• If a class overrides **equals()**, it must override **hashCode()**.

• If 2 objects are equal, then their hashCode values must be equal as well.

• If 2 objects have same hashCode() then objects may/may not be same.

• If a field is not used in equals(), then it must not be used in hashCode().

• hashCode() is a candidate for caching to enhance performance.

**How HashMap works in Java**  
How HashMap works in Java or sometime how get method work in HashMap is common interview questions now days. Almost everybody who worked in Java knows what hashMap is, where to use hashMap or difference between hashtable and HashMap then why this interview questionbecomes so special? Because of the breadth and depth this question offers. It has become very popular java interview question in almost any senior or mid-senior level java interviews.  
  
Questions start with simple statement   
  
**"Have you** **used HashMap before" or "What is HashMap? Why do we use it “**   
Almost everybody answers this with yes and then interviewee keep talking about common facts about hashMap like [hashMapaccpt null while hashtabledoesn't](http://www.blogger.com/),[HashMap is not synchronized](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html), hashMap is fast and so on along with basics like its stores key and value pairs etc.  
This shows that person has used hashMap and quite familier with the funtionalitiesHashMap offers but interview takes a sharp turn from here and next set of follow up questions gets more detailed about fundamentals involved in hashmap. Interview here you and come 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; I can find it out in Google at any time etc.  
But some interviewee definitely answer this and will say **"HashMap works onprinciple** **of hashing**, we have put () and get () method for storing and retrieving data from hashMap. When we pass an object to put () method to store it on hashMap, hashMap implementation calls  
hashcode() method hashMap key object and by applying that hashcode on its own hashing funtion it identifies a bucket location for storing value object , important part here is HashMap stores both key+value 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 HashMap . This answer is very much acceptable and does make sense that interviewee has fair bit of knowledge how hashing works and how HashMap works in Java.  
But this is just start of story and going forward when depth increases a little bit and when you put interviewee on scenarios every java developers faced day by day basis. So next question would be more likely about collision detection and collision resolution in Java HashMape.g   
  
**"What will happen if two different objects have same hashcode?”**  
Now from here confusion starts some time interviewer will say that since Hashcode is equal objects are equal and HashMap will throw exception or not store it again etc. then you might want to remind them aobut equals and hashCode() contract that two unequal object in Java very much can have equal hashcode. Some will give up at this point and some will move ahead and say "Since hashcode () is same, bucket location would be same and collision occurs in hashMap, Since HashMap use a linked list to store in bucket, value object will be stored in next node of linked list." great this answer make sense to me though there could be some other collision resolution methods available this is simplest and HashMap does follow this.  
But story does not end here and final questions interviewer ask like   
  
**"How will you retreive if two different objects have same hashcode?”**   
 Hmmmmmmmmmmmmm  
Interviewee will say we will call get() method and then HashMap uses keys hashcode to find out bucket location and retreives object but then you need to remind him that there are two objects are stored in same bucket , so they will say about traversal in linked list until we find the value object , then you ask **how do you identify vlaue object because you don't value object to compare** ,So until they know that HashMap stores both Key and Value in linked list node 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 linked list 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 () and equals ()** and keys and values object in 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.  
Some good developer point out here that using immutable, final object with proper equals () and hashcode () implementation would act as perfect Java HashMap keys and**improve performance of Java hashMap by reducing collision**. I*mmutablity also allows caching there hashcode of different keys* which makes overall retreival process very fast and suggest that String and various wrapper classes e.g Integer provided by Java Collection API are very good HashMap keys.  
  
Now if you clear all this java hashmap interview question 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%. Java Hashmap does that by creating another new bucket array of size twice of previous size of hashmap, and then start putting every old element into that new bucket array and 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 exists while resizing hashmap in Java, if two thread at the same time found that now Java 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 :)   
  
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 () method and there importance?*  
*Benefit of immutable object?*  
*race condition on hashmap in Java*  
*Resizing of Java HashMap*  
  
Just to summararize here are the answers which does makes sense for above questions  
  
**How HashMAp works in Java**  
HashMap works on principle of hashing, we have put () and get () method for storing and retrieving object form hashMap.When we pass an both key and value to put() method to store on HashMap, it uses key object hashcode() method to calculate hashcode and they by applying hashing on that hashcode it identifies bucket location for storing value object.  
While retrieving it uses key object equals method to find out correct key value pair and return value object associated with that key. HashMap uses linked list in case of collision and object will be stored in next node of linked list.  
Also hashMap stores both key+value tuple in every node of linked list.  
  
**What will happen if two different HashMap key objects have same hashcode?**  
They will be stored in same bucket but no next node of linked list. And keys equals () method will be used to identify correct key value pair in HashMap.  
  
In terms of usage 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 a lot HashMap comes as very handy there.

Collections classes are heart of java API though I feel using them judiuously is an art.its my personal experience where I have improved performance byusing ArrayList where legacy codes are unnecesarily used Vectoretc.  
  
JDK 1.5 introduce some good concurrent collections which is highly efficient for high volume , low latency system.  
  
The synchronized collections classes, Hashtable and Vector, and the synchronizedwrapper 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 concurrentapplications -- 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 ConcurrentModificationExceptions.  
  
The 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.  
  
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 ConcurrentHashMapindroduced 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.

///////////////////

questionoftenly asked in interview to check whether candidate understand correctusage of collection classes and aware of alternative solutions available.

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 doesn't allow nulls).

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

3. HashMap is non synchronized whereas Hashtable is synchronized.

4. Iterator in the HashMap is  fail-fast  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.

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 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);

# Introduction to hashing

On this and the following pages, we explore **hashing**, a technique used (among other applications) to implement Java's common [map](http://www.javamex.com/tutorials/collections/using_4.shtml) and [set](http://www.javamex.com/tutorials/collections/using_3.shtml) classes. Hashing is designed to solve the problem of needing to **efficiently** find or store an item in a collection. For example, if we have a list of 10,000 words of English and we want to check if a given word is in the list, it would be inefficient to successively compare the word with all 10,000 items until we find a match. Hashing is a technique to make things more efficient by effectively narrowing down the search at the outset.

## What is hashing?

Hashing means using some function or algorithm to map object data to some representative integer value. This so-called **hash code** (or simply **hash**) can then be used as a way to **narrow down our search** when looking for the item in the map.

## How hashing works

Purely as an example to help us grasp the concept, let's suppose that we want to map a list of string keys to string values (for example, map a list of countries to their capital cities). So let's say we want to store the data in Table 1 in the map.