https://java2blog.com/how-hashmap-works-in-java/
https://java2blog.com/hashcode-and-equals-method-in-java/

Most common interview questions are How HashMap works in java, “How get and put method of HashMap work internally”. Here I am trying to explain internal functionality with an easy example. `HashMap` is one of the most used Collections in java.Rather than going through theory, we will start with example first, so that you will get better understanding and then we will see how get() and put() function work in java.

Let’s take a very simple example. I have a `Country` class, we are going to use Country class object as key and its `capitalname` (string) as value. Below example will help you to understand, how these key value pair will be stored in hashmap.

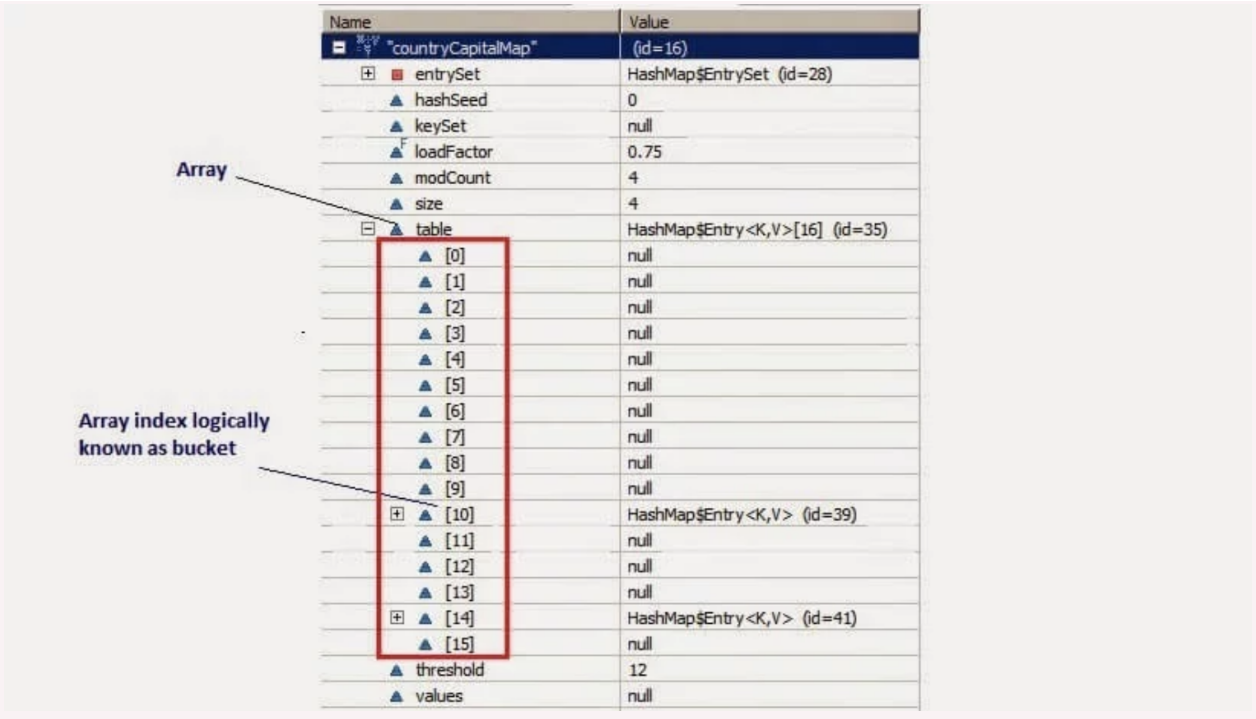
1. Country.java

```java
package org.arpit.java2blog;
public class Country {

    String name;
    long population;

    public Country(String name, long population) {
        super();
        this.name = name;
        this.population = population;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public long getPopulation() {
        return population;
    }
    public void setPopulation(long population) {
        this.population = population;
    }

    // If length of name in country object is even then return 31(any random number) and if odd the
    // This is not a good practice to generate hashcode as below method but I am doing so to give b
    @Override
    public int hashCode() {
        if(this.name.length()%2==0)
            return 31;
        else
            return 95;
    }
    @Override
    public boolean equals(Object obj) {

        Country other = (Country) obj;
        if (name.equalsIgnoreCase((other.name)))
            return true;
        return false;
    }

}
```


2. HashMapStructure.java

```java
import java.util.HashMap;
import java.util.Iterator;

public class HashMapStructure {

    /**
     * @author Arpit Mandliya
     */
    public static void main(String[] args) {

        Country india=new Country("India",1000);
        Country japan=new Country("Japan",10000);

        Country france=new Country("France",2000);
        Country russia=new Country("Russia",20000);

        HashMap<Country, String> countryCapitalMap=new HashMap<Country,String>();
        countryCapitalMap.put(india,"Delhi");
        countryCapitalMap.put(japan,"Tokyo");
        countryCapitalMap.put(france,"Paris");
        countryCapitalMap.put(russia,"Moscow");

        Iterator countryCapitalIter=countryCapitalMap.keySet().iterator();//put debug point at this
        while(countryCapitalIter.hasNext())
        {
            Country countryObj=countryCapitalIter.next();
            String capital=countryCapitalMap.get(countryObj);
            System.out.println(countryObj.getName()+"----"+capital);
        }
    }

}
```

Now put debug point at line 24 and right click on project->debug as-> java application. Program will stop execution at line 24 then right click on `countryCapitalMap` then select watch.You will be able to see structure as below.



Now From above diagram, you can observe the following points

1. There is an `Entry[]` array called table which has size 16.
2. This table stores Entry class's object. HashMap class has an inner class called `Entry` . This Entry have key value as an instance variable. Let's see structure of entry class Entry Structure.

```
static class Entry implements Map.Entry
{
        final K key;
        V value;
        Entry next;
        final int hash;
        ...//More code goes here
}
```

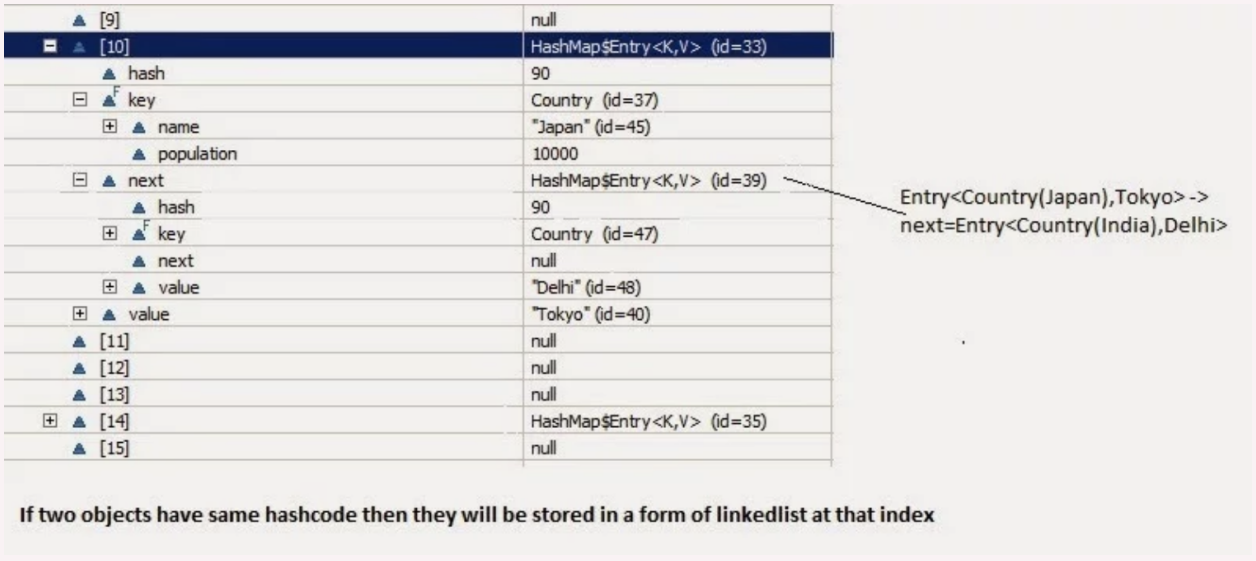
1. Whenever we try to put any key value pair in hashmap, Entry class object is instantiated for key value and that object will be stored in above mentioned `Entry[]` (table). Now you must be wondering, where will above created Entry object get stored(exact position in table). The answer is, hash code is calculated for a key by calling Hascode() method. This hashcode is used to calculate index for above Entry[] table.
2. Now, If you see at array index 10 in above diagram, It has an Entry object named `HashMap$Entry` .
3. We have put 4 key-values in Hashmap but it seems to have only 2!!!! This is because if two objects have same hashcode, they will be stored at same index. Now the question arises how? It stores objects in the form of `LinkedList` (logically).

So how hashcode of above country key-value pairs are calculated.

```
Hashcode for Japan = 95 as its length is odd.
Hashcode for India =95 as its length is odd
```

```
HashCode for Russia=31 as its length is even.
HashCode for France=31 as its length is even.
```

Below diagram will explain LinkedList concept clearly.



So now if you have good understanding of `Hashmap` structure,Lets go through put and get method.

Put

Let's see the implementation of 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 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;
}
```

now lets understand above code step by step

1. Key object is checked for null. If key is null then it will be stored at `table[0]` because hashcode for null is always 0.
2. Key object's hashcode() method is called and hash code is calculated. This hashcode is used to find index of array for storing Entry object. It may happen sometimes that, this hashcode function is poorly written so JDK designer has put another function called hash() which takes above-calculated hash value as argument. If you want to learn more

about hash() function, you can refer hash and indexFor method in hashmap.

3. `indexFor(hash,table.length)` is used to calculate exact index in table array for storing the Entry object.
4. As we have seen in our example, if two key objects have same hashcode(which is known as **collision**) then it will be stored in form of linkedlist.So here, we will iterate through our linkedlist.
    - If there is no element present at that index which we have just calculated then it will directly put our Entry object at that index.
    - If There is element present at that index then it will iterate until it gets `Entry->next` as null.
    - What if we are putting same key again, logically it should replace old value. Yes, it will do that. While iterating it will check key equality by calling `equals()` method(**key.equals(k)**), if this method returns true then it replaces value object with current Entry's value object.
    - If it did not find the duplicate key, then current `Entry` object will become first node in linkedlist and current Entry -> next will become an existing first node on that index.

Get

Lets see implementation of get now:

```
/**
     * Returns the value to which the specified key is mapped, or {@code null}
     * if this map contains no mapping for the key.
     *
     *
     * 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.)
     *
     *
     * 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 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;
    }
```

As you got the understanding on put functionality of hashmap. So to understand get functionality is quite simple. If you pass any key to get value object from hashmap.

1. Key object is checked for null. If key is null then value of Object resides at `table[0]` will be returned.

2. Key object's hashcode() method is called and hash code is calculated.
3. indexFor(hash,table.length) is used to calculate exact index in table array using generated hashcode for getting the Entry object.
4. After getting index in table array, it will iterate through linkedlist and check for key equality by calling equals() method and if it returns true then it returns the value of Entry object else returns null.

Key points to Remeber

- HashMap has a inner class called `Entry` which stores key-value pairs.
- Above Entry object is stored in Entry[ ](Array) called table
- An index of table is logically known as bucket and it stores first element of `LinkedList` .
- Key object's hashcode() is used to find bucket of that `Entry` object.
- If two key object 's have same `hashcode` , they will go in same bucket of table array.
- Key object 's `equals()` method is used to ensure uniqueness of key object.
- Value object 's equals() and hashcode() method is not used at all

---

hashcode() and equals() method in java

In this post ,we will try to understand hashcode() and equals() method in java.

These methods can be found in the Object class and hence available to all java classes.Using these two methods, an object can be stored or retrieved from a Hashtable, HashMap or HashSet.

- hashcode()
- equals()

hashcode():

You might know if you put entry in HashMap, first hashcode is calculated and this hashcode used to find bucket(index) where this entry will get stored in hashMap.You can read more at How hashMap works in java. What if you don't override hashcode method, it will return integer representation of memory address.

equals():

You have to override equals method, when you want to define equality between two object. If you don’t override this method, it will check for reference equality(==) i.e. if tow reference refers to same object or not

Lets override default implemenation of hashcode() and equals():

You don’t have to always override these methods, but lets say you want to define equality of country object based on name, then you need to override equals method and if you are overriding equals method, you should override hashcode method too. Below example will make it clear.

Lets see with the help of example.We have a class called Country

1. Country.java

```java
package org.arpit.java2blog;

public class Country {

    String name;
    long population;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public long getPopulation() {
        return population;
    }
    public void setPopulation(long population) {
        this.population = population;
    }

}
```

This country class have two basic attributes- name and population.

Now create a class called “EqualityCheckMain.java”

```java
package org.arpit.java2blog;

public class EqualityCheckMain {

    /**
     * @author arpit mandliya
     */
    public static void main(String[] args) {

        Country india1=new Country();
        india1.setName("India");
        Country india2=new Country();
        india2.setName("India");
        System.out.println("Is india1 is equal to india2:" +india1.equals(india2));
    }

}
```

When you run above program, you will get following output:-

```
Is india1 is equal to india2:false
```

In the above program, we have created two different objects and set their name attribute to “india”.

Because both references india1 and india2 are pointing to different object, as default implementation of equals check for ==,equals method is returning false. In real life, it should have return true because no two countries can have same name.

Now let's override equals and return true if two country’s name are same.

Add this method to above country class:

```java
@Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Country other = (Country) obj;
        if (name == null) {
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        return true;
    }
```

and now run EqualityCheckMain.java again

You will get following output:

```
Is india1 is equal to india2:true
```

Now this is because overriden equals method return true if two country have same name.

One thing to remember here, signature of equals method should be same as above.

Lets put this Country objects in hashmap:

Here we are going to use Country class object as key and its capital name(string) as value in HashMap.

```
package org.arpit.java2blog;

import java.util.HashMap;
import java.util.Iterator;

public class HashMapEqualityCheckMain {

    /**
     * @author Arpit Mandliya
     */
    public static void main(String[] args) {
        HashMap<Country,String> countryCapitalMap=new HashMap<Country,String>();
        Country india1=new Country();
        india1.setName("India");
        Country india2=new Country();
        india2.setName("India");

        countryCapitalMap.put(india1, "Delhi");
        countryCapitalMap.put(india2, "Delhi");

        Iterator countryCapitalIter=countryCapitalMap.keySet().iterator();
        while(countryCapitalIter.hasNext())
        {
            Country countryObj=countryCapitalIter.next();
            String capital=countryCapitalMap.get(countryObj);
            System.out.println("Capital of "+ countryObj.getName()+"----"+capital);

        }
    }
}
```

When you run above program, you will see following output:

```
Capital of India----Delhi
Capital of India----Delhi
```

Now you must be wondering even through two objects are equal why HashMap contains two key value pair instead of one.This is because First HashMap uses hashcode to find bucket for that key object, if

hashcodes are same then only it checks for equals method and because hashcode for above two country objects uses default hashcode method,Both will have different memory address hence different hashcode.

Now lets override hashcode method.Add following method to Country class

```
@Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        return result;
    }
```

Now run HashMapEqualityCheckMain.java again

You will see following output:

```
Capital of India----Delhi
```

So now hashcode for above two objects india1 and india2 are same, so Both will be point to same bucket,now equals method will be used to compare them which will return true.

This is the reason java doc says “if you override equals() method then you must override hashCode() method”

Key points to remember:

1. If you are overriding equals method then you should override hashcode() also.
2. If two objects are equal then they must have same hashcode.
3. If two objects have same hashcode then they may or may not be equal

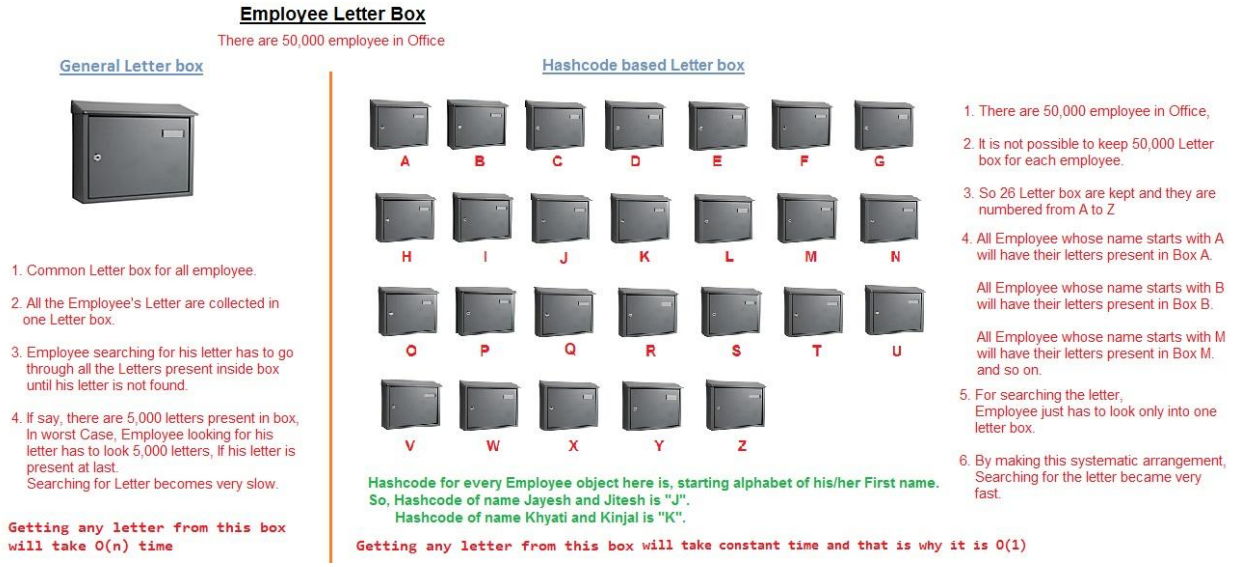
4. Always use same attributes to generate equals and hashcode as in our case we have used name.

Complexity:-

https://javabypatel.blogspot.com/2015/10/time-complexity-of-hashmap-get-and-put-operation.html

Hashmap works on principle of hashing and internally uses hashcode as a base, for storing key-value pair.

With the help of hashcode, Hashmap distribute the objects across the buckets in such a way that hashmap put the objects and retrieve it in constant time O(1).



From the above example, it is clear that, put operation in hashmap requires,

Step 1. Computing hashcode of key,

Step 2. Calculating array index/bucket from hashcode and then,

Step 3. With the help of index calculated, directly jump to that index/bucket.

Step 4. Now, each and every element in the bucket is scanned sequentially to see, is there any

key-value pair present, which has the same key we are trying to put.

If key-value pair is found, which has same key then instead of storing

new entry / key-value pair, it simply replace value stored against key.

If no matching key is found then it will go till end of the list and create a new key-value

pair at the end.

So, only costly operation what we see here is iterating the linked list after getting the bucket or array index.

If it has to iterate linked list, then how get and put operation is O(1)?

This question is perfectly valid. Hashmap put and get operation time complexity is O(1) with assumption that key-value pairs are well distributed across the buckets. It means hashcode implemented is good.

In above Letter Box example, If say hashcode() method is poorly implemented and returns hashcode 'E' always, In this case.

1. hashcode computed of Employee "Daniel" will be 'E', So all Letter of Daniel will end up in

    Letter box marked "E".

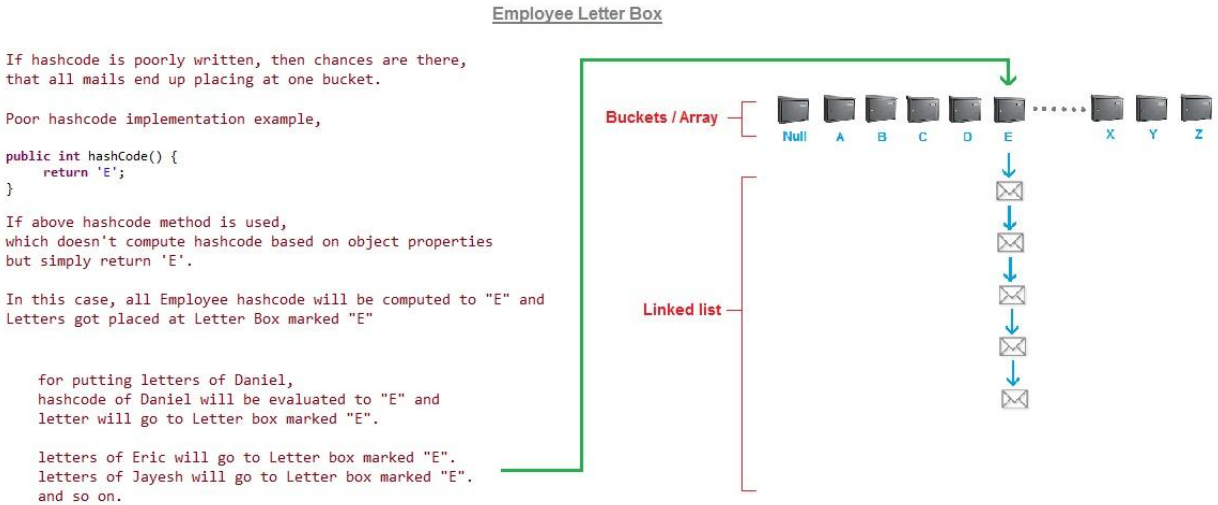
2. hashcode computed of Employee "Jayesh" will be 'E', So all Letter of Jayesh will end up in

Letter box marked "E".

3. hashcode computed of Employee "Eric" will be 'E', So all Letter of Eric will end up putting in

Letter box marked "E".

In this case how, hashmap will look like,

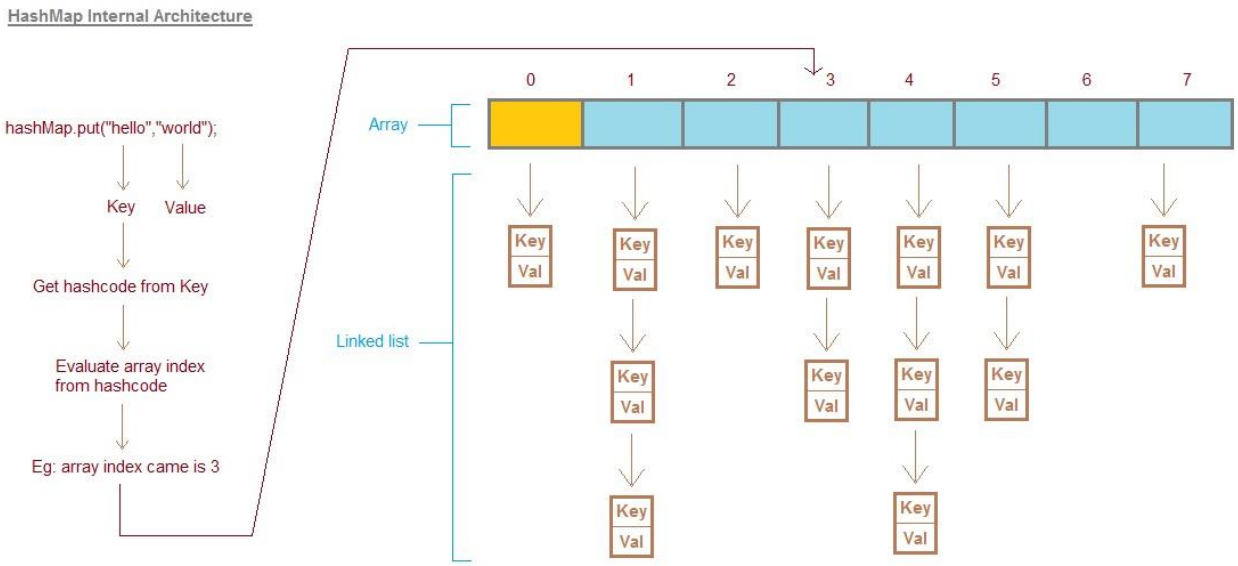


Imagine the time it will take to search a Letter of Daniel, Eric, Jayesh or any Employee. Since all letters are placed in one bucket, Put and Get operation will no longer have time complexity of O(1) because put and get operation has to scan each letter inside the bucket for matching key.

In above case, get and put operation both will have time complexity O(n).

Hashmap best and average case for Search, Insert and Delete is O(1) and worst case is O(n).

Hashcode is basically used to distribute the objects systematically, so that searching can be done faster.



We can say that item lookup inside bucket take expected O(1) time only under the assumptions that good hashcode() function is implemented,

Above line means to say that, If hashcode() function is written good then, hashcode generated will distribute the items across all the buckets and doesn't end up putting all item in one bucket.

Say if we have 5 items and 5 bucket then good hashcode method will distribute each item in each bucket, this gives best complexity of O(1) as each bucket have only one item.

So look up required is only once.

Now, if we have 10 items and 5 bucket then good hashcode method will distribute 2 items in each bucket, In this case, number of items is doubled, still for searching any element it will require only 2 look up.

Now, if we have 20 items and 5 bucket then good hashcode method will distribute 4 items in each bucket, In this case, number of items is doubled, still for searching any element it will require only 4 look up. So far, so good.

Now, if we have 40 items and 5 bucket then good hashcode method will distribute 8 items in each bucket, In this case, number of items is doubled, still for searching any element it will require only 8 look up.

Now, performance is little bit degraded.

Now, if we have 80 items and 5 bucket then good hashcode method will distribute 16 items in each bucket, In this case, number of items is doubled, still for searching any element it will require only 16 look up.

Now, performance is little bit degraded.

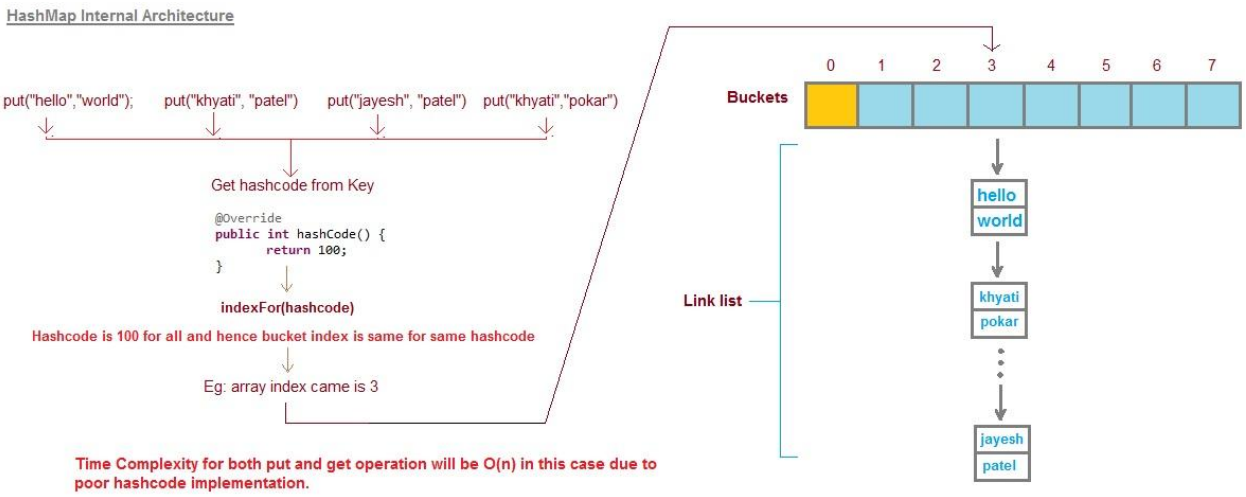
But, if you observe, as the number of items are doubled, elements that need to be searched within bucket, that is the number of look ups are not increasing very high and remain almost constant compared to number of items increased.

That is why it is called that hashmap's get and put operation takes O(1) time.

To be very precise, The amortized/average case performance of Hashmap is said to be O(1) for put and get operation.

Remember, hashmap's get and put operation takes O(1) time only in case of good hashcode implementation which distributes items across buckets.

In case of poor hashcode implementation, chances are there, that all elements get placed in one bucket and hashmap look like below,



In above case, where all key-value pair are placed in one bucket, In worst case, the time it will take for both put and get operation will be O(n) where n = number of key-value pair present.

Put operation has to look into each key-value pair in bucket to see matching key is present,

If present then it needs to replace the value where key is matched.

If not present, then insert new key-value pair at end of link list.

This make put operation in worst case as O(n).

Get operation has to do linear search in bucket for Key look up, since all key-value pair are placed in one bucket, and hence complexity of get operation in worst case will be O(n).

This is why it's important to design good hash functions.