**How does HashMap Works in Java? (IMP)**

**1. Single statement answer**

If anybody asks me to describe “**How HashMap works?**“, I simply answer: “**On principle of Hashing**“. As simple as it is. Now before answering it, one must be very sure to know at least basics of Hashing. Right??

**2. What is Hashing?**

[Hashing](https://en.wikipedia.org/wiki/Hash_function) in its simplest form, is a way to assigning a unique code for any variable/object after applying any formula/algorithm on its properties.

A true [hash function](https://howtodoinjava.com/security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/) must follow this rule –

“Hash function should **return the same hash code each and every time** when the function is applied on same or equal objects. In other words, two equal objects must produce the same hash code consistently.”

All objects in Java inherit a default implementation of hashCode() function defined in Objectclass. This function produces hash code by typically converting the internal address of the object into an integer, thus producing different hash codes for all different objects.

**3. Entry class in HashMap**

A map by definition is :**“An object that maps keys to values”**. Very easy.. right?

So, there must be some mechanism in HashMap to store this key-value pair. The answer is YES. HashMaphas an inner class Entry, which looks like this:

|  |
| --- |
| Entry class |
| static class Entry<K ,V> implements Map.Entry<K, V>  {      final K key;      V value;      Entry<K ,V> next;      final int hash;      ...//More code goes here  } |

Surely Entry class has key and value mapping stored as attributes. key has been marked as final and two more fields are there: next and hash.

We will try to understand the need of these fields as we go forward.

**4. How HashMap.put() methods works internally**

Before going into put() method’s implementation, it is very important to learn that instances of Entryclass are stored in an array. HashMap class defines this variable as:

|  |
| --- |
| entry array |
| /\*\*   \* The table, resized as necessary. Length MUST Always be a power of two.   \*/  transient Entry[] table; |

Now look at code implementation of put() method:

|  |
| --- |
| put() method |
| /\*\*  \* Associates the specified value with the specified key in this map. If the  \* map previously contained a mapping for the key, the old value is  \* replaced.  \*  \* @param key  \*            key with which the specified value is to be associated  \* @param value  \*            value to be associated with the specified key  \* @return the previous value associated with <tt>key</tt>, or <tt>null</tt>  \*         if there was no mapping for <tt>key</tt>. (A <tt>null</tt> return  \*         can also indicate that the map previously associated  \*         <tt>null</tt> with <tt>key</tt>.)  \*/  public V put(K key, V value) {      if (key == null)      return putForNullKey(value);      int hash = hash(key.hashCode());      int i = indexFor(hash, table.length);      for (Entry<K , V> e = table[i]; e != null; e = e.next) {          Object k;          if (e.hash == hash && ((k = e.key) == key || key.equals(k))) {              V oldValue = e.value;              e.value = value;              e.recordAccess(this);              return oldValue;          }      }        modCount++;      addEntry(hash, key, value, i);      return null;  } |

**4.1. What put() method does**

Let’s note down the internal working of put method in hashmap.

1. First of all, the key object is checked for null. If the key is null, the value is stored in table[0]position. Because hashcode for null is always 0.
2. Then on next step, a hash value is calculated using the key’s hash code by calling its hashCode()method. This hash value is used to calculate the index in the array for storing Entry object. JDK designers well assumed that there might be some poorly written hashCode() functions that can return very high or low hash code value. To solve this issue, they introduced another hash()function and passed the object’s hash code to this hash() function to bring hash value in the range of array index size.
3. Now indexFor(hash, table.length) function is called to calculate exact index position for storing the Entry object.

**4.2. How collisions are resolved**

Here comes the main part. Now, as we know that two unequal objects can have the same hash code value, how two different objects will be stored in same array location called **bucket**.

The answer is LinkedList. If you remember, Entry class had an attribute "next". This attribute always points to the next object in the chain. This is exactly the behavior of LinkedList.

1. So, in case of collision, Entry objects are stored in linked list form. When an Entry object needs to be stored in particular index, HashMap checks whether there is already an entry?? If there is no entry already present, the entry object is stored in this location.

If there is already an object sitting on calculated index, its next attribute is checked. If it is null, and current entry object becomes next node in linkedlist. If next variable is not null, procedure is followed until next is evaluated as null.

1. What if we add the another value object with same key as entered before. Logically, it should replace the old value. How it is done? Well, after determining the index position of Entry object, while iterating over linkedist on calculated index, HashMap calls equals method on key object for each entry object.

All these entry objects in linkedlist will have similar hashcode but equals() method will test for true equality. If key.equals(k) will be true then both keys are treated as same key object. This will cause the replacing of value object inside entry object only.

**In this way, HashMap ensure the uniqueness of keys**.

**5. How HashMap.get() methods works internally?**

Now we know how key-value pairs are stored in HashMap. Next big question is : what happens when an object is passed in get() method of HashMap? How the value object is determined?

Answer we already should know that the way key uniqueness is determined in put() method , same logic is applied in get() method also. The moment hashmap identify the exact match for the key object passed as an argument, it simply returns the value object stored in current Entry object.

If no match is found, get() method returns null.

|  |
| --- |
| get() method |
| /\*\*  \* Returns the value to which the specified key is mapped, or {@code null}  \* if this map contains no mapping for the key.  \*  \* <p>  \* More formally, if this map contains a mapping from a key {@code k} to a  \* value {@code v} such that {@code (key==null ? k==null :  \* key.equals(k))}, then this method returns {@code v}; otherwise it returns  \* {@code null}. (There can be at most one such mapping.)  \*  \* </p><p>  \* A return value of {@code null} does not <i>necessarily</i> indicate that  \* the map contains no mapping for the key; it's also possible that the map  \* explicitly maps the key to {@code null}. The {@link #containsKey  \* containsKey} operation may be used to distinguish these two cases.  \*  \* @see #put(Object, Object)  \*/  public V get(Object key) {      if (key == null)      return getForNullKey();      int hash = hash(key.hashCode());      for (Entry<K , V> e = table[indexFor(hash, table.length)]; e != null; e = e.next) {          Object k;          if (e.hash == hash && ((k = e.key) == key || key.equals(k)))              return e.value;      }      return null;  } |

Above code is same as put() method till if (e.hash == hash && ((k = e.key) == key || key.equals(k))), after this simply value object is returned.

**6. Key notes on internal working of HashMap**

1. Data structure to store entry objects is an array named table of type Entry.
2. A particular index location in array is referred as bucket, because it can hold the first element of a linkedlist of entry objects.
3. Key object’s hashCode() is required to calculate the index location of Entry object.
4. Key object’s equals() method is used to maintain uniqueness of keys in map.
5. Value object’s hashCode() and equals() method are not used in HashMap’s get() and put()methods.
6. Hash code for null keys is always zero, and such entry object is always stored in zero index in Entry[].