<http://javarevisited.blogspot.sg/2015/08/difference-between-HashMap-vs-TreeMap-vs-LinkedHashMap-Java.html>

Difference between HashMap, LinkedHashMap and TreeMap in Java

Map is one of the most important data structure from Java Collection Framework.  It provides hash table data structure functionality with it's rich implementations like HashMap, Hashtable, LinkedHashMap and little bit of sorting with TreeMap. So if you are looking to store key value pairs in Java program,  you have wide range of choices available depending upon your requirement. Main difference between LinkedHashMap, TreeMap and HashMap comes in there internal implementation and specific features, which makes them useful in certain scenarios. For example, [HashMap](http://java67.blogspot.sg/2013/02/10-examples-of-hashmap-in-java-programming-tutorial.html) is a general purpose Map (hash table data structure), which should be used whenever you need a hashing based data structure for storing your mappings (key value pairs). **TreeMap** provides you sorting, on top of hashing offered by Map interface, which means you can not only retrieve elements in constant time i.e. O(1) time, but also iterate through those mapping in a predefined sorted order, but you need to pay heavy price to keep mappings in sorted order. On the other hand, **LinkedHashMap** is a compromise between these two, it doesn't provide sorting but unlike HashMap, it provides ordering e.g. maintaining mappings in a order they are inserted into Map, known as *insertion order* or order on which they are accessed, called *access order*. Apart from these three popular Map implementation, you also have some special purpose Map implementations e.g. [EnumMap](http://javarevisited.blogspot.sg/2012/09/what-is-enummap-in-java-example-tutorial.html) for storing mapping with enum constants as keys,  it is highly optimized for enum constants. You also have a special map called [WeakHashMap](http://javarevisited.blogspot.sg/2014/03/difference-between-weakreference-vs-softreference-phantom-strong-reference-java.html) for creating a Garbage Collector friendly Cache, where values become eligible for garbage collection as soon as there is no other reference to them apart from keys in WeakHashMap.Then there is [IdentityHashMap](http://javarevisited.blogspot.sg/2013/01/difference-between-identityhashmap-and-hashmap-java.html) for creating a Map which uses identity instead of equality for comparing keys, since identity equality is rare, you get less number of collision on this Map and finally JDK 5 introduced [ConcurrentHashMap](http://javarevisited.blogspot.sg/2013/02/concurrenthashmap-in-java-example-tutorial-working.html) for better scalability in multi-threaded environment, where number of reader threads clearly out numbers number of writer threads.

## LinkedHashMap vs TreeMap vs HashMap

Though all three classes implements java.util.Map interface and follows general contract of a Map interface, defined in terms of [equals() and hashCode()](http://javarevisited.blogspot.sg/2015/01/why-override-equals-hashcode-or-tostring-java.html) method, they also have several differences in terms of Ordering, Sorting, permitting null elements, Iteration, Performance, Speed and internal implementation. Let's have a quick look on each of these property.

#### Ordering and Sorting

HashMap doesn't provide any ordering guarantee for entries, which means, you can not assume any order while [iterating over keys and values of HashMap](http://java67.blogspot.sg/2013/08/best-way-to-iterate-over-each-entry-in.html). This behavior of HashMap is similar to Hashtable, while other two Map implementation provides ordering guarantee.

LinkedHashMap can be used to maintain insertion-order, on which keys are inserted into Map or it can also be used to maintain an access-order, on which keys are accessed. This provides LinkedHashMap an edge over HashMap without compromising too much performance.

TreeMap provides you complete control over sorting elements by passing [custom Comparator](http://javarevisited.blogspot.sg/2014/01/java-comparator-example-for-custom.html) of your choice, but with expense of some performance. Since entries are stored in a tree based data structure, it provides lower performance than HashMap and LinkedHashMap.

#### Null keys and Values

HashMap allows one null key and multiple null values. It keeps null key based entries on index[0] on internal bucket. If you look at put() method of HashMap, you can see, it doesn't throw [NullPointerException for null keys](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html). Since LinkedHashMap is a sub class of HashMap, it also allows null keys and values.

On the other hand TreeMap, which sorts elements in natural order doesn't allow null keys because compareTo() method throws NullPointerException if compared with null. If you are using TreeMap with [user defined Comparator](http://java67.blogspot.sg/2014/11/java-8-comparator-example-using-lambda-expression.html), than it depends upon implementation of compare() method.

#### Iterators

Iterators returned by all these Map's collection view methods e.g. values() or keySet() is [fail-fast iterators](http://java67.blogspot.sg/2015/06/what-is-fail-safe-and-fail-fast-iterator-in-java.html), which means they will throw ConcurrentModificatoinException, if Collection is modified structurally once Iteration begins, except by using remove() method of Iterator.

By the way, it's worth remembering that apart from adding or removing more mappings, it can also be any operation which affects iteration order of LinkedHashMap. In access-ordered LinkedHashMap, even querying the Map with get() method is a structural modification, because it changes the iteration order, on the other hand updating value in a insertion-ordered linked hash map is not a structural modification.

Finally, fail-fast behavior is not guaranteed, and they throw ConcurrentModificationException on best-effort basis, which means do not write code, which depends upon this behavior. It should only be used to detect programming bugs.

#### Performance and Speed

Since HashMap is bare bone implementation of java.util.Map interface, it provides constant time performance for get() and put() operation, where put is used to store key value pair and get is used to retrieve value based upon key. BTW, constant time performance is only if mappings are distributed uniformly across bucket location, but HashMap is certainly faster than Hashtable because it's not synchronized. [Iteration over Map](http://java67.blogspot.sg/2014/05/3-examples-to-loop-map-in-java-foreach.html) is directly proportional to the "capacity" + "size" of HashMap, that's why it's important to set the initial capacity high enough, if iteration performance is important. You can further use **initial capacity** and **load factor** to fine tune your HashMap performance, to avoid rehashing of HashMap.

Since TreeMap is based on tree data structure, it provides log(n) time for get(), put(), containsKey() and remove() operation, so it's costlier than HashMap, if order is not concerned.

LinkedHashMap is a trade-off between two, like HashMap it also provides constant time performance for add, contains and remove, though its slightly slower than HashMap, to maintain linked list. By the way Iteration over Map in case of LinkedHashMap is slightly faster than HashMap, because time required is proportional to size only. So if you need insertion order or *access order*, consider using LinkedHashMap over TreeMap in Java.

#### Thread-safety and Synchronization

All three Map implementation are [not thread-safe](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html), which means you can not use them safely in a multi-threaded application. Though you can synchronized them externally by using Collections.synchronizedMap(Map map) method. Alternatively you can also use there concurrent counter part e.g. ConcurrentHashMap which is also a better choice than HashMap in a concurrent Java application.

When using synchronized Map e.g. synchronized LinkedHashMap or SortedMap, you must do at the time or creating map to prevent accidental non synchronized access. You can use following idiom to create Synchronized Map in Java :

**Synchronized LinkedHashMap**

Map<Integer, Integer> numbers **=** Collections**.**synchronizedMap(**new** LinkedHashMap<>());

**Synchronized TreeMap**

SortedMap<Integer, String> sorted **=** Collections**.**synchronizedSortedMap(**new** TreeMap<>());

Remember to use Collections.synchronizedMap() for synchronizing HashMap, LinkedHashMap and Collections.synchronizedSortedMap() method for synchronizing TreeMap. If you are not comfortable then see this guide on [how to synchronize HashMap in Java](http://java67.blogspot.sg/2015/02/how-to-synchronize-hashmap-in-java-with.html).

#### Internal Implementation

TreeMap is Red-Black tree based NavigableMap implementation, while HashMap is internally backed by an array. It uses index[0] to store entries corresponding to null keys. In fact questions related to inner working of HashMap is very popular in Java, for example [How get() method of HashMap works internally](http://java67.blogspot.sg/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html) is one of the frequently used questions to Senior Java developers. On the other hand LinkedHashMap extends HashMap and uses linked list to provide insertion order guarantee. It uses doubly linked list running through all of it's entries, which can also be used to maintain access-order. Remember, insertion order is not affected if a key is re-inserted into LinkedHashMap, but access order is affected, if LinkedHashMap is created to maintain access-order.

## When to use LinkedHashMap, TreeMap and HashMap in Java

You can use a LinkedHashMap, when you need to keep your mappings in either **insertion order** or **access-order**. LinkedHashMap by default keeps elements in the order, on which they are inserted, and this order is reflected when you [traverse over LinkedHashMap](http://javarevisited.blogspot.sg/2011/12/how-to-traverse-or-loop-hashmap-in-java.html), but it also provides a constructor, which allows you to keep entries in *access-order*, i.e. order in which they are accessed. One of the clever use of Java LinkedHashMap is to use it as Least Recently Use or **LRU Cache**.

**TreeMap** is your go to map implementation if you want to keep keys  in a sorted order, either in there natural order defined by Comparable interface or a custom order imposed by Comparator interface, though it's worth remembering that your compareTo() or compare() method must be [consistent with equals() method](http://java67.blogspot.sg/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html), because Map interface is defined in terms of equals and TreeMap uses compareTo for comparing keys. So if keys compare() or compareTo() implementation is not consistent, then it will fail to obey Map's general contract.

**HashMap** is your general purpose hashing based collection, whenever you need to use a hash table data structure in Java to store key value pairs, first choice goes to HashMap in single threaded environment. If you happened to use a Map in a multi-threaded environment consider using [Hashtable, synchronized HashMap or ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html) from Java Collection Framework.

Since LinkedHashMap solved problem of chaotic ordering provided by Hashtable and HashMap, without incurring high cost associated with TreeMap, you can also used LinkedHashMap to create a copy of a Map in Java, as shown in below example.

### An example of using LinkedHashMap, TreeMap and HashMap in Java

Let's see an example of how to use these Map implementations. In this example, we will use HashMap to create a general purpose Cache, TreeMap to create a sorted Cache and we will use LinkedHashMap for copying a Map (cache) and maintaining orders in the original Map.

import java.util.Collections;

import java.util.HashMap;

import java.util.LinkedHashMap;

import java.util.Map;

import java.util.SortedMap;

import java.util.TreeMap;

/\*\*

\* Java Program to demonstrate How to use LinkedHashMap, TreeMap and HashMap.

\* It shows that HashMap doesn't guarantee any order, TreeMap keeps them in

\* sorted order determined by default using Comparable or explicit Comparator

\* provided by client, and LinkedHashMap also keep mapping in order they

\* are added or accessed.,

\*

\* @author Javin Paul

\*/

public class MapTest {

public static void main(String args[]){

//Using HashMap as general purpose single threaded cache

Map<Integer, String> cache = new HashMap<>();

cache.put(1, "Stuart");

cache.put(2, "Steven");

cache.put(3, "James");

cache.put(4, "Ian");

System.out.printf("Name of Employee with id %d is %s %n", 1, cache.get(1));

System.out.println("Order of Entries in HashMap - Not guaranteed");

System.out.println(cache);

//Using TreeMap to create a sorted cache, sorting keys on reverse order

SortedMap<Integer, String> sortedCache = new TreeMap<>(Collections.reverseOrder());

sortedCache.putAll(cache);

System.out.println("Order of Entries in TreeMap - Sorted in reverse order");

System.out.println(sortedCache);

//Using LinkedHashMap to create copy of a Map in Java

Map<Integer, String> copy = new LinkedHashMap<>(sortedCache);

System.out.println("Order of Entries in a copy Map created by LinkedHashMap");

System.out.println(copy);

}

}

**Output:**

Name of Employee with id 1 is Stuart

Order of Entries in HashMap - Not guaranteed

{1=Stuart, 2=Steven, 3=James, 4=Ian}

Order of Entries in TreeMap - Sorted in reverse order

{4=Ian, 3=James, 2=Steven, 1=Stuart}

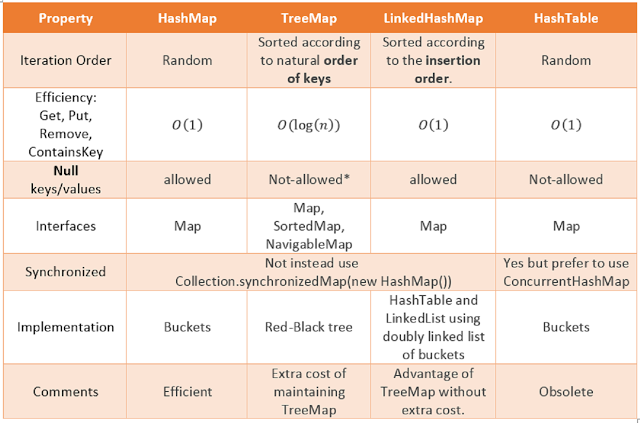
Order of Entries in a copy Map created by LinkedHashMap

{4=Ian, 3=James, 2=Steven, 1=Stuart}

You can see that TreeMap has sorted mappings in reverse order, because of reverse Comparator provided to it. Also LinkedHashMap has created a copy of TreeMap and order of entries are retained.

### Summary

Here is the summary of differences between HashMap, LinkedHashMap and TreeMap in Java  :

[](http://2.bp.blogspot.com/--9mLb1kGbjU/VdsKlyusXwI/AAAAAAAADo0/tYU-8u-qLmU/s1600/Difference+between+HashMap,+TreeMap,+LinkedHashMap+and+hashtable+in+Java.png)

That's all on **difference between LinkedHashMap, TreeMap and HashMap in Java**. Though all three are Map implementation, they have different purpose and used accordingly. Use LinkedHashMap, if you need to maintain insertion or access order of mappings e.g. in LRU Cache. Use TreeMap, if you need to maintain mappings in a sorted order, either in there natural order or a custom order defined by Comparator and use HashMap for all your general purpose hashing based collection requirement. HashMap allows you to retrieve object in O(1) time, if you know key.

**Further Reading**

If you are really serious about mastering various classes from Java Collection framework then I suggest you to read following two books, both of them are classic and highly recommended for experienced Java developer who wants to constantly improve their skill and knowledge of Java API.

* Java Generics and Collection By Maurice Naftalin ([see here](http://www.amazon.com/dp/0596527756/?tag=javamysqlanta-20))
* Effective Java 2nd Edition by Joshua Bloch ([see here](http://www.amazon.com/dp/0321356683/?tag=javamysqlanta-20))

If you like this article and want to know more about other Collection classes, do check following interview questions analysis and explanation from Java Collection framework :

* What is difference between HashSet, TreeSet and LinkedHashSet in Java? ([answer](http://javarevisited.blogspot.sg/2012/11/difference-between-treeset-hashset-vs-linkedhashset-java.html))
* What is difference between HashMap and ArrayList in Java? ([answer](http://java67.blogspot.sg/2012/08/difference-between-hashmap-and-ArrayList-in-Java.html))
* What is difference between HashSet and ArrayList in Java? ([answer](http://java67.blogspot.sg/2012/07/difference-between-arraylist-hashset-in-java.html))
* 5 differences between HashMap and Hashtable in Java? ([answer](http://java67.blogspot.sg/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html))
* What is difference between ArrayList and LinkedList in Java? ([answer](http://java67.blogspot.sg/2012/12/difference-between-arraylist-vs-LinkedList-java.html))
* How to use NavigableMap in Java 6? [[example](http://javarevisited.blogspot.sg/2013/01/what-is-navigablemap-in-java-6-example-submap-head-tail.html)]
* How to use BlockingQueue in Java Program? [[example](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html)]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

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 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  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?**

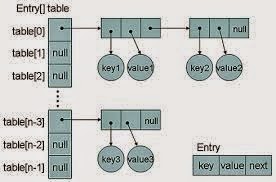
Now from here onwards real confusion starts, Some time candidate will say that since hashcode is equal, both objects are equal and HashMap  will throw exception or not store them again etc, Then you might want to remind them about [equals() and hashCode() contract](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html)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

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+HashMap+works+in+Java+(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 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 :)

## 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. 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, 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 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 form of Map.Entry object.

**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 .

**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 null key?**

Null key is handled specially in HashMap, there are two separate method 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 retreived 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 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 form 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 array of default length e.g. 16. After some research, Java team founds 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 worst case where hash function is not distributing keys properly.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html>

Difference between ConcurrentHashMap, Hashtable and Synchronized Map in Java

**ConcurrentHashMap vs Hashtable vs Synchronized Map**

Though all three collection classes are thread-safe and can be used in multi-threaded, concurrent Java application, there is significant difference between them, which arise from the fact that how they achieve their thread-safety. Hashtable is a legacy class from JDK 1.1 itself, which uses synchronized methods to achieve thread-safety. All methods of Hashtable are synchronized which makes them quite slow due to contention if number of thread increases. Synchronized Map is also not very different than Hashtable and provides similar performance in concurrent Java programs. Only difference between Hashtable and Synchronized Map is that later is not a legacy and you can wrap any Map to create it's synchronized version by using Collections.synchronizedMap() method. On the other hand, ConcurrentHashMap is especially designed for concurrent use i.e. more than one thread. By default it simultaneously allows 16 threads to read and write from Map without any external synchronization. It is also very scalable because of stripped locking technique used in [internal implementation of ConcurrentHashMap](http://javarevisited.blogspot.sg/2013/02/concurrenthashmap-in-java-example-tutorial-working.html) class. Unlike Hashtable and Synchronized Map, it never locks whole Map, instead it divides the map in segments and locking is done on those. Though it perform better if number of reader threads is greater than number of writer threads.

To be frank, Collections classes are heart of Java API though I feel using them judiciously is an art. Its my personal experience where I have improved performance of Java application by using ArrayList where legacy codes were unnecessarily using Vector etc. Prior Java 5, One of the major drawback of Java Collection framework was lack of scalability. In multi-threaded Java application synchronized collection classes like Hashtable and Vector quickly becomes bottleneck; to address scalability JDK 1.5 introduces some good concurrent collections which is highly efficient for high volume, low latency system electronic trading systems In general those are backbone for Concurrent fast access of stored data. In this tutorial we will look on ConcurrentHashMap, Hashtable, HashMap and synchronized Map and see difference between ConcurrentHashMap and Hashtable and synchronized Map in Java. We have already discussed some key [difference between HashMap and Hashtable in Java](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html) in this blog and those will also help you to answer this question during interviews.

## Why need ConcurrentHashMap and CopyOnWriteArrayList

The synchronized collections classes, Hashtable and Vector, and the synchronized wrapper classes, Collections.synchronizedMap() and Collections.synchronizedList(), provide a basic conditionally thread-safe implementation of Map and List. However, several factors make them unsuitable for use in highly concurrent applications for example their single collection-wide lock is an impediment to scalability and it often becomes necessary to lock a collection for a considerable time during iteration to prevent [ConcurrentModificationException](http://javarevisited.blogspot.sg/2011/10/java-iterator-tutorial-example-list.html).

ConcurrentHashMap and CopyOnWriteArrayList implementations provide much higher concurrency while preserving thread safety, with some minor compromises in their promises to callers. ConcurrentHashMap and CopyOnWriteArrayList are not necessarily useful everywhere you might use HashMap or ArrayList, but are designed to optimize specific common situations. Many concurrent applications will benefit from their use.

### 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. This will be more clear by looking at this diagram which explains internal working of ConcurrentHashMap in Java.

### 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 synchronized Map. Here are some of common differences between ConcurrentHashMap and synchronized map in Java

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://java67.blogspot.com/2013/02/10-examples-of-hashmap-in-java-programming-tutorial.html>

10 Examples of HashMap in Java - Programming Tutorial

HashMap in Java is one of the most popular Collection class among Java programmers. After by article [How HashMap works in Java](http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html), which describes theory part of Java HashMap ,I thought to share, How to use HashMap in Java with fundamental HashMap examples, but couldn't do that and it was slipped. HashMap is a a data structure, based on hashing, which allows you to store object as key value pair, advantage of using HashMap is that, you can retrieve object on constant time i.e. O(1), if you know the key. HashMap implements Map interface and supports [Generics](http://javarevisited.blogspot.com/2011/09/generics-java-example-tutorial.html) from Java 1.5 release, which makes it type safe. There are couple of more Collections, which provides similar functionalities like HashMap, which can also be used to store key value pair. Hashtable is one of them, but Hashtable is synchronized and performs poor in single threaded environment. See [Hashtable vs HashMap](http://java67.blogspot.com/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html) for complete differences between them. Another one, relatively new is ConcurrentHashMap, which provides better performance than Hashtable in concurrent environment and should be preferred. See [difference between ConcurrentHashMap and HashMap](http://java67.blogspot.com/2012/08/difference-between-hashmap-and-concurrentHashMap-java-collection.html) for detail differences. In this Java tutorial, we will see different examples of HashMap, like adding and removing entries, iterating over Java HashMap, checking size map, finding if a key or value exists on Map and various other examples, which we used frequently.

## Java HashMap Example

Before going to see these examples, few things to note about Java HashMap.It’s not [synchronized](http://java67.blogspot.com/2013/01/difference-between-synchronized-block-vs-method-java-example.html), so don't share your HashMap among multiple threads. Another common cause of error is clearing Map and reusing it, which is perfectly valid in single threaded environment but if done in multi-threaded environment can create subtle bugs.

### Java HashMap Example 1: Create and add objects in HashMap

In first example of HashMap, we will create and add object into our Map. Always use Generics, if you are not working in Java 1.4. Following code will create HashMap with keys of type String and values of type Integer with default size and load factor.

HashMap<String, Integer> cache = new HashMap<String, Integer>();

alternatively you can create HashMap from copying data from another Map or [Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html) as shown in below example:

Hashtable<Integer, String> source = new Hashtable<Integer,String>();

HashMap<Integer, String> map = new HashMap(source);

You can also supply load factor (percentage of size, which if fulled trigger resize of HashMap) and initialiCapacity while creating instance by using overloaded constructor provided in API. Adding elements, also called put operation, requires key and value object. Here is an example of adding key and value in Java HashMap:

map.put(21, "Twenty One");

map.put(21.0, "Twenty One"); //this will throw compiler error because 21.0 is not integer

### Java HashMap Example 2: Retrieving value from HashMap

Another basic example is retrieving value from HashMap. in order to retrieve values, we need to know key object. let's use the key inserted in last example, for getting value back from Map. get(key) method is used to get value form HashMap :

Integer key = 21;

String value = map.get(key);

System.out.println("Key: " + key +" value: "+ value);

Output: Key: 21 value: Twenty One

### Java HashMap Example 3: Iterating over HashMap

Another way to get value from HashMap is by iterating over whole Map. Sometime we do want to loop through whole map and perform operations on each key value pair, we can use Iterator for that purpose. In order to use Iterator, we first need Set of keys, which can be retrieved using map.keySet() method. By the way there are multiple ways to loop through Map in Java, see here for [4 ways to loop HashMap in Java](http://javarevisited.blogspot.com/2011/12/how-to-traverse-or-loop-hashmap-in-java.html). Here is an example of iterating over Map using java.util.Iterator :

map.put(21, "Twenty One");

map.put(31, "Thirty One");

Iterator<Integer> keySetIterator = map.keySet().iterator();

while(keySetIterator.hasNext()){

Integer key = keySetIterator.next();

System.out.println("key: " + key + " value: " + map.get(key));

}

Output:

key: 21 value: Twenty One

key: 31 value: Thirty One

### Java HashMap Example 4: Size and Clear in HashMap

Two fundamental example of HashMap, is finding out how many elements are stored in Map, known as size of Map and clearing HashMap to reuse. Java Collection API provides two convinient method called size() and clear() to perform these operation on java.util.HashMap, here is code example.

System.out.println("Size of Map: " + map.size());

map.clear(); //clears hashmap , removes all element

System.out.println("Size of Map: " + map.size());

Output:

Size of Map: 2

Size of Map: 0

You can reuse Map by clearing it, but be careful if its been shared between multiple threads without proper [synchronization](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html). Since you may need to prevent other thread from accessing map when its getting clear. I suggest not to do, until you have very good reason of doing it.

### Java HashMap Example 5 and 6: ContainsKey and ContainsValue Example

In this example of Java HashMap, we will learn how to check if Map contains a particular object as key or value. java.util.HashMap provides convenient methods like containsKey(Object key) and containsValue(Object value) which can be used to for checking existence of any key value in HashMap. here is code example :

System.out.println("Does HashMap contains 21 as key: " + map.containsKey(21));

System.out.println("Does HashMap contains 21 as value: " + map.containsValue(21));

System.out.println("Does HashMap contains Twenty One as value: " + map.containsValue("Twenty One"));

Output:

Does HashMap contains 21 as key: true

Does HashMap contains 21 as value: false

Does HashMap contains Twenty One as value: true

### Java HashMap Example 7: Checking if HashMap is empty

In this Map example, we will learn how to check if HashMap is empty in Java. There are two ways to find out if Map is empty, one is using size() method, if size is zero means Map is empty. Another way to check if HashMap is empty is using more readable isEmpty() method which returns true if Map is empty. Here is code example :

boolean isEmpty = map.isEmpty();

System.out.println("Is HashMap is empty: " + isEmpty);

Output:

Is HashMap is empty: false

### Java HashMap Example 8: Removing Objects from HashMap

Another common example of Java HashMap is removing entries or mapping from Map. Java.util.HashMap provides remove(Object key) method, which accept key and removes mapping for that key.This method, returns null or the value of entry, just removed. Here is a code example of removing key value from HashMap:

Integer key = 21;

Object value = map.remove(key);

System.out.println("Following value is removed from Map: " + value);

Output:

Following value is removed from Map: Twenty One

### Java HashMap Example 9: Sorting HashMap in Java

HashMap is an unsorted Map in Java, neither key or value is sorted. If you want to sort HashMap than you can sort it based upon key or value, see [how to sort HashMap on keys and values](http://javarevisited.blogspot.com/2012/12/how-to-sort-hashmap-java-by-key-and-value.html) for full code example. Alternatively, you can use SortedMap in Java like TreeMap. TreeMap has constructor which accepts Map and can create a Map sorted on natural order of key or any custom sorting order defined by [Comparator](http://java67.blogspot.com/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html). Only thing is key should be naturally comparable and there compareTo() method shouldn't throw exception. Just to remind there is no Collections.sort() method defined for Map is only for List and it’s implementation e.g. [ArrayList or LinkedList](http://java67.blogspot.com/2012/12/difference-between-arraylist-vs-LinkedList-java.html). So any sorting for Map require SortedMap or custom code for sorting on either key or value. here is code example of sorting HashMap in Java by using [TreeMap](http://javarevisited.blogspot.com/2011/12/treemap-java-tutorial-example-program.html) in natural order of keys:

map.put(21, "Twenty One");

map.put(31, "Thirty One");

map.put(41, "Thirty One");

System.out.println("Unsorted HashMap: " + map);

TreeMap sortedHashMap = new TreeMap(map);

System.out.println("Sorted HashMap: " + sortedHashMap);

Output:

Unsorted HashMap: {21=Twenty One, 41=Thirty One, 31=Thirty One}

Sorted HashMap: {21=Twenty One, 31=Thirty One, 41=Thirty One}

### Java HashMap Example 10: Synchronized HashMap in Java

You need to synchronize HashMap if you want to use it in multi-threaded environment. If you are running on Java 1.5 and above consider using ConcurrentHashMap in place of synchronized HashMap because it provide better concurrency. If your project is still on JDK 1.4 than you got to use either Hashtable or synchronized Map. Collections.synchronizedMap(map) is used to synchronize HashMap in Java. See [here](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html) for full code example. This method returns a thread-safe version of Map and all map operation is serialized.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html

How get method of HashMap or Hashtable works internally in Java

In this article, I am revisiting couple of interesting question related to internal working of HashMap in Java, mostly asked to senior Java developers, ranging from 4 to 6 and upto 8 years of experience. I did cover lot of these questions from HashMap, ranging from thread-safety to race conditions, in my post about [internal working of Java HashMap](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html), but I thought to revisit two of those questions, *How get method of HashMap or Hashtable works internally in Java* and *What happens if two different keys return same hashCode*, how do you return value form HashMap in that case. These are the question, which is highly popular in investment banking domain, and preferred choice of interviewer, while interviewing experienced Java professional. If these questions are still being asked, it means not many are answering them in the clarity and confidence they are looking for. This motivates me to revisit these questions again. On a side note, in order to understand these questions, you should have good knowledge of [equals and hashcode method](http://java67.blogspot.com/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html) as well. At least you should know that :

1) Two unequal object may return same hashcode.

2) When two objects are equal by equals(), they must have same hashcode.

## How get method of Hashtable works in Java

I also suggest reading equals and hashcode chapters from [Effective Java](http://www.amazon.com/gp/product/0321356683/ref=as_li_ss_tl?ie=UTF8&tag=javamysqlanta-20&linkCode=as2&camp=1789&creative=390957&creativeASIN=0321356683) book to filling your gaps. By the way due to f this reason, it's a requirement for key object in HashMap to implement both equals() and hashCode(), in fact that is also one of the popular question among experienced Java programmers, asked as what is requirement for an object to be used as key in hash based collection e.g. HashMap, Hashtable and ConcurrentHashMap. Another optional, but worth mentioning requirement of keys in hash based collection is being [Immutable object](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html). Keeping this knowledge along with general knowledge of hashing algorithm, which revolves around hash function, let's see those HashMap questions again :

**1) How does get(Key key) method works internally in HashMap, and Hashtable in Java?**

Here are steps, which happens, when you call get() method with key object to retrieve corresponding value from hash based collection

a) Key.hashCode() method is used to find the bucket location in backing array. (Remember HashMap is backed by array in Java) Though hashcode() is not used directly, but they are passed to internal hash() function.

b) In backing array or better known as bucket, key and values are stored in form of a nested class called Entry. If there is only one Entry at bucket location, than value from that entry is returned. Pretty straightforward right?

Things get little tricky, when Interviewer ask second question, **What happens if two keys has same hashCode?** If multiple keys has same hashCode, than during put() operation collision had occurred, which means multiple Entry object stored in a bucket location. Each Entry keep track of another Entry, forming a [linked list data structure](http://javarevisited.blogspot.com/2013/05/find-if-linked-list-contains-loops-cycle-cyclic-circular-check.html) there. Now, if we need to retrieve value object in this situation, following steps will be followed :

1) Call hashCode() method of key to find bucket location.

2) Traverse thought linked list, comparing keys in each entries using keys.equals() until it return true.

So, we use equals() method of key object to find correct entry and than return value from that. Remember key.equals() method, and this is what Interviewer want to know. I have seen many programmer mentioning value.equals(), which may be due to interview nervousness, but that’s incorrect. Since you don't have value object passed to get() method, there is no question of calling equals and hashCode method on value object.

That's all on these two HashMap questions guys. Remember to mention about key.hashCode() and key.equals(), whenever some one ask how get method of HashMap or Hashtable works in Java. Value object is not used, it's just exist in Entry, so that get can return it.

Related **Java HashMap tutorials** you may like

[Difference between ConcurrentHashMap and HashMap in Java](http://java67.blogspot.com/2012/08/difference-between-hashmap-and-concurrentHashMap-java-collection.html)

[5 Difference between HashMap and Hashtable in Java](http://java67.blogspot.com/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html)

[10 ways to use HashMap in Java](http://java67.blogspot.com/2013/02/10-examples-of-hashmap-in-java-programming-tutorial.html)

[How to sort HashMap in Java by keys](http://javarevisited.blogspot.com/2012/12/how-to-sort-hashmap-java-by-key-and-value.html)

[How to get key from HashMap by passing values](http://javarevisited.blogspot.com/2013/02/how-to-get-key-from-value-in-hashtable.html)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://javarevisited.blogspot.sg/2010/10/difference-between-hashmap-and.html>

What is difference between HashMap and Hashtable in Java?

**HashMap vs Hashtable in Java**

Though both Hashtable and HashMap are data-structure based upon hashing and implementation of Map interface, main difference between them is that HashMap is not thread-safe but Hashtable is thread-safe. Which means you cannot use HashMap in multi-threaded Java application without external synchronization. Another difference is HashMap allows one null key and null values but Hashtable doesn't allow null key or values. Also thread-safety of hash table is achieved using internal synchronization, which makes it slower than HashMap. By the way *difference between HashMap and Hashtable in Java* is one of the frequently asked in core Java interviews to check whether candidate understand correct usage of collection classes and aware of alternative solutions available. Along with [How HashMap internally works in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html) and [ArrayList vs Vector](http://javarevisited.blogspot.sg/2011/09/difference-vector-vs-arraylist-in-java.html), this  is one of the oldest question from Collection framework in Java. Hashtable is a legacy Collection class and it's there in Java API from long time but it got re-factored to implement Map interface in Java 4 and from there Hashtable became part of Java Collection framework. Hashtable vs HashMap in Java is so popular a question that it can top any list of [Java Collection interview Question](http://javarevisited.blogspot.sg/2011/11/collection-interview-questions-answers.html). You just can't afford not to prepare HashMap vs Hashtable before going to any Java programming interview. In this Java article we will not only see some important differences between HashMap and Hashtable but also some similarities between these two collection classes. Let's first see How different they are :

## Difference between HashMap and Hashtable in Java

Both HashMap and Hashtable implements Map interface but there are some significant difference between them which is important to remember before deciding whether to use HashMap or Hashtable in Java. Some of them is thread-safety, synchronization and speed. here are those differences :

1.The HashMap class is roughly equivalent to Hashtable, except that it is non synchronized and permits nulls. (HashMap allows null values as key and value whereas [Hashtable](http://javarevisited.blogspot.sg/2012/01/java-hashtable-example-tutorial-code.html) doesn't allow nulls).

2. One of the major **differences between HashMap and Hashtable** is that HashMap is non synchronized whereas Hashtable is synchronized, which means Hashtable is thread-safe and can be shared between multiple threads but HashMap can not be shared between multiple threads without proper synchronization. Java 5 introduces [ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html) which is an alternative of Hashtable and provides better scalability than Hashtable in Java.

3. Another significant difference between HashMap vs Hashtable is that Iterator in the HashMap is  a fail-fast iterator  while the enumerator for the Hashtable is not 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. This is also an important [difference between Enumeration and Iterator in Java](http://javarevisited.blogspot.sg/2010/10/what-is-difference-between-enumeration.html).

4. One more notable *difference between Hashtable and HashMap* is that because of thread-safety and synchronization Hashtable is much slower than HashMap if used in Single threaded environment. So if you don't need synchronization and HashMap is only used by one thread, it out perform Hashtable in Java.

5. HashMap does not guarantee that the order of the map will remain constant over time.

### HashMap and Hashtable : note on Some Important Terms

1)Synchronized means only one Thread can modify a hash table at one point of time. Basically, it means that any thread before performing an update on a Hashtable will have to acquire a lock on the object while others will wait for lock to be released.

2) Fail-safe is relevant from the context of iterators. If an [Iterator or ListIterator](http://javarevisited.blogspot.sg/2011/10/java-iterator-tutorial-example-list.html) has been created on a collection object and some other thread tries to modify the collection object "structurally", a concurrent modification exception will be thrown. It is possible for other threads though to invoke "set" method since it doesn't modify the collection "structurally". However, if prior to calling "set", the collection has been modified structurally, "IllegalArgumentException" will be thrown.

3) Structurally modification means deleting or inserting element which could effectively change the structure of map.

HashMap can be synchronized by

Map m = Collections.synchronizeMap(hashMap);

In Summary there are significant *differences between Hashtable and HashMap in Jav*a e.g. thread-safety and speed and based upon that only use Hashtable if you absolutely need thread-safety, if you are running Java 5 consider using ConcurrentHashMap in Java.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*