1.    Marshalling – Convert a Java object into a XML file.

2.    Unmarshalling – Convert XML content into a Java Object.

<http://javarevisited.blogspot.sg/2011/11/collection-interview-questions-answers.html>

[**http://howtodoinjava.com/category/core-java/collections/**](http://howtodoinjava.com/category/core-java/collections/)

[**http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/**](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/)

##### Explain Collection’s hierarchy?

**Collection: List, Queue, Set → SortedSet**

**Map → SortedMap**

##### How to convert an array of String to arraylist?

String[] words = {"ace", "boom", "crew", "dog", "eon"};

List wordList = Arrays.asList(words);

##### How to sort list in reverse order? à Collections.reverse(list);

Main classes implementing Map interface are: **HashMap, Hashtable, EnumMap, IdentityHashMap, LinkedHashMap and Properties.**

##### What are IdentityHashMap and WeakHashMap?

**IdentityHashMap** is similar to HashMap except that **it uses reference equality when comparing elements**. IdentityHashMap class is not a widely used Map implementation. While this class implements the Map interface, it intentionally violates Map’s general contract, which mandates the use of the equals() method when comparing objects. IdentityHashMap is designed for use only in the rare cases wherein reference-equality semantics are required.

**WeakHashMap (It is used for cahching)** is an implementation of the Map interface **that stores only weak references to its keys**. Storing only weak references allows a key-value pair to be garbage collected when its key is no longer referenced outside of the WeakHashMap. This class is intended primarily for use with key objects whose equals methods test for object identity using the == operator. Once such a key is discarded it can never be recreated, so it is impossible to do a look-up of that key in a WeakHashMap at some later time and be surprised that its entry has been removed.

##### What are different Collection views provided by Map interface?

Map interface provides 3 views of key-values pairs stored in it:

·         key set view

·         value set view

·         entry set view

All the views can be navigated using iterators.

##### When to use HashMap or TreeMap?

TreeMap is special form of HashMap. **It maintains the ordering of keys** which is missing in HashMap class. This ordering is **by default “natural ordering”**. The default ordering can be override by providing an instance of Comparator class, whose compare method will be used to maintain ordering of keys.

Please note that **all keys inserted into the map must implement the Comparable interface** (this is necessary to decide the ordering). Furthermore, all such keys must be mutually comparable: k1.compareTo(k2) must not throw a ClassCastException for any keys k1 and k2 in the map. If the user attempts to put a key into the map that violates this constraint (for example, the user attempts to put a string key into a map whose keys are integers), the put(Object key, Object value) call will throw a ClassCastException.

##### Difference between Iterator and Enumeration?

Iterators differ from enumerations in three ways:

·         Iterators allow the caller to remove elements from the underlying collection during the iteration with its remove() method. You can not add/remove elements from a collection when using enumerator.

·         Enumeration is available in legacy classes i.e Vector/Stack etc. whereas Iterator is available in all modern collection classes.

·         Another minor difference is that Iterator has improved method names e.g. Enumeration.hasMoreElement() has become Iterator.hasNext(), Enumeration.nextElement() has become Iterator.next() etc.

##### Difference between HashMap and HashSet?

HashMap is collection of key-value pairs whereas HashSet is un-ordered collection of unique elements. That’s it. No need to describe further.

##### Difference between Iterator and ListIterator?

There are three Differences are there:

·         We can use Iterator to traverse Set and List and also Map type of Objects. But List Iterator can be used to traverse for List type Objects, but not for Set type of Objects.

·         By using Iterator we can retrieve the elements from Collection Object in forward direction only whereas List Iterator, which allows you to traverse in either directions using hasPrevious() and previous() methods.

·         ListIterator allows you modify the list using add() remove() methods. Using Iterator you can not add, only remove the elements.

##### Difference between ArrayList and LinkedList?

·         LinkedList store elements within a doubly-linked list data structure. ArrayList store elements within a dynamically resizing array.

·         LinkedList allows for constant-time insertions or removals, but only sequential access of elements. In other words, you can walk the list forwards or backwards, but grabbing an element in the middle takes time proportional to the size of the list. ArrayLists, on the other hand, allow random access, so you can grab any element in constant time. But adding or removing from anywhere but the end requires shifting all the latter elements over, either to make an opening or fill the gap.

·         LinkedList has more memory overhead than ArrayList because in ArrayList each index only holds actual object (data) but in case of LinkedList each node holds both data and address of next and previous node.

##### What do you understand by iterator fail-fast property?

**Fail-fast Iterators fail as soon as they realized that structure of Collection has been changed since iteration has begun**. Structural changes means adding, removing or updating any element from collection while one thread is Iterating over that collection.

Fail-fast behavior is implemented by keeping a modification count and if iteration thread realizes the change in modification count it throws ConcurrentModificationException.

Iterator fail-fast property checks for any modification in the structure of the underlying collection everytime we try to get the next element. If there are any modifications found, it throws ConcurrentModificationException. All the implementations of Iterator in Collection classes are fail-fast by design except the concurrent collection classes like ConcurrentHashMap and CopyOnWriteArrayList.

##### 32) What is difference between fail-fast and fail-safe?

You have understood fail-fast in previous question. **Fail-safe iterators** are just opposite to fail-fast. **They never fail if you modify the underlying collection on which they are iterating**, because they work on clone of Collection instead of original collection and that’s why they are called as fail-safe iterator.

Iterator of CopyOnWriteArrayList is an example of fail-safe Iterator also iterator written by ConcurrentHashMap keySet is also fail-safe iterator and never throw ConcurrentModificationException.

Iterator fail-safe property work with the clone of underlying collection, hence it’s not affected by any modification in the collection. By design, all the collection classes in java.util package are fail-fast whereas collection classes in java.util.concurrent are fail-safe. Fail-fast iterators throw ConcurrentModificationException whereas fail-safe iterator never throws ConcurrentModificationException. Check this post for [CopyOnWriteArrayList Example](http://www.journaldev.com/1289/java-arraylist-vs-copyonwritearraylist-and-exploring-iterator).

##### What is BlockingQueue?

**A Queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element.**

BlockingQueue methods come in four forms: one throws an exception, the second returns a special value (either null or false, depending on the operation), the third blocks the current thread indefinitely until the operation can succeed, and the fourth blocks for only a given maximum time limit before giving up.

##### What is Comparable and Comparator interface?

In java. all collection which have feature of automatic sorting, uses compare methods to ensure the correct sorting of elements. For example classes which use sorting are TreeSet, TreeMap etc.

**To sort the data elements a class needs to implement Comparator or Comparable interface**. That’s why all Wrapper classes like Integer,Double and String class implements Comparable interface.

**Comparable helps in preserving default natural sorting, whereas Comparator helps in sorting the elements in some special required sorting pattern.** The instance of comparator if passed usually as collection’s constructor argument in supporting collections.

**HashMap vs Hashtable in** **Java**

Both HashMap and Hashtable implements Map interface but there are some significant difference

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](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) 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. 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.

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

1)Synchronized means only one [Thread](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) 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](http://javarevisited.blogspot.sg/2012/02/why-wait-notify-and-notifyall-is.html) to be released.

**Difference between ConcurrentHashMap and Collections.synchronizedMap and Hashtable in Java**

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.

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

ConcurrentHashMap is better suited for situation where you have multiple readers and one

Writer or fewer writers since Map gets locked only during write operation. If you have equaled number of reader and writer than [ConcurrentHashMap](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) will perform in line of hashtable or synchronized hashMap.

<http://howtodoinjava.com/2013/06/14/popular-hashmap-and-concurrenthashmap-interview-questions/>

To better visualize the ConcurrentHashMap, let it consider as a group of HashMaps. To get and put key-value pairs from hashmap, you have to calculate the hashcode and look for correct bucket location in array of Collection.Entry. Rest you have read on previous related article on how hashmap works

In concurrentHashMap, the **difference lies in internal structure to store these key-value pairs**. ConcurrentHashMap has an addition concept of segments. It will be easier to understand it you think of one segment equal to one HashMap [conceptually]. A concurrentHashMap is divided into number of segments [default 16] on initialization. ConcurrentHashMap allows similar number (16) of threads to access these segments concurrently so that each thread work on a specific segment during high concurrency.

This way, if when your key-value pair is stored in segment 10; code does not need to block other 15 segments additionally. This structure provides a very high level of concurrency.

*ConcurrentHashMap Internal Structure*

In other words, **ConcurrentHashMap uses a multitude of locks, each lock controls one segment of the map**. When setting data in a particular segment, the lock for that segment is obtained. So essentially **update operations are synchronized**.

**When getting data, a volatile read is used** without any synchronization. If the volatile read results in a miss, then the lock for that segment is obtained and entry is again searched in synchronized block.

**Difference between ArrayList and Vector in Java**

1) Vector and ArrayList are index based and backed up by an array internally.

2) Both ArrayList and Vector maintains the insertion order of element. Means you can assume that you will get the object in the order you have inserted if you iterate over ArrayList or Vector.

3) Both [Iterator](http://javarevisited.blogspot.com/2010/10/what-is-difference-between-enumeration.html) and ListIterator returned by ArrayList and Vector are [fail-fast](http://javarevisited.blogspot.sg/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html).

4) ArrayList and Vector also allows null and duplicates.

First and foremost difference between Vector and ArrayList is that Vector is synchronized and ArrayList is not, ArrayList is way faster than Vector.

Whenever Vector crossed the threshold specified it increases itself by value specified in capacityIncrement field while you can increase size of ArrayList by calling ensureCapacity () method.

**Difference between List and Set in Java Collection**

1) Fundamental difference between List and Set in Java is allowing duplicate elements. List in Java allows duplicates while Set doesn't allow any duplicate. If you insert duplicate in Set it will replace the older value. Any implementation of Set in Java will only contains unique elements.

2) Another significant difference between List and Set in Java is order. List is an Ordered Collection while Set is an unordered Collection. List maintains insertion order of elements, means any element which is inserted before will go on lower index than any element which is inserted after. Set in Java doesn't  maintain any order. Though Set provide another alternative called SortedSet which can store Setelements in specific Sorting order defined by [Comparable and Comparator](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html) methods of Objects stored in Set.

**What is difference between Synchronized Collection and ConcurrentCollection?**

Java5 has added several new ConcurrentCollection classes e.g. **ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue** etc, which has made Interview questions on Java Collection even trickier. Java Also provided way to get Synchronized copy of collection e.g. ArrayList, HashMap by using Collections.**synchronizedMap**() Utility function.**One Significant difference is that ConccurentCollections has better performance than synchronized Collection because they lock only a portion of Map to achieve concurrency and Synchronization.**

1. Marshalling – Convert a Java object into a XML file.

2.    Unmarshalling – Convert XML content into a Java Object.

<http://javarevisited.blogspot.sg/2011/11/collection-interview-questions-answers.html>

**HashMap vs Hashtable in** **Java**

Both HashMap and Hashtable implements Map interface but there are some significant difference

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](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) 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. 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.

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

1)Synchronized means only one [Thread](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) 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](http://javarevisited.blogspot.sg/2012/02/why-wait-notify-and-notifyall-is.html) to be released.

**Difference between ConcurrentHashMap and Collections.synchronizedMap and Hashtable in Java**

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.

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

ConcurrentHashMap is better suited for situation where you have multiple readers and one

Writer or fewer writers since Map gets locked only during write operation. If you have equaled number of reader and writer than [ConcurrentHashMap](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) will perform in line of hashtable or synchronized hashMap.

**Difference between ArrayList and Vector in Java**

1) Vector and ArrayList are index based and backed up by an array internally.

2) Both ArrayList and Vector maintains the insertion order of element. Means you can assume that you will get the object in the order you have inserted if you iterate over ArrayList or Vector.

3) Both [Iterator](http://javarevisited.blogspot.com/2010/10/what-is-difference-between-enumeration.html) and ListIterator returned by ArrayList and Vector are [fail-fast](http://javarevisited.blogspot.sg/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html).

4) ArrayList and Vector also allows null and duplicates.

First and foremost difference between Vector and ArrayList is that Vector is synchronized and ArrayList is not, ArrayList is way faster than Vector.

Whenever Vector crossed the threshold specified it increases itself by value specified in capacityIncrement field while you can increase size of ArrayList by calling ensureCapacity () method.

**Difference between List and Set in Java Collection**

1) Fundamental difference between List and Set in Java is allowing duplicate elements. List in Java allows duplicates while Set doesn't allow any duplicate. If you insert duplicate in Set it will replace the older value. Any implementation of Set in Java will only contains unique elements.

2) Another significant difference between List and Set in Java is order. List is an Ordered Collection while Set is an unordered Collection. List maintains insertion order of elements, means any element which is inserted before will go on lower index than any element which is inserted after. Set in Java doesn't  maintain any order. Though Set provide another alternative called SortedSet which can store Setelements in specific Sorting order defined by [Comparable and Comparator](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html) methods of Objects stored in Set.

**What is difference between Synchronized Collection and ConcurrentCollection?**

Java5 has added several new ConcurrentCollection classes e.g. **ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue** etc, which has made Interview questions on Java Collection even trickier. Java Also provided way to get Synchronized copy of collection e.g. ArrayList, HashMap by using Collections.**synchronizedMap**() Utility function.**One Significant difference is that ConccurentCollections has better performance than synchronized Collection because they lock only a portion of Map to achieve concurrency and Synchronization.**

**HashCode and Equals method in Java object – A pragmatic concept**

Java.lang.Object has methods called hasCode() and equals(). In some case these methods are overridden to perform certain purpose. it is necessary to override  both method.

**hashCode():**this method provides the has code of an object. the default implementation of hashCode() provided by Object is derived by mapping the **memory address to an integer value**.

public native int hashCode(); -- It indicates that hashCode is the native implementation

hashCode() method is used to get a unique integer for given object. This integer is used for determining the bucket location, when this object needs to be stored in some HashTable like data structure. By default, Object’s hashCode() method returns and integer representation of memory address where object is stored.

**equals()**

This particular method is used to make equal comparison between two objects. There are two types of comparisons in Java. One is using “= =” operator and another is “equals()”. .equals()” refers to equivalence relations. So in broad sense you say that two objects are equivalent they satisfy the “equals()” condition.

This method checks if some other object passed to it as an argument is *equal* to the object on which this method is invoked.

**why it is necessary to override this method.** :

There is one reason that if want to compare two objects based upon the equals() method. Although in a very simple class like “Emp”, you can achieve without overriding hashCode() method. But if you do this , you are going to violate the contract for the methods hashCode() and hashCode() of the object class. The similar case is for the method equals(). So funcational point is that if want to compare two objects based upon the equals() method you have to override both hashCode() and equals() methods.

How HashMap works in Java

<http://howtodoinjava.com/2012/10/09/how-hashmap-works-in-java/>

It is works on “On principle of Hashing“.

What is Hashing

Hashing in its simplest form, is a way to assigning a unique code for any variable/object after applying any formula/algorithm on its properties. A true Hashing function must follow this rule:

Hash function should return the same hash code each and every time, when function is applied on same or equal objects. In other words, two equal objects must produce same hash code consistently.

### What put() method actually does

First of all, key object is checked for null. If key is null, value is stored in table[0] position. Because hash code for null is always 0.

- Then on next step, a hash value is calculated using key’s hash code by calling its hashCode() method. This hash value is used to calculate index in array for storing Entry object. JDK designers well assumed that there might be some poorly written hashCode() functions that can return very high or low hash code value. To solve this issue, they introduced another hash() function, and passed the object’s hash code to this hash() function to bring hash value in range of array index size.

Now indexFor(hash, table.length) function is called to calculate exact index position for storing the Entry object.

- Here comes the main part. Now, as we know that two unequal objects can have same hash code value, how two different objects will be stored in same array location [called bucket].

Answer is LinkedList. If you remember, Entry class had an attribute “next”. This attribute always points to next object in chain. This is exactly the behavior of LinkedList.

### How get() methods works internally

Now we have got the idea, how key-value pairs are stored in HashMap. Next big question is : what happens when an object is passed in get method of HashMap? How the value object is determined?

Answer we already should know that the way key uniqueness is determined in put() method , same logic is applied in get() method also. The moment HashMap identify exact match for the key object passed as argument, it simply returns the value object stored in current Entry object.

If no match is found, get() method returns null.

### Key Notes

1.    Data structure to store Entry objects is an array named **table** of type Entry.

2.    A particular index location in array is referred as bucket, because it can hold the first element of a LinkedList of Entry objects.

3.    Key object’s hashCode() is required to calculate the index location of Entry object.

4.    Key object’s equals() method is used to maintain uniqueness of Keys in map.

5.    Value object’s hashCode() and equals() method are not used in HashMap’s get() and put() methods.

6.    Hash code for null keys is always zero, and such Entry object is always stored in zero index in Entry[].

"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 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.

HashMap stores key-value pair in Map.Entry static nested class implementation. HashMap works on hashing algorithm and uses hashCode() and equals() method in put and getmethods.When we call putmethod by passing key-value pair, HashMap uses Key hashCode() with hashing to find out the index to store the key-value pair. The Entry is stored in the LinkedList, so if there are already existing entry, it uses equals() method to check if the passed key already exists, if yes it overwrites the value else it creates a new entry and store this key-value Entry.When we call get method by passing Key, again it uses the hashCode() to find the index in the array and then use equals() method to find the correct Entry and return it’s value. Below image will explain these detail clearly.

**1.**    **What is the benefit of Generics in Collections Framework?**

Java 1.5 came with Generics and all collection interfaces and implementations use it heavily. Generics allow us to provide the type of Object that a collection can contain, so if you try to add any element of other type it throws compile time error. This avoids ClassCastException at Runtime because you will get the error at compilation. Also Generics make code clean since we don’t need to use casting and instanceof operator. It also adds up to runtime benefit because the bytecode instructions that do type checking are not generated.

-----------------------------

# [How HashMap works in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.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?**

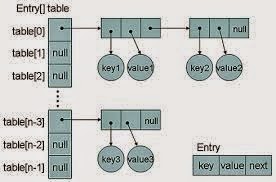
Now from here onwards real confusion starts, Some time candidate will say that since hashcode is equal, both objects are equal and HashMap  will throw exception or not store them again etc, Then you might want to remind them about [equals() and hashCode() contract](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html)that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

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

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

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

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

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 .

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.

Read more: <http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html#ixzz3KTriVtD7>