Hash Map is a Hash table based implementation of the Map interface. This implementation provides all of the optional map operations, and permits null values and the null key. (The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.) This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.” … from Java API

After reading this definition, some question comes in my mind:

How hash map store data internally?  
What happen when I try to store some new information in map?  
How hash map find my data?

And when I tried to explore it, I find it more and more interesting.

HashMap has a static class named Entry which implements Map.Entry interface. The Entry class looks like:

static class Entry implements Map.Entry {  
final Object key;  
Object value;  
final int hash;  
Entry next;  
Entry(int i, Object obj, Object obj1, Entry entry) {  
value = obj1;  
next = entry;  
key = obj;  
hash = i;

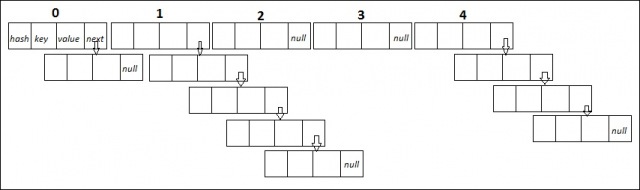
}  
// Other methods

}

Every time we insert ainto hashmap using .put() method, a new Entry object is created (not true is some cases. if key already exists, then it just replace the value). Map internally used two data structures to manage/store data:

Array  
Link List

This image shows how hashmap manage data. Here

[](http://www.techartifact.com/blogs/2012/01/how-hashmap-work-in-java.html/hashmap-design)

Each index of array is a bucket  
To identify the bucket for any , Hash map use key.hashCode() and perform some operation:  
Bucket (index) =HashMap.indexFor (HashMap.hash(key.hashCode()), entryArray.length)  
It means, two keys with different hashCode can fall under same bucket.  
If a bucket is empty (table[i] is null) then the Entry object is simply inserted at ith position  
table[i] = new Entry(hash, key, value, null)  
If a bucket has some values, then the following algo is used:  
Entry entry = table[i]

Table[i] = new Entry(hash,key,value,entry)

It means, the latest entry resides on the top of the bucket.

Load Factor- load factor is the ratio of number of keys to the length of an array. You will now have a threshold (the maximum number of elements that can be stored.The load factor is how full the HashMapis allowed to get before the capacity is doubled. The default load factor of 0.75 means that the HashMap is allowed to reach 75% capacity before it calls the rehash() method and doubles in capacity.The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased. When the number of entries in the hash table exceeds the product of the load factor and the current capacity, the hash table is rehashed,so that the hash table has been increased double the number of buckets.

Generally the default load factor of a hashtable=0.75″ is if the hashtable is 75% full, then it will be re-hashed twice of the initial capacity.  
Higher values decrease the space overhead but increase the lookup cost. The expected number of entries in the map and its load factor should be taken into account when setting its initial capacity, so as to minimize the number of rehash operations. If the initial capacity is greater than the maximum number of entries divided by the load factor, no rehash operations will ever occur.

RACE CONDITION ON HASHMAP IN JAVA

Race condition exists while resizing hashmap in Java. If two threads, at the same time, find that Hashmap needs resizing, they both try resizing the hashMap.  
In the process of resizing of hashmap, the element in bucket(which is stored in linked list) get reversed in order during the migration to new bucket, because java hashmap doesn’t append the new element at tail, instead it appends the new element at head to avoid tail traversing.  
If race condition happens then you will end up with an infinite loop.

WHAT WILL HAPPEN IF TWO DIFFERENT HASHMAP KEY OBJECTS HAVE SAME HASHCODE?  
COLLISION OCCURS-Since hashcode() is same, bucket location would be same and collision occurs in hashMap.Since HashMap use a linked list to store in bucket, “Key and Value” object will be stored in next node of linked list.

COLLISION RESOLUTION

We can find the bucket location by calling the hasCode function on the key object.After finding bucket location, we will call keys.equals() method to identify correct node in linked list and return associated value object for that key in HashMap.