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

**"Have you used HashMap before"** or**"What is HashMap? Why do we 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](http://www.blogger.com/), [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 HashMap 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 struck 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  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 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 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 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 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 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 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(**)**](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html) 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.

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 :)

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. Immutabiility is required, in order to prevent changes on fields used to calculate hashCode() because if key object return different hashCode during insertion and retrieval than it won't be possible to get 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](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html), then you go for that as well. Since equals() and hashCode() method is used during reterival of value object from HashMap, its important that key object correctly override these methods and follow contact. If unequal object return different hashcode than chances of collision will be less which subsequently improve performance of HashMap.

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

This is an extension of previous questions. Ofcourse 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 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 provide 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

* Concept of hashing
* Collision resolution in HashMap
* Use of equals () and hashCode () and there importance in HashMap?
* Benefit of immutable object?
* Race condition on HashMap  in Java
* Resizing of Java HashMap

Just to summarize here are the answers which does makes sense for above questions

**How HashMap  works in Java**

HashMap  works on principle of hashing, we have put() and get() method for storing and retrieving object form HashMap .When we pass an both key and value to put() method to store on HashMap , it uses key object hashcode() method to calculate hashcode and they 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+value tuple in every node of linked list.

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

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

In terms of usage Java HashMap is very versatile and I have mostly used HashMap as cache in 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.

### Difference between ConcurrentHashMap and Hashtable

So **what is the difference between Hashtable and ConcurrentHashMap** , both can be used in multithreaded environment but once the size of Hashtable becomes considerable large performance degrade because for iteration it has to be **locked**for longer duration.

Since **ConcurrentHashMap introduced concept of segmentation** , how large it becomes only certain part of it get locked to provide thread safety so many other readers can still access map without waiting for iteration to complete.

In Summary **ConcurrentHashMap only locked certain portion of Map** while Hashtable lock full map while doing iteration.

### Difference between ConcurrentHashMap and Collections.synchronizedMap

ConcurrentHashMap is designed for concurrency and improve performance while HashMap which is non synchronized by nature can be synchronized by applying a wrapper using Collections.synchronizedMap. Here are some of common **differences between ConcurrentHashMap and Collections.synchronizedMap**

**ConcurrentHashMap do not allow null keys or null values** while HashMap allows null keys.

# Differences between HashMap and a Hashtable

**1)** [Hashtable](https://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html" \t "_blank) is synchronized (i.e. methods defined inside Hashtable), whereas [HashMap](https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html" \t "_blank) is not. If you want to make a HashMap thread-safe, use Collections.synchronizedMap(map) or ConcurrentHashMap class.

Methods inside HashTable are defined synchronized as below:

|  |
| --- |
| public synchronized boolean contains(Object obj){ ... }  public synchronized boolean containsKey(Object obj){ ... }  public synchronized Object get(Object obj){ ... }  public synchronized Object put(Object obj, Object obj1){ ... }  public synchronized Object remove(Object obj){ ... } |

**2)** **Hashtable** **does not allow null keys or values**. HashMap **allows one null key** (other null keys will simply overwrite first null key) and **any number of null values**.

|  |
| --- |
| Hashtable<String, String> hashTable = new Hashtable<String, String>();  hashTable.put(null, "value");  //OR  hashTable.put("key", null);    Output:    Exception in thread "main" java.lang.NullPointerException      at java.util.Hashtable.hash(Unknown Source)      at java.util.Hashtable.put(Unknown Source)      at test.core.MapExamples.main(MapExamples.java:12) |

**3)** **Hashtable is legacy class** and was not part of the initial Java Collections Framework (later it was included in JDK 1.2). **HashMap is part of Collections since it’s birth**. Also note that Hashtable extends the Dictionary class, which as the Javadocs state, is obsolete and has been replaced by the Map interface in newer JDK versions.

|  |
| --- |
| //HashTable is defined as  public class Hashtable extends Dictionary implements Map, Cloneable, Serializable {}  //HashMap is defined as  public class HashMap extends AbstractMap implements Map, Cloneable, Serializable {} |

**4)** Iterator in the HashMap **is fail-fast** and throw ConcurrentModificationException if any other Thread modifies the map structurally by adding or removing any element except Iterator’s own **remove()** method. But this is not a guaranteed behavior and will be done by JVM on best effort. The **enumerator for the Hashtable is not fail-fast**.

|  |
| --- |
| HashMap<String, String> hashMap = new HashMap<String, String>();  hashMap.put("key1", "value1");  hashMap.put("key2", "value2");  hashMap.put("key3", "value3");  hashMap.put("key4", "value4");    Iterator<String> iterator = hashMap.keySet().iterator();  while(iterator.hasNext()){      iterator.next();      iterator.remove();      System.out.println(hashMap);  }    Output:    {key3=value3, key2=value2, key1=value1}  {key2=value2, key1=value1}  {key1=value1}  {} |

**5)** Finally, Map fixes a minor deficiency in the Hashtable interface. Hashtable has a method called “[**contains()**](https://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html#contains%28java.lang.Object%29)” (along with “**containsValue()**” and “**containsKey()**“), which returns true if the Hashtable contains a given value. Given its name, you may expect this method to return true if the Hashtable contained a given key, because the key is the primary access mechanism for a Hashtable. The Map interface eliminates this source of confusion by removing this method to and has only “**containsValue()**” and “**containsKey()**“.

|  |
| --- |
| public boolean containsKey(Object obj) {...}  public boolean containsValue(Object obj) {...} |

## Suggestion regarding usage of HashMap vs Hashtable

There is hardly any job which HashMap or it’s related classes (i.e. LinkedHashMap or ConcurrentHashMap) can not do which HashTable does. So, there is no good reason to use Hashtable in new code you write. **Always prefer to use HashMap over HashTable**.