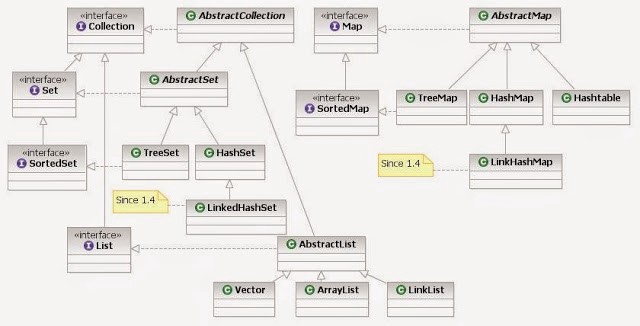
First of all, you must be familiar with the Collection tree before moving ahead for HashMap. You can find the Map interface and then the HashMap Class hierarchy.



By Ervinn at en.wikibooks [CC BY-SA 3.0 ], from Wikimedia Commons

Don’t miss the Keynotes on “the internal working of HashMap” given at the end of this article.

What is HashMap?

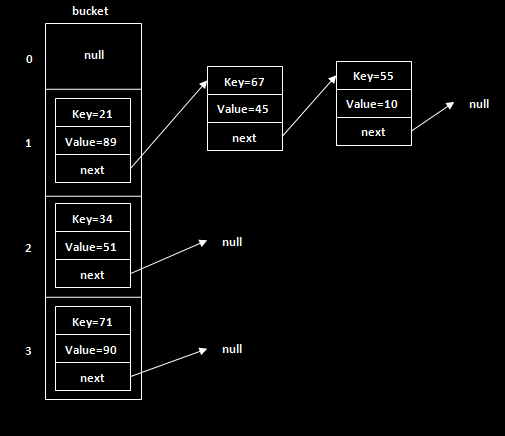
HashMap is a part of the Java collection framework. It uses a technique called Hashing. It implements the map interface. It stores the data in the pair of Key and Value. HashMap contains an array of nodes, and the node is represented as a class. It uses an array and LinkedList data structure internally for storing Key and Value. There are four fields in HashMap.

There are two main methods used for hashmap, so you must be aware of both of them.

The equals() and hashcode() are the two important methods provided by the **Object** class for comparing objects! Since the Object class is the parent class for all Java objects, hence all objects inherit the default implementation of these two methods.

* **equals():** It checks the equality of two objects. It compares the Key, whether they are equal or not. It is a method of the Object class. It can be overridden. If you override the equals() method, then it is mandatory to override the hashCode() method.
* **hashCode():** This is the method of the object class. It returns the memory reference of the object in integer form. The value received from the method is used as the bucket number. The bucket number is the address of the element inside the map. Hash code of null Key is 0.

Internal Working of HashMap in java



Manipulation of Nodes in a bucket

* **Buckets:** The array of the node is called buckets. Each node has a data structure like a LinkedList. More than one node can share the same bucket. It may be different in capacity.

We use **put()** method to insert the Key and Value pair in the HashMap. The **default size of HashMap is 16 (0 to 15)**.

HashMap map = new HashMap<>();  
map.put("Rax",19);  
map.put("Rakshit",29);  
map.put("Robert",39);

From the above code snippet, let’s see at which index the Key, value pair will be saved into HashMap. When we call the put() method, then it calculates the hash code of the Key “Rax”. Suppose the hash code of “Rax” is 2657860. To store the Key in memory, we have to calculate the index.

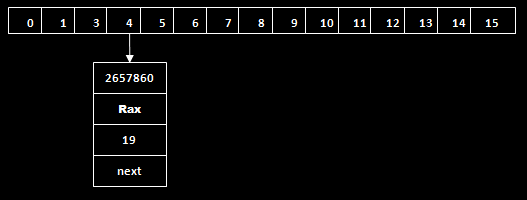
Index minimizes the size of the array. The Formula for calculating the index is:

Index = hashcode(Key) & (n-1)

Where n is the size of the array. Hence the index value for “Rax” is:

Index = 8745511 & (16–1) = 4

The value 4 is the computed index value where the Key and value will store in HashMap.

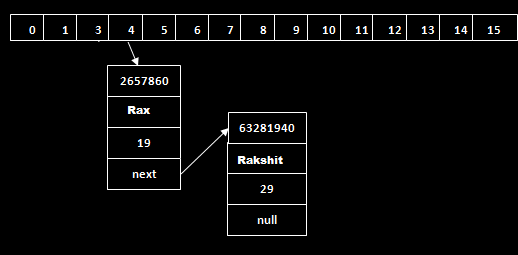


Setting node for Hashcode value = 4

Now there might be chances that another key’s hashcode is equivalent to the code for key “Rax”. It is called **“Hash Collision”.**

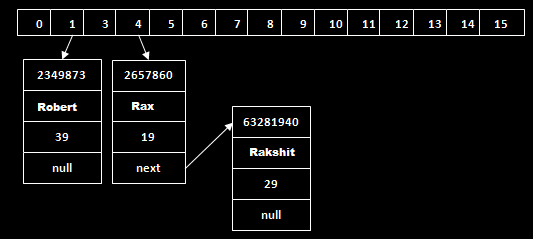
The value 4 is the computed index value where the Key will be stored in HashMap. In this case, equals() method checks that both Keys are equal or not. If Keys are the same, replace the value with the current value. Otherwise, connect this node object to the existing node object through the LinkedList. Hence both Keys will be stored at index 4.

The value 4 is the computed index value where the Key will be stored in HashMap. In this case, equals() method checks that both Keys are equal or not. If Keys are the same, replace the value with the current value. Otherwise, connect this node object to the existing node object through the LinkedList. Hence both Keys will be stored at index 4.



Hash Collision and Node manipulation scenario

Similarly, we will store the Key “Robert.” Suppose the hashcode for the Key is 2349873. The index value will be 1. Hence this Key will be stored at index 1.



Manipulation of Node with index = 1

**Now it is time for get() method.**

get() method is used to get the value by its Key. It will not fetch the value if you don’t know the Key. When get(K Key) method is called, it calculates the hash code of the Key.

Suppose we have to fetch the Key “Rax”. The following method will be called.

map.get(**new** Key(“Aman”));

It generates the hash code as 2657860. Now calculate the index value of 2657860 by using the index formula. The index value will be 4, as we have calculated above. get() method search for the index value 4. It compares the first element Key with the given Key. If both keys are equal, then it returns the value else check for the next element in the node if it exists. In our scenario, it is found as the first element of the node and returns the value 19.

Let’s fetch another Key “Rakshit.”

The hash code of the Key “Rakshit” is 63281940. The calculated index value of 63281940 is 4, as we have calculated for the put() method. Go to index 4 of the array and compare the first element’s Key with the given Key. It also compares Keys. In our scenario, the given Key is the second element, and the next of the node is null. It compares the second element Key with the specified Key and returns the value 29. It returns null if the next of the node is null.

You must be aware of the **entry class** as well.

A map by definition is: **“An object that maps keys to values…”.**

So, there must be some mechanism in HashMap to store this key-value pair. The answer is YES. HashMap has a nested static class Entry, which looks like this:

static class Entry implements Map.Entry  
{  
 final K key;  
 V value;  
 Entry next;  
 final int hash;  
 ...  
}

Surely Entry class has key and value mapping stored as attributes. key has been marked as final and two more fields are there: next and hash.

How collisions are resolved

Here comes the main part. Now, as we know that two unequal objects can have the same hash code value, how two different objects will be stored in the same array location called a bucket.

The answer is LinkedList. If you remember, Entry class had an attribute "next". This attribute always points to the next object in the chain. This is exactly the behavior of LinkedList.

**1.**So, in case of collision, Entry objects are stored in linked list form. When an Entry object needs to be stored in a particular index, HashMap checks whether there is already an entry?? If there is no entry already present, the entry object is stored in this location.

If there is already an object sitting on the calculated index, its next attribute is checked. If it is null, and current entry object becomes the next node in the linked list. If next variable is not, the procedure is followed until next is evaluated as null.

**2.**What if we add the another value object with the same key as entered before. Logically, it should replace the old value. How it is done? Well, after determining the index position of Entry object, while iterating over linkedlist on the calculated index, HashMap calls the equal method on key object for each entry object.

All these entry objects in linkedlist will have similar hashcode but equals() the method will test for true equality. If key.equals(k) will be true then both keys are treated as the same key object. This will cause the replacing of value object inside entry object only.

In this way, HashMap ensures the uniqueness of keys.

Keynotes on the internal working of HashMap

* Data structure to store entry objects is an array named table of type Entry.
* A particular index location in the array is referred to as bucket because it can hold the first element of a linked list of entry objects.
* Key object’s hashCode() is required to calculate the index location of the Entry object.
* Key object’s equals() method is used to maintain the uniqueness of keys in the map.
* Value object’s hashCode() and equals() method are not used in HashMap’s get() and put() methods.
* Hash code for null keys is always zero, and such entry object is always stored in zero indexes in Entry[].

I hope you find this article helpful. Feel free to drop any comments.

*Reference from: The Complete Reference by Herbert Schildt*