***** How HashMap<K,V> works internally ?
----

a) While working with HashSet or HashMap every object must be compared because duplicate objects are not allowed.

b) Whenever we add any new key to HashMap object to verify whether key is unique or duplicate, HashMap internally uses hashCode(), == operator and equals method.

c) While adding the key object in the HashMap, first of all it will invoke the hashCode() method to retrieve the corresponding key hashcode value.
   Example :- hm.put(key,value);
            then internally key.hashCode();

d) If the newly added key and existing key hashCode value both are same (Hash collision), then only == operator is used for comparing those keys by using reference or memory address, if both keys references are same then existing key value will be replaced with new key value.

If the reference of both keys are different then only equals(Object obj) method is invoked to compare those keys by using content comparison. [If Overridden from Object class]

If the equals(Object obj) method returns true (content wise both keys are same), this new key is duplicate then existing key value will be replaced by new key value.

If equals(Object obj) method returns false, this new key is unique, new entry (key-value) will be inserted in the same Bucket by using Singly LinkedList

Note :- equals(Object obj) method is invoked only when two keys are having same hashcode as well as their references are different.

e) Actually by calling hashcode method we are not comparing the objects, we are just storing the objects in a group so the currently adding key object will be compared with its SAME HASHCODE GROUP objects, but not with all the keys which are available in the Map.

f) The main purpose of storing objects into the corresponding group to decrease the number of comparison so the efficiency of the program will be increased.

g) To insert an entry in the HashMap, HashMap internally uses Hashtable data structure.

h) Now, for storing same hashcode object into a single group, hash table data structure internally uses one more data structure called Bucket.

i) The Hashtable data structure internally uses Node class array object.[Node<K,V> []table]

j) The bucket data structure internally uses LinkedList data structure, It is a singly linked list again implemented by Node class only.

*k) A bucket is group of entries of same hash code keys.

l) Performance wise LinkedList is not good to serach, so from java 8 onwards LinkedList is changed to Binary tree to decrease the number of comparison within the same bucket hashcode if the number of entries are greater than 8.

** equals() and hashCode() method contract :

Both the methods are working together to find out the duplicate objects in the Map.

*If equals() method invoked on two objects and it returns true then hashcode of both the objects must be same.

Note : IF TWO OBJECTS ARE HAVING SAME HASH CODE THEN IT MAY BE SAME OR DIFFERENT OBJECT BUT IF EQUALS(OBJECT OBJ) METHOD RETURNS TRUE THEN BOTH OBJECTS MUST RETURN SAME HASHCODE.

```
package com.ravi.hashmap_internal;

import java.util.HashMap;

public class HashMapInternal
{
    public static void main(String[] args)
    {
          HashMap<String,Integer> hm1 = new HashMap<>();
         hm1.put("A", 1);
         hm1.put("A", 2);
         hm1.put(new String("A"), 3);
         System.out.println("Size is :"+hm1.size());
         System.out.println(hm1);

         System.out.println("...................");

         HashMap<Integer,Integer> hm2 = new HashMap<>();
         hm2.put(128, 1);
         hm2.put(128, 2);
         System.out.println("Size is :"+hm2.size());
         System.out.println(hm2);
         System.out.println("...................");

         HashMap<Object,Object> hm3 = new HashMap<>();
         hm3.put("A", 1);
         hm3.put("A", 2);
         hm3.put(new String("A"), 3);
         hm3.put(65, 4);
         System.out.println("Size is :"+hm3.size());
         System.out.println(hm3);
    }
}
```

What will happen if we don't follow the contract ?
Case 1 :

If we override only equals(Object obj)

If we override only equals(Object obj) method for content comparison
then same object (duplicate object) will have different hashcode (due to Object class hashCode())
hence same object (content wise) will move into two different buckets [Duplication].

Case 2 :

If we override only hashCode() method

If we overrdie only hashCode() method then two objects which are having same hashcode (due to overriding) will go to same bucket but == operator and equals(Object obj) method of Object class, both will return false hence duplicate object will be inserted into same bucket by using Singly LinkedList.

So, the conclusion is, compulsory we need to override both the methods for removing duplicate elements.

```
package com.ravi.hashmap_internal;

import java.util.HashMap;
import java.util.Objects;

class Customer
{
    private Integer customerId;
    private String customerName;


    public Customer(Integer customerId, String customerName)
    {
        super();
        this.customerId = customerId;
        this.customerName = customerName;
    }

    @Override
    public String toString()
    {
        return "Customer [customerId=" + customerId + ", customerName=" + customerName +
"]";
    }

    @Override
    public int hashCode()
    {
        return Objects.hash(customerId, customerName);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Customer other = (Customer) obj;
        return Objects.equals(customerId, other.customerId) && Objects.equals(customerName,
other.customerName);
    }


}

public class HashMapInternalDemo1
{
    public static void main(String[] args)
    {
       Customer c1 = new Customer(111, "Scott");
       Customer c2 = new Customer(111, "Scott");

       System.out.println(c1.hashCode()+" : "+c2.hashCode());
       System.out.println(c1.equals(c2));


       System.out.println("............................");

       HashMap<Customer,String> map = new HashMap<>();
       map.put(c1, "A");
       map.put(c2, "B");

       System.out.println(map.size());
       System.out.println(map);

    }

}
```

Note :

Customer class we are using as a HashMap key so it must override
hashCode() and equals(Object obj) as well as at advanced level, It must be immutable class.

All the Wrapper classes and String class are immutable as well as
hashCode() and equals(Object obj) methods are overridden in these classes so perfectly suitable to becoming HashMap key.

so final conclusion is, In our user-defined class which we want to use as a HashMap key must be immutable and hashCode() and equals(Object obj) method must be overridden.

Instead of BLC class we can also use simply record because record is implicitly final (immutable) and hashCode() and equals(Object obj) methods are overridden.

```
package com.ravi.hashmap_internal;

import java.util.HashMap;

//perfect for HashMap key
record Customer(Integer id, String name)
{
}

public class HashMapInternalDemo1
{
    public static void main(String[] args)
    {
       Customer c1 = new Customer(111, "Scott");
       Customer c2 = new Customer(111, "Scott");

       System.out.println(c1.hashCode()+" : "+c2.hashCode());
       System.out.println(c1.equals(c2));


       System.out.println("............................");

       HashMap<Customer,String> map = new HashMap<>();
       map.put(c1, "A");
       map.put(c2, "B");

       System.out.println(map.size());
       System.out.println(map);

    }
}
```