# CORE JAVA - Day 60

## Agenda

- **Hashing Algorithm**
- **LinkedHashMap**

HashMap stores the data in an unordered collections. Now let us understand why HashMap stores the data in an unordered collection with an example.

**Example:**

**MapIntro.java**

```
import java.util.HashMap;

public class MapIntro{
   public static void main(String[] args){
       HashMap<String, Integer> names = new HashMap<String, Integer>();
       names.put("anna", 27);
       names.put("bob", 81);
       names.put("elle", 27);
       names.put("otto", 54);
       names.put("arora", 18);
       names.put("ivi", 45);
       names.put("jj", 72);

       System.out.println(names);
   }
}
```

**Output:**

```
$java MapIntro.java
{jj=72, arora=18, bob=81, otto=54, anna=27, ivi=45, elle=27}
```

As you can see from the above example, the output is not in the same order which is inserted into the HashMap and it is not inserting in some random order it is following **Hashing Algorithm** to insert the value inside the HashMap and **Hashing Algorithm is applied only to the keys not to the values**. Now let's understand the working of Hashing Algorithm using the example.

Whenever you create an object of HashMap, internally HashMap gets created which has an **initial capacity =16** and it will be stored with a value null initially. Once initial capacity is 75% filled then capacity of HashMap doubles every time.

| | n=16 |
|---|---|
| 0 | null |
| 1 | null |
| 2 | null |
| 3 | null |
| 4 | null |
| 5 | null |
| 6 | null |
| 7 | null |
| 8 | null |
| 9 | null |
| 10 | null |
| 11 | null |
| 12 | null |
| 13 | null |
| 14 | null |
| 15 | null |

Keys to be inserted inside the HashMap are:

**"anna", "bob", "elle", "otto", "arora", "ivi", "jj"**

## Hashing Algorithm:

```
hc = hashcode(key)

hash = hc^(hc>>>16)

index =hash & (n-1)
```

Whenever you try to put the elements inside the HashMap that time Hashing Algorithm is applied.

- When the first key-value pair is inserted inside the HashMap i.e "anna" and 27, hashing algorithm is applied step by step as shown below.

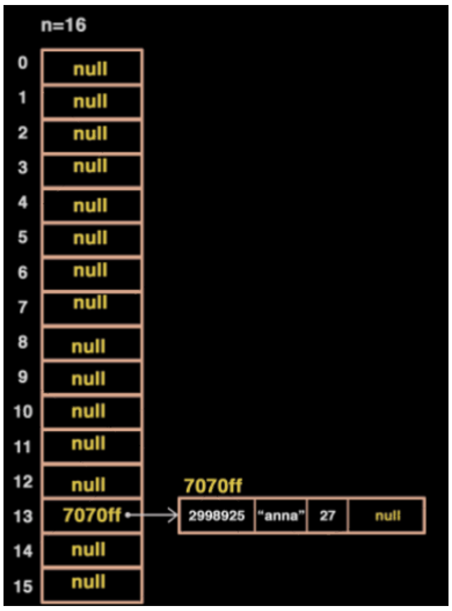
## Hashing Algorithm

hc = hashcode(key) = hashcode("anna") = 2998944

hash =hc^(hc>>>16) =2998944^(2998944>>>16) = 2998925

index = hash & (n-1) = 2998925 & (16-1) = 13

First hashcode of "anna" is calculated**, hashcode is present inside the object class and object class is base class for all the classes thus it is inherited in all the class.** In this example, we are trying to insert string value (inside the string class also hashcode method has been inherited and it has been overridden), here it uses formula to calculate the hascode of given string. Formula to calculate hashcode is
s[0]*31^(n-1)+s[1]*31^(n-2)+……+s[n-1]
Once hashcode of "anna" is calculated then using the value of hc and calculate the value of hash. Next by substituting the value of hash and n, index value is calculated. After calculating hashing algorithm, index = 13, so in 13th index key-value pair is stored. Now we will see how it will be stored inside the hashMap.
First it will create separate new memory inside which key, value, hash and null will be stored (moving further we will understand why null is inserted over there) the address of that will be stored inside the index 13, now it is referring to that address as shown below.



- Next "bob" and 81 is put into the hashMap and hashing Algorithm is calculated as shown below.

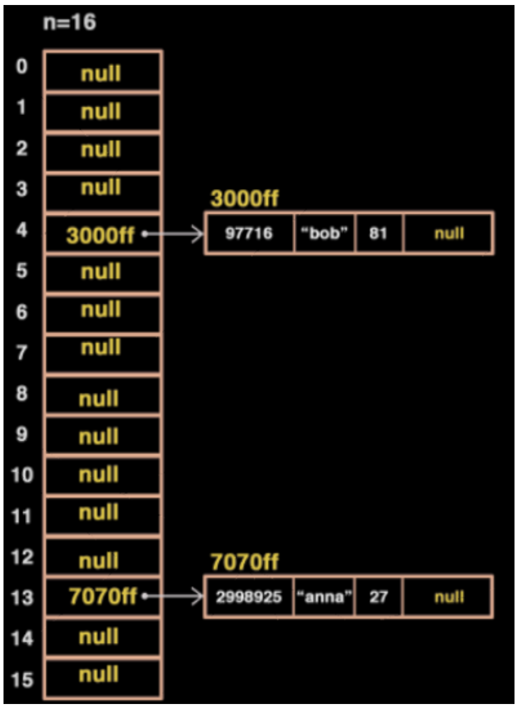
## Hashing Algorithm

hc = hashcode(key) = hashcode("bob") = 97717

hash =hc^(hc>>>16) =97717^(97717>>>16) = 97716

index = hash & (n-1) = 97716 & (16-1) = 4

Now, in index 4 "bob" and its value is inserted as shown below.



- Next “elle” and 27 is put into the hashMap and hashing Algorithm is calculated as shown below.

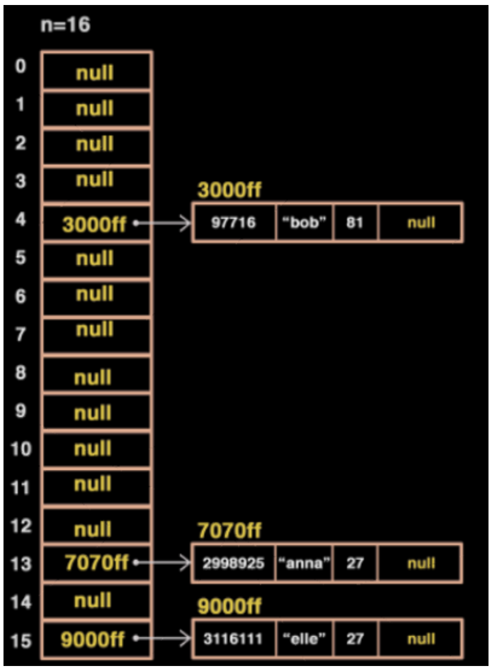
**Hashing Algorithm**

hc = hashcode(key) = hashcode(“elle”) = 3116128

hash =hc^(hc>>>16) =3116128^(3116128>>>16) = 3116111

index = hash & (n-1) = 3116128 & (16-1) = 15

Now, in index 15 “bob” and its value is inserted as shown below.



- Next “otto” and 27 is put into the hashMap and hashing Algorithm is calculated as shown below.

**Hashing Algorithm**

hc = hashcode(key) = hashcode(“otto”) = 34321984

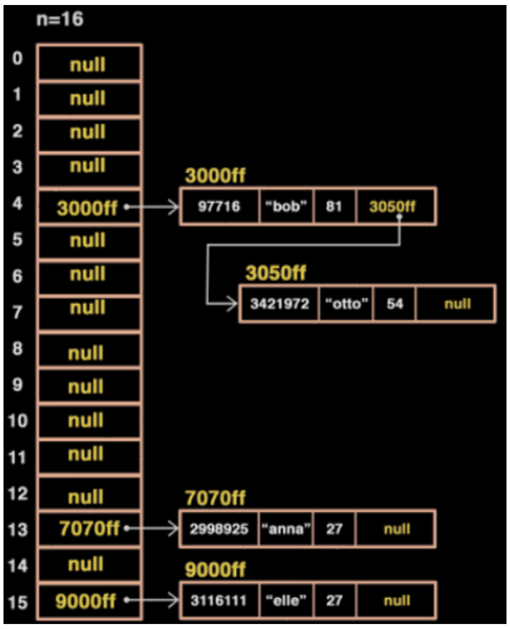
hash =hc^(hc>>>16) =34321984^(34321984>>>16) = 34321972

index = hash & (n-1) = 34321972 & (16-1) = 4

Now if you observe you got index value as 4, but in index 4 already “bob” and 81 is present then where will you store the “otto” and 27. This is the problem which will occur in hashing, the problem is called as **collision**. To overcome this collision there is a collision resolving techniques shown below.

1. Separate chaining(Open Hashing)
2. Open Addressing(closed Hashing)
   a. Linear probing
   b. Quadratic probing
   c. Double Hashing

Among all these collision resolving techniques java makes use of **separate chaining.** Separate chaining internally makes use of Linked list to store the values. First "otto" and 27 is stored in one memory location that address is taken and stored inside the memory of "bob" and 81 where null is present i.e null value present inside "bob" and 81 is now replaced with address of "otto" as shown below.



- Next "arora" and 18 is put into the hashMap and hashing Algorithm is calculated as shown below.

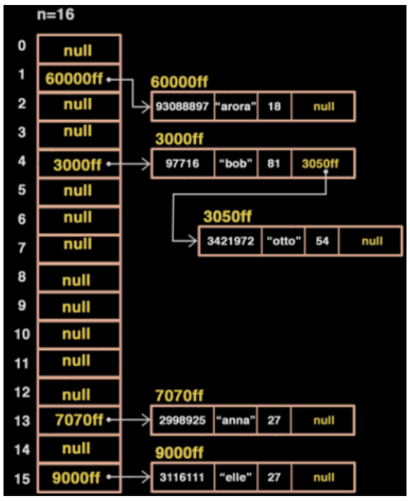
## Hashing Algorithm

hc = hashcode(key) = hashcode("arora") = 93088013

hash =hc^(hc>>>16) =93088013^(93088013>>>16) = 93088897

index = hash & (n-1) = 93088897 & (16-1) = 1

Now, in index 1 "arora" and its value is inserted as shown below.



- Next "ivi" and 45 is put into the hashMap and hashing Algorithm is calculated as shown below.

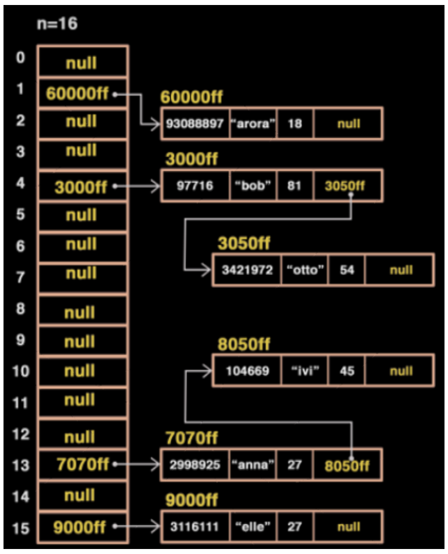
## Hashing Algorithm

hc = hashcode(key) = hashcode("ivi") = 104668

hash =hc^(hc>>>16) =104668^(104668>>>16) = 104669

index = hash & (n-1) = 104669 & (16-1) = 13

Now, in index 13 "ivi" and its value is inserted by making use of linked list because in index 13 already "anna" is present as shown below.



- Next “jj” and 72 is put into the hashMap and hashing Algorithm is calculated as shown below.

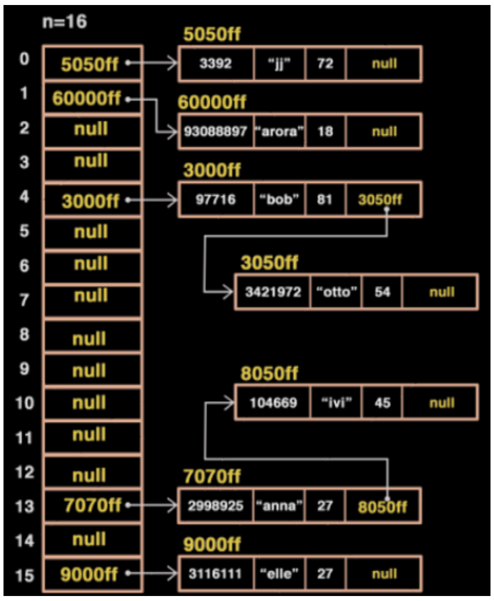
## Hashing Algorithm

hc = hashcode(key) = hashcode(“jj”) = 3392

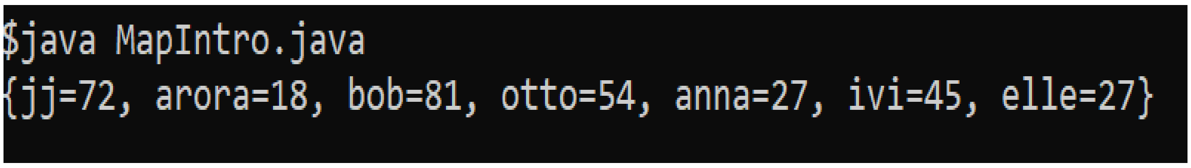
hash =hc^(hc>>>16) =3392^(3392>>>16) = 3392

index = hash & (n-1) = 3392 & (16-1) = 0

- Now, in index 0 “ivi” and its value is inserted as shown below



**After inserting all the elements inside the hashmap output looks as shown below.**

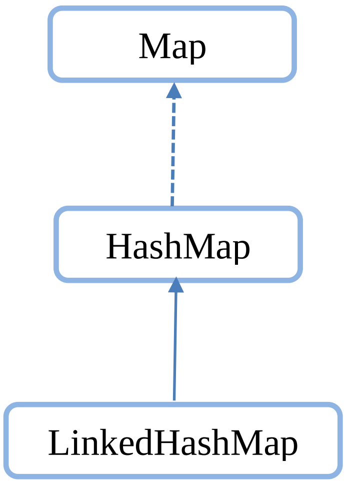


```
$java MapIntro.java
{jj=72, arora=18, bob=81, otto=54, anna=27, ivi=45, elle=27}
```

Now if you observe the value stored internally, the order in which values are stored internally in the same order we are getting the output not in some random order. The advantages of using hashing algorithm is it will be easy to fetch the value present inside the hash map i.e whenever you call get(key) it internally applies hashing algorithm such that it will directly go to that particular index and return the value present inside the hashmap.

Now you understood that hashmap doesn't store in the order of insertion but if the user wants the element to be stored in the **insertion order** then you have to make use of **LinkedHashMap** instead of HashMap.

LinkedHashMap inherits HashMap and HashMap implements Map as shown below.



## Example:

## MapIntro.java

```
import java.util.LinkedHashMap;

public class MapIntro{
   public static void main(String[] args){
       LinkedHashMap<String, Integer> names = new LinkedHashMap<String, Integer>();
       names.put("anna", 27);
       names.put("bob", 81);
       names.put("elle", 27);
       names.put("otto", 54);
       names.put("arora", 18);
       names.put("ivi", 45);
       names.put("jj", 72);

       System.out.println(names);
   }
}
```

## Output:

```
$java MapIntro
{anna=27, bob=81, elle=27, otto=54, arora=18, ivi=45, jj=72}
```

As you can see from the output **LinkedHashMap follows insertion order**.

**Example-2: To get number of key-value pair present inside the HashMap**

**MapIntro.java**

```
import java.util.LinkedHashMap;

public class MapIntro{
   public static void main(String[] args){
       LinkedHashMap<String, Integer> names = new LinkedHashMap<String, Integer>();
       names.put("anna", 27);
       names.put("bob", 81);
       names.put("elle", 27);
       names.put("otto", 54);
       names.put("arora", 18);
       names.put("ivi", 45);
       names.put("jj", 72);

       System.out.println(names.size());
   }
}
```

**Output:**

```
$java MapIntro.java
7
```

**size()** method returns the number of key-value pair present inside the LinkedHashMap.

**There are many such methods which we have seen in HashMap which are also available in LinkedHashMap. Why not try by yourself?**