array of HashMap is of fixed size, and if you keep storing objects, at some point of time hash function will return same bucket location for two different keys, this is called collision in HashMap. In this case, a linked list is formed at that bucket location and a new entry is stored as next node.

**Map.Entry Interface** – This Interface gives a map entry(key-value pair). HashMap in java stores both key and value objects, in bucket, as an object of Entry class which implements this nested interface.

**How put() method internally works in HashMap**

Private int hashing(Object key){

Int hashValue = key.hashCode();

Int index = hashValue % numberOfBuckets;

Return index;

}

i. Using HashCode() method hash value will be calculated. Using that hash(index) it will be ascertained, in which bucket particular entry will be stored.

ii. equals() method is used to find if such a key already exists in that bucket.

a. If Yes then new value will overwrite old value for matched key.

b. if No then a new node is created with map entry and within the same bucket.

iii. But there is a possibility that same hash value will be calculated for two or multiple different keys(according to equals() method) that means collision occurred which means multiple entry objects stored in a bucket location. Each entry keeps track of another entry forming LinkedList data structure there.

**How get(Key key) method internally works in HashMap.**

**i.**When get(Obect key ) is invoked, again hashCode() method is called on key object to calculate hash value to determine bucket(index of internal array) where that entry object is stored, in case there are more than one entry objects with in the same bucket stored as linkedlist. Equals() method will be used to find correct key for that need to traverse LinkedList until correct key is found get() method will return value object stored in Entry object.

**In case of Null key**

As we know that HashMap also allows null, though there can only be one null key in HashMap. While storing the Entry object HashMap implementation checks if the key is null, in case key is null, it always map to bucket 0 as hash is not calculated for null keys.

private V getForNullKey() {

if (size == 0) {

return null;

}

for (Entry<K,V> e = table[0]; e != null; e = e.next) {

if (e.key == null)

return e.value;

}

return null;

}

**Why String, Integer and other wrapper classes are considered good keys?**

String, Integer and other wrapper classes are natural candidates of HashMap key, and String is most frequently used key as well because String is immutable and final, and overrides equals and hashcode() method. Other wrapper class also shares similar property. Immutability is required, in order to prevent changes on fields used to calculate hashCode() because if key object returns different hashCode during insertion and retrieval than it won't be possible to get an object from HashMap.

**Necessary point about hacode() and equals() method**

1.If two objects are equal by equals() method then there hashcode returned by hashCode() method must be same.

2.Whenever hashCode() mehtod is invoked on the same object more than once within single execution of application, hashCode() must return same integer provided no information or fields used in equals and hashcode is modified. This integer is not required to be same during multiple execution of application though.

3. If two objects are not equals by equals() method it is not require that there hashcode must be different. Though it’s always good practice to return different hashCode for unequal object. Different hashCode for distinct object can improve performance of hashmap or hashtable by reducing collision.