How does hashmap work Internally – Explained:

In this post, let us see how hashmap works internally – How the elements are added and retrieved from buckets.

Features of HashMap:

- Implementation of Map Interface – With Key and Value pairs
- No order of maps, also order of map changes over time
- Accepts one null key and multiple null values
- Keys should be unique and cannot have duplicates
- Unsynchronized i.e. Not Thread Safe
- Initial capacity is 16 and load factor is 0.75
- Synchronized externally using Collections.SynchronizedMap(hashmap)

What is Initial Capacity and Load factor?

Initial Capacity is number of buckets in hashtable, that i.e. by default HashMap provides 16 buckets when created, we can define number of buckets required when creating the HashMap.

Load Factor indicates when a certain capacity of elements are populated in HashMap, it size will increaase. Load factor decides how much of capacity should be filled before increasing the size of hashmap.

https://www.youtube.com/watch?v=nff4KWFtwW0&feature=youtu.be

HashMap Creation:

- public HashMap()
- public HashMap(int initialCapacity)
- public HashMap(int initialCapacity, float loadFactor)
- public HashMap(Map<? extends k, ? extends v> m) –> Cretes new HashMap with the specified Map.

To understand how hashmap works internally, let us consider we have an employee model class with a variable employee name,

Let us create employee object and put the employee object as key in HashMap and let us see how it gets inserted into HashMap.

How does put method work in HashMap

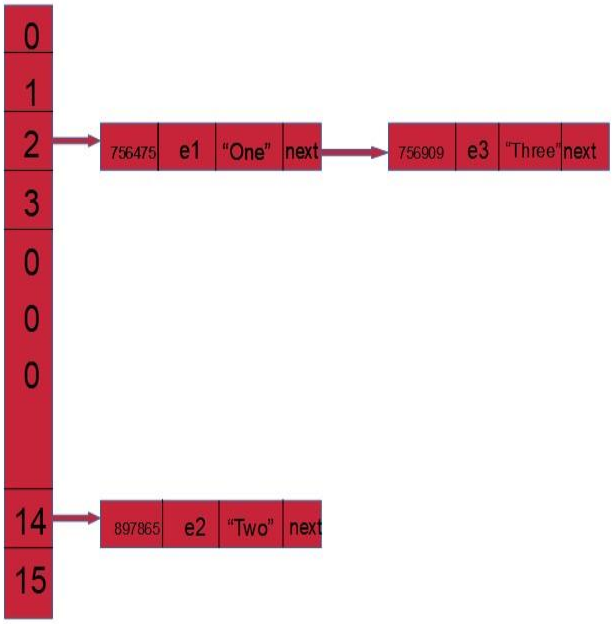
```
Employee e1 = new Employee("Alpha")
Employee e2 = new Employee("Beta")
Employee e3 = new Employee("Charlie")

HashMap<Employee, String> hm = new
HashMap<>()

hm.put(e1, "One");
hm.put(e2, "Two");
hm.put(e3, "Three");

Equals()

hashcode() for e1 – 756475 → 2
hashcode() for e2 – 897865 → 14
hashcode() for e3 – 756909 → 2
```



From the above,

We have created 3 employee objects – e1, e2 and e3 with names Alpha, Beta and Charlie

Then we have put these values in hashmap with employee object as Key

HashMap<Employee, String> hm = new HashMap<Employee, String>();

hm.put(e1, "One");
hm.put(e1, "Two");
hm.put(e1, "Three");

Now we have inserted the values into HashMap, we can see how they are populated in Buckets.

For our example, let us consider the hashcode values of

e1 – 756475 and it puts in bucket 2
e2 – 897865 and it puts in bucket 14
e3 – 756909 and it puts in bucket 2 again

First it checks the bucket, if any values are present. Since we do not have any values, it inserts the value in bucket 2.
It will insert hashcode first , i.e. 756475, next it will insert the key of HashMap – here the key is e1, then the value “One”, and last will be the pointer to next node. For now it will be null as no other elements are present to be populated in bucket.

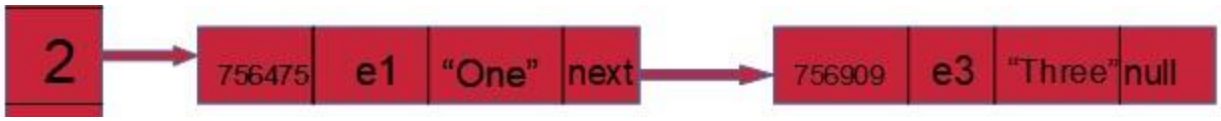
Second we have the hashmap with element key e2 and value “Two”. It will check the bucket 14 and since no value is already present, it will populate the values in bucket 14. hashcode of e2 – 897865, then the element Key, then element value, followed by the next value pointer, which is null for now.

Third, we have the hashmap with element key e3 and value “Three”. It will check for the bucket 2, and we have a value in it.

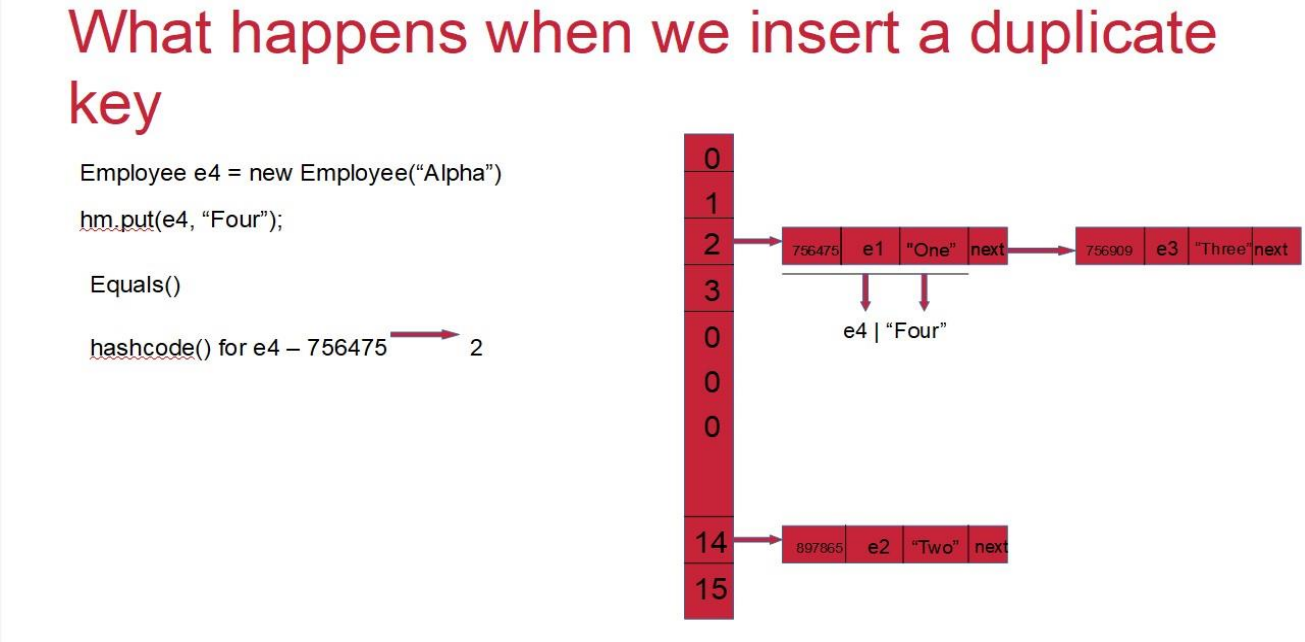
Here comes the key role for equals() method, it will check whether the e3 Key is unique when compared to e1. (e3.equals(e1)). It will return false as both objects are not equal. Then from the node which is kept as null when inserting first element, will point to next element i.e. e3.

In the same bucket as second entry, e3 will be populated with hashcode – 756909, Key – e3, value – Three and next will be kept as null until we get another value to be populated.

e1 entry which had null next now will point to e3 entry.



What happens when we insert duplicate Key?



Here we have created another employee object e4, with same value as of e1.

Employee e4 = new Employee("Alpha");

hm.put(e4, "Four");

How as we discussed, it will calculate the hashcode for the object e4. The hascode is 756475 and it points to bucket 2.

But the bucket 2 already has more records available. So first will check for hashcode, if the inserting record hashcode and available record hashcode matches, then it will compare the Key. This is where equals() plays an important role.

In the above case, e1.equals(e4) which will return true as both objects are equal. Then the values of object e1 will be replaced with values of e4.

so now the hashbucket first entry will be of e4 values,

Important Keypoints:

- HashMap uses hashcode() to identify the bucket where the record to be inserted
- If a record is already available in the bucket, it will check for key (to check uniqueness) with equals() method, if it return true then the record will be replaced. If it false, new record will be inserted.
- When more than one record is available in the bucket, it will have a next node which will point to the next available record in the bucket.

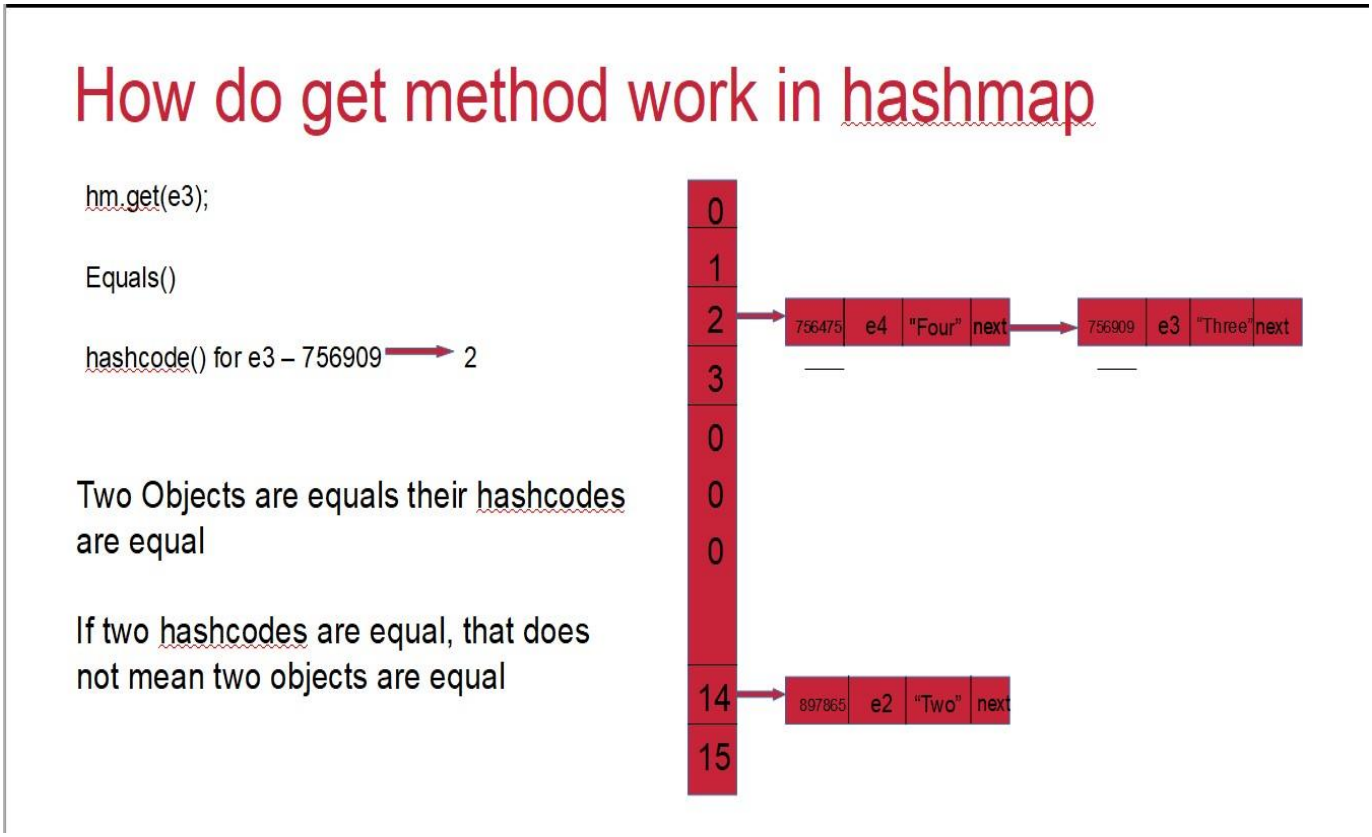
How does get() method work in HashMap?

Now let us try to retrieve an element from hashmap, let us retrieve e3,

hm.get(e3);

Now hashmap will use hashcode() and will identify the bucket in which element might be present. Next it will go to the bucket (In this case 2) and finds more than one record is available.

Now it will compare the key, if both the keys are matched then it will retrieve the value.



So When we try to get e3, first it will calculate the hashcode

hashcode() for Key e3 : 756909 and based on indexing to identify the bucket, it is calculated to be Bucket 2.

But Bucket 2, already has more than one record. Next the calculated hashcode will be compared to the inserted record in hashmap bucket 2. First it will compare 756909 with first record in hashmap that is 756475. Both are not equal, next will check the next node. Here the next node is pointing to another record. So it will go to the next record and will compare the hashcodes. 756909 and next record's hashcode 756909 both are equal. Next will take the key of that record from bucket 2 and will compare with the key user requested. In this case, e3.equals(e3) which will return true.

Since both objects are equal and hashcodes are matching, that record will be retrieved from hashbucket and given to the user. This is how get method works.

Key Note:

If two objects are equal, their hashcode will always be same
If two hashcodes are same, that doesn't mean both objects are equal.