**HashMap Internal Working:**

\*HashMap works on the principle of hashing principle

\*Hashing is the mechanasim of assigining the unique code to a variable or an attribute using some algoritham enable to retrieval.

\*A true hashing mechanism should always return same hashcode.when it is applied on the same object.

\*Because the HashMap stores the Objects as **Entry instances,**not as**key and value**

static class Entry<K,V> implements Map.Entry<K,V>

{

final K key;

V value;

Entry<K,V> next;

final int hash;

........

}

you know that the hashmap stores **the Entry instances** in an array and not as **a key,value** pairs.

**How to Prevent Duplicates:**

public class HashMapEg

{

public static void main(String[] args)

{

Map map = new HashMap();

map.put(1,"sam");

map.put(1,"Ian");

map.put(1,"Scott");

map.put(null,"asdf");

System.out.println(map);

}

}

**output as {null=asdf, 1=Scott}**

All the Entry Objects in the LinkedList will have the same hashcode but hashmap uses  equals () method checks the equality **if key.equals(k)** is true then it will replace the value object inside the Entry class and not the key. So this way it prevents the duplicate key being inserted.

**How Get() works:**

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;

}

**how**[**put() method**](https://www.javainterviewpoint.com/java-hashmap-putk-key-v-value-example/)**works internally ?**

The Code implementation of the put method will be like below

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;

}

First, it checks for the if the key given is null or not, if the given key is null it will be stored in the ‘0’th position as the hashcode of null will be zero.

Then it applies the hashcode to the key.hashCode() by calling the hashcode method. In order to get the value within the limits of an array the hash(key.hashCode())is called which does some shifting operations to the hashcode.

The indexFor() method is used to get the exact location to store the Entry object.

Then comes the most important part what happens if two different object has the same hashcode( eg : Aa,BB will have the same hashcode) and will it be stored in the same bucket. To handle this let’s think of the [LinkedList](https://www.javainterviewpoint.com/category/linkedlist/) in data structure it will have a next attribute which will always point to the next object . The same way the next attribute in the Entry class points to the next object. Using this different objects with the same hashcode will be placed next to each other.

In the case of the **Collision** , the HashMap checks for the value of the next attribute if it is null it inserts the Entry object in that location , if next attribute is not null then it keeps the loop running till next attribute is null then stores the Entry object there.

**Collision:**

A collision occurs when a hash function returns same bucket location for two different keys.  
Since all hash based Map class e.g. HashMap uses [equals() and hashCode() contract](http://javarevisited.blogspot.com/2015/01/why-override-equals-hashcode-or-tostring-java.html) to find the bucket

**HASHSET:**

**HashSet** uses HashMap internally to store it’s objects. Whenever you create a HashSet object, one **HashMap** object associated with it is also created. This HashMap object is used to store the elements you enter in the HashSet. The elements you add into HashSet are stored as **keys** of this HashMap object. The value associated with those keys will be a **constant**.

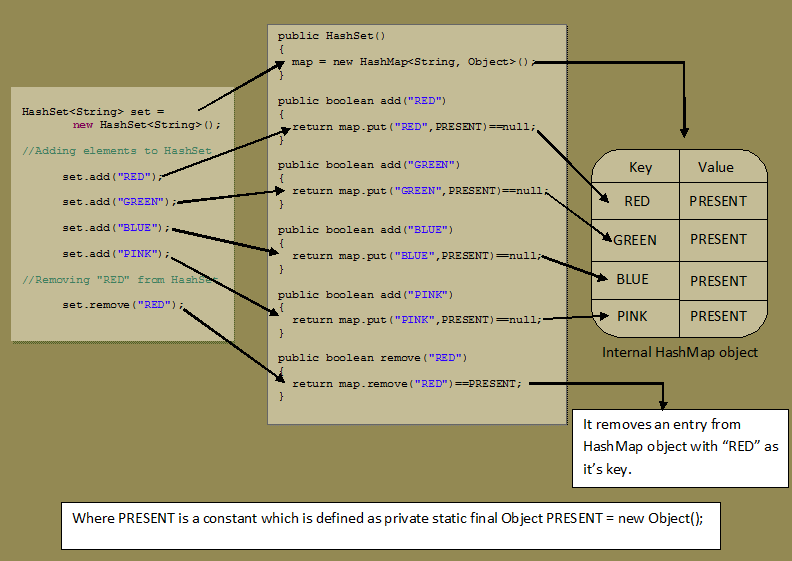
## How HashSet Works Internally In Java?

## Whenever you insert an element into HashSet using ****add()**** method, it actually creates an entry in the internally backing HashMap object with element you have specified as it’s key and constant called “****PRESENT****” as it’s value. This “PRESENT” is defined in the HashSet class as below.

|  |
| --- |
| private static final Object PRESENT = new Object(); |

Let’s have a look at add() method of HashSet class

|  |  |
| --- | --- |
| 1  2  3  4 | public boolean add(E e)  {        return map.put(e, PRESENT)==null;  } |



**Strings that are "" don't contain any value but are still instantiated are empty, Strings that are not instantiated are null**