How HashMap works

HashMap in Java works on hashing principle. It is a data structure which allows us to store object and retrieve it in constant time O(1) provided we know the key. In hashing, hash functions are used to link key and value in HashMap. Objects are stored by calling put(key, value) method of HashMap and retrieved by calling get(key) method. When we call put method, hashcode() method of key object is called so that hash function of map can find a bucket location to store value object, which is actually index of internal array, known as table. HashMap internally store mapping in form of Map.Entry object which contains both key and value object. When you want to retrieve the object, you call get() method and again pass key object. This time again key object generate same hash code (it's mandatory for it to do so to retrieve object and that's why HashMap keys are immutable e.g. String) and we end up at same bucket location. If there is only one object then it is returned and that's your value object which you have stored earlier. Things get little tricky when collisions occurs. Since internal array of HashMap is of fixed size, and if you keep storing objects, at some point of time hash function will return same bucket location for two different keys, this is called collision in HashMap. In this case, a linked list is formed at that bucket location and new entry is stored as next node. If we try to retrieve object from this linked list, we need an extra check to search correct value, this is done by equals() method. Since each node contains an entry, HashMap keep comparing entry's key object with passed key using equals() and when it return true, Map returns corresponding value. Since searching in lined list is O(n) operation, in worst case hash collision reduce a map to linked list. This issue is recently addressed in Java 8 by replacing linked list to tree to search in O(logN) time. By the way, you can easily verify how HashMap work by looking at code of HashMap.java in your Eclipse IDE, if you know [how to attach source code of JDK in Eclipse](http://javarevisited.blogspot.com/2012/12/how-to-attach-source-in-eclipse-Jar-JDK-debugging.html).  
  
  
How HashMap works in Java or sometime how get method work in HashMap is a very common question on Java interviews now days. Almost everybody who worked in Java knows about HashMap, where to use HashMap and difference between Hashtable and HashMap then why this interview question becomes so special? Because of the depth it offers. It has become very popular Java interview question in almost any senior or mid-senior level Java interviews. Investment banks mostly prefer to ask this question and some time even ask you to implement your own HashMap based upon your coding aptitude. Introduction of [ConcurrentHashMap](http://javarevisited.blogspot.co.uk/2013/02/concurrenthashmap-in-java-example-tutorial-working.html) and other concurrent collections has also made this questions as starting point to delve into more advanced feature. let's start the journey.

**How HashMap Internally Works in Java**

Questions start with simple statement :

**Have you used HashMap before**or**What is HashMap? Why do you use it**

Almost everybody answers this with yes and then interviewee keep talking about common facts about HashMap like HashMap accept null while Hashtable doesn't, [HashMap is not synchronized](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html), HashMap is fast and so on along with basics like its stores key and value pairs etc. This shows that person has used HashMap and quite familiar with the functionality it offers, but interview takes a sharp turn from here and next set of follow-up questions gets more detailed about fundamentals involved with HashMap in Java . Interviewer strike back with questions like :

**Do you Know how HashMap works in Java** or **How does get () method of HashMap works in Java**

And then you get answers like,  I don't bother its standard Java API, you better look code on Java source or Open JDK; I can find it out in Google at any time etc. But some interviewee definitely answer this and will say **HashMap works on principle of hashing**, we have put(key, value) and get(key) method for storing and retrieving Objects from HashMap. When we pass Key and Value object  toput() method on Java HashMap, HashMap implementation calls [hashCode method](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html)on Key object and applies returned hashcode into its own hashing function to find a bucket location for storing Entry object, important point to mention is that HashMap in Java stores both key and value object as Map.Entry in bucket which is essential to understand the retrieving logic. If people fails to recognize this and say it only stores Value in the bucket they will fail to explain the retrieving logic of any object stored in Java HashMap . This answer is very much acceptable and does make sense that interviewee has fair bit of knowledge on how hashing works and how HashMap  works in Java. But this is just start of story and confusion increases when you put interviewee on scenarios faced by Javadevelopers on day by day basis. Next question could be about collision detection and collision resolution in Java HashMap  e.g.

**What will happen if two different objects have same hashcode?**

Now from here onwards real confusion starts, Some time candidate will say that since hashcode is equal, both objects are equal and HashMap  will throw exception or not store them again etc, Then you might want to remind them about [equals() and hashCode() contract](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html)that two unequal object in Java can have same hash code. Some will give up at this point and few will move ahead and say "Since hashcode is same, bucket location would be same and collision will occur in HashMap, Since HashMap use LinkedList to store object, this entry (object of Map.Entry comprise key and value )  will be stored in [LinkedList](http://javarevisited.blogspot.sg/2012/02/difference-between-linkedlist-vs.html). Great this answer make sense though there are many collision resolution methods available  like linear probing and chaining, this is simplest and HashMap in Java does follow this. But story does not end here and interviewer asks

**How will you retrieve Value object  if two Keys will have same hashcode?**

[how HashMap works internally in Java](http://2.bp.blogspot.com/-wrzDeQGAe1I/TWu8pLuLr4I/AAAAAAAAADE/V017G-6Q61w/s1600/java_logo_50_50.jpg)Interviewee will say we will call get() method and then HashMap uses Key Object's hashcode to find out bucket location and retrieves Value object but then you need to remind him that there are two Value objects are stored in same bucket , so they will say about [traversal in LinkedList](http://javarevisited.blogspot.sg/2010/10/how-do-you-find-length-of-singly-linked.html)until we find the value object , then you ask *how do you identify value object because you don't  have value object to compare* ,Until they know that HashMap  stores both Key and Value in LinkedList node or as Map.Entry they won't be able to resolve this issue and will try and fail.

But those bunch of people who remember this key information will say that after finding bucket location , we will **call keys.equals() method** to identify correct node in LinkedList and return associated value object for that key in Java HashMap . Perfect this is the correct answer.

In many cases interviewee fails at this stage because they get confused between[hashCode()](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html) and equals(**)** or keys and values object in Java HashMap  which is pretty obvious because they are dealing with the hashcode() in all previous questions and equals() come in picture only in case of retrieving value object from HashMap in Java. Some good developer point out here that using immutable,[final object](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) with proper equals() and hashcode() implementation would act as perfect Java HashMap  keys and**improve performance of Java HashMap  by reducing collision**. Immutability*also allows caching there hashcode of different keys* which makes overall retrieval process very fast and suggest that [String](http://javarevisited.blogspot.sg/2011/07/string-vs-stringbuffer-vs-stringbuilder.html)and various wrapper classes e.g. Integer very good keys in Java HashMap.

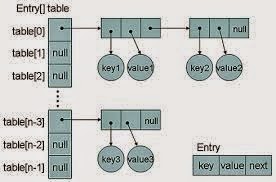
[](http://4.bp.blogspot.com/-adRczhctozE/VD_eimhTQbI/AAAAAAAACCg/lfA1G5GZXyM/s1600/How%2BHashMap%2Bworks%2Bin%2BJava%2B(1).jpg)

Figure HashMap

Now if you clear this entire Java HashMap interview,  You will be surprised by this very interesting question "**What happens On HashMap in Java if the size of the HashMap  exceeds a given threshold defined by load factor ?"**. Until you know how HashMap  works exactly you won't be able to answer this question. If the size of the Map exceeds a given threshold defined by load-factor e.g. if load factor is .75 it will act to re-size the map once it filled 75%. Similar to other collection classes like [ArrayList](http://javarevisited.blogspot.sg/2011/05/example-of-arraylist-in-java-tutorial.html),  Java HashMap re-size itself by creating a new bucket array of size twice of previous size of HashMap , and then start putting every old element into that new bucket array. This process is called rehashing because it also applies hash function to find new bucket location.

If you manage to answer this question on HashMap in Java you will be greeted by **"do you see any problem with resizing of HashMap  in Java"**, you might not be able to pick the context and then he will try to give you hint about multiple thread accessing the Java HashMap and potentially looking for **race condition on HashMap  in Java**.

So the answer is Yes there is potential [race condition](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) exists while resizing HashMap in Java, if two [thread](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html)at the same time found that now HashMap needs resizing and they both try to resizing. on the process of resizing of HashMap in Java , the element in bucket which is stored in linked list get reversed in order during there migration to new bucket because Java HashMap  doesn't append the new element at tail instead it append new element at head *to avoid tail traversing*. If race condition happens then you will end up with an infinite loop. Though this point you can potentially argue that what the hell makes you think to use HashMap  in multi-threaded environment to interviewer :)

Read more: <http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html#ixzz3mvbP9wAH>

Summary:

1. The Bucket is a table of entities. And each entry contains fields of key, value, hash value and the reference of next entity whoes hash value is the same as this one.
2. The value of the an entry is just a reference of an object that means the space cost of an entity is fixed: key(variable)+value(4 bytes in 32bit system or 8 bytes in 64bit system, like pointer in c++)+hash value(4 bytes or 8 bytes)+next(same as value). So In general, HashMap is not so space consuming as I think before.
3. The hash value is calculated by the hashcode method of the key object, so if we want to use objct of custom class as the key, it should override hashcode method of Object.
4. Java use linked list to deal with collision. And if the keys are comparable, it apply a tree method to reduce the searching time from O(n) to O(log(n))