**[How HashMap works in Java](https://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html" \o "How HashMap works in Java)**

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 the key object is called so that hash function of the map can find a bucket location to store value object, which is actually an index of the internal array, known as the table. HashMap internally stores mapping in the form of **Map.Entry** object which contains both key and value object. When you want to retrieve the object, you call [the get() method](http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html) and again pass the key object. This time again key object generate same hash code (it's mandatory for it to do so to retrieve the 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](http://java67.blogspot.com/2012/09/top-10-tricky-java-interview-questions-answers.html) when collisions occur. It's easy to answer this question if you have read good book or course on data structure and algorithms like [this](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https%3A%2F%2Fwww.pluralsight.com%2Fcourses%2Fads-part1)one. If you know how hash table data structure works then this is a piece of cake.  
  
Since the 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 a new entry is stored as next node.  
  
If we try to retrieve an 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 keeps comparing entry's key object with the passed key using equals() and when it return true, Map returns the corresponding value.  
  
  
Since searching inlined 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 the tree to search in O(logN) time. By the way, you can easily verify how HashMap works by looking at the 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 sometimes how does get method work in HashMap is a very common question on Java interviews nowadays. 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](http://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html) in almost any senior or mid-senior level Java interviews. Investment banks mostly prefer to ask this question and sometimes even ask you to implement your own HashMap based upon your coding aptitude. The 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 a 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 answers this and will say **HashMap works on the 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  to put() 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 a bucket which is essential to understand the retrieving logic.   
  
If people fail 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 a 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 Java developers 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 the same hashcode?**

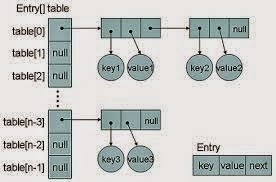
Now from here onwards real confusion starts, sometime candidate will say that since hashcode is equal, both objects are equal and HashMap  will throw exception or not store them again etc, Then you might want to remind them about [equals() and hashCode() contract](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html)that two unequal objects in Java can have same hashcode. 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 uses 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 the same hashcode?**

[how HashMap works internally in Java](https://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 a 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 the performance of Java HashMap  by reducing collision**. Immutability*also allows caching their 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.

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

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 the 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 the 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 the 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 the bucket which is stored in linked list get reversed in order during their migration to new bucket because Java HashMap  doesn't append the new element at tail instead it append new element at the 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 :)

## Some more Hashtable and HashMap Questions

Few more question on HashMap in Java which is contributed by readers of Javarevisited blog:

**1) Why String, Integer and other wrapper classes are considered good keys?**

String, Integer and other wrapper classes are natural candidates of HashMap key, and String is most frequently used key as well because [String is immutable and final](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html), and overrides equals and hashcode() method. Other wrapper class also shares similar property. Immutability is required, in order to prevent changes on fields used to calculate hashCode() because if key object returns different hashCode during insertion and retrieval than it won't be possible to get an object from HashMap.   
  
Immutability is best as it offers other advantages as well like [thread-safety](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html), If you can  keep your hashCode same by only making certain fields final, then you go for that as well. Since equals() and hashCode() method is used during retrieval of value object from HashMap, it's important that key object correctly override these methods and follow contact. If unequal object returns different hashcode than chances of collision will be less which subsequently improve the performance of HashMap.

**2) Can we use any custom object as a key in HashMap?**

This is an extension of previous questions. Of course you can use any Object as key in Java HashMap provided it follows equals and hashCode contract and its hashCode should not vary once the object is inserted into [Map](http://javarevisited.blogspot.sg/2011/12/how-to-traverse-or-loop-hashmap-in-java.html). If the custom object is Immutable than this will be already taken care because you can not change it once created.

**3) Can we use ConcurrentHashMap in place of Hashtable?**

This is another question which getting popular due to increasing popularity of ConcurrentHashMap. Since we know Hashtable is synchronized but ConcurrentHashMap provides better concurrency by only locking portion of map determined by concurrency level. ConcurrentHashMap is certainly introduced as Hashtable and can be used in place of it, but Hashtable provides stronger thread-safety than ConcurrentHashMap. See my post [difference between Hashtable and ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html) for more details.

Personally, I like this question because of its depth and number of concept it touches indirectly if you look at questions asked during interview this HashMap  questions has verified

* The concept of hashing
* Collision resolution in HashMap
* Use of equals () and hashCode () and their importance in HashMap?
* The benefit of the immutable object?
* Race condition on HashMap  in Java
* Resizing of Java HashMap

Just to summarize here are the answers which do make sense for above questions

**How HashMap  works in Java**

HashMap  works on the principle of hashing, we have put() and get() method for storing and retrieving object from HashMap.When we pass both key and value to put() method to store on HashMap, it uses key object hashcode() method to calculate hashcode and them by applying hashing on that hashcode it identifies bucket location for storing value object. While retrieving it uses key object equals method to find out correct key value pair and return value object associated with that key. HashMap  uses linked list in case of collision and object will be stored in next node of linked list. Also, [HashMap stores both key and value tuple](http://java67.blogspot.com/2013/02/10-examples-of-hashmap-in-java-programming-tutorial.html) in every node of linked list in the form of Map.Entry object.

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

They will be stored in the same bucket but no next node of linked list. And keys equals () method will be used to identify correct key value pair in HashMap.

**How null key is handled in HashMap? Since equals() and hashCode() are used to store and retrieve values, how does it work in case of the null key?**  
The null key is handled specially in HashMap, there are two separate methods for that putForNullKey(V value) and getForNullKey(). Later is offloaded version of get() to look up null keys.  Null keys always map to index 0.  This null case is split out into separate methods for the sake of performance in the two most commonly used operations (get and put), but incorporated with conditionals in others. In short, equals() and hashcode() method are not used in case of null keys in HashMap.  
  
here is how nulls are retrieved from HashMap  
  
   **private** V **getForNullKey**() {

**if** (size == **0**) {

**return** **null**;

}

**for** (Entry<K,V> e = table[**0**]; e != **null**; e = e.next) {

**if** (e.key == **null**)

**return** e.value;

}

**return** **null**;

}

In terms of usage, Java HashMap is very versatile and I have mostly used HashMap as cache in an electronic trading application I have worked. Since finance domain used Java heavily and due to performance reason we need caching HashMap and ConcurrentHashMap  comes as very handy there. You can also check following articles from Javarevisited to learn more about HashMap and Hashtable in Java:

## HashMap Changes in JDK 1.7 and JDK 1.8

There is some [performance improvement done on HashMap and ArrayList from JDK 1.7](http://javarevisited.blogspot.com/2014/07/java-optimization-empty-arraylist-and-Hashmap-cost-less-memory-jdk-17040-update.html), which reduce memory consumption. Due to this empty Map are lazily initialized and will cost you less memory. Earlier, when you create HashMap e.g. new HashMap() it automatically creates an array of default length e.g. 16. After some research, Java team found that most of this Map are temporary and never use that many elements, and only end up wasting memory. Also, From JDK 1.8 onwards HashMap has introduced an improved strategy to deal with high collision rate. Since a poor hash function e.g. which always return location of same bucket, can turn a HashMap into linked list, i.e. converting get() method to perform in O(n) instead of O(1) and someone can take advantage of this fact, Java now internally replace linked list to a binary true once certain threshold is breached. This ensures performance or order O(log(n)) even in the worst case where a hash function is not distributing keys properly.  
  
**Further Reading**  
If you are preparing for Java interviews then I suggest you take a look at [Java Interview Guide: 200+ Java Interview questions](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https%3A%2F%2Fwww.udemy.com%2Fjava-interview-questions-and-answers%2F), an excellent resource for all level of Java programmers. This book contains questions from all important Java topics. You can also join [Data Structures and Algorithms](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https%3A%2F%2Fwww.udemy.com%2Fdata-structures-and-algorithms-deep-dive-using-java%2F) course to brush up some of your data structure and algorithm skills.

Read more: <https://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html#ixzz6OZWvyDrN>