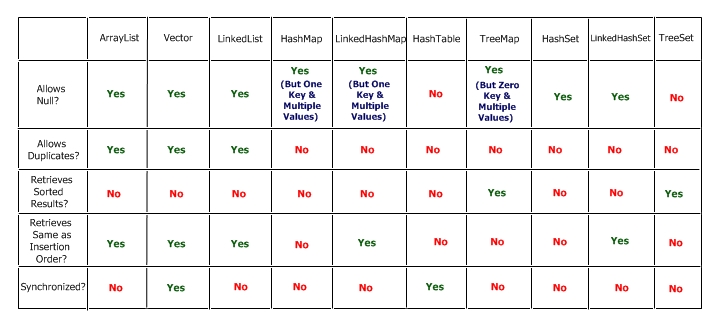
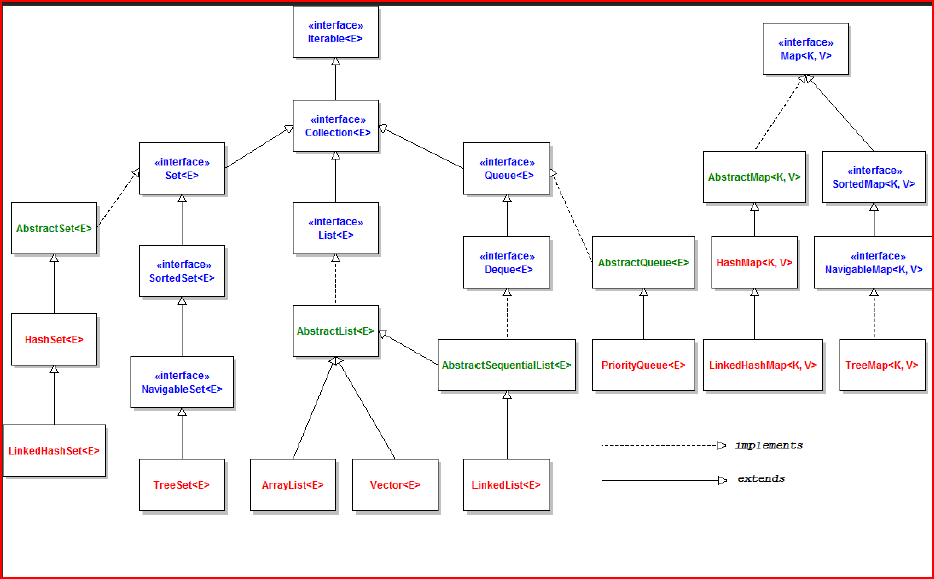
JAVA COLLECTIONS FRAMEWORK QUESTIONS AND ANSWERS

1. [Collection\_Comparison](#Collection_Comparison)
2. [Collection\_hierarchy](#Collection_hierarchy)
3. [Diff\_Enumeration\_Iterator](#Diff_Enumeration_Iterator)
4. [Convert\_List\_Set](#Convert_List_Set)
5. [How\_HashMap\_works](#How_HashMap_works)
6. [same\_hascode\_in\_hashmap](#same_hascode_in_hashmap)
7. [CopyOnWriteArrayList](#CopyOnWriteArrayList)
8. Conncurent SEt
9. [Convert HashSet to TreeSet](#Convert_List_Set)
10. [Difference between ArrayList Linked List](#diff_arraylist_linkedlist)
11. [Comparator vs Comparable](#comparable_comparator)

**Comparison Chart between different Collection**  
   




# [Difference between Java Enumeration and Iterator](http://stackoverflow.com/questions/948194/difference-between-java-enumeration-and-iterator)

Looking at the Java API Specification for the [Iterator](http://java.sun.com/javase/6/docs/api/java/util/Iterator.html) interface, there is an explanation of the differences between [Enumeration](http://java.sun.com/javase/6/docs/api/java/util/Enumeration.html):

Iterators differ from enumerations in two ways:

* Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
* Method names have been improved.

The bottom line is, both Enumeration and Iterator will give successive elements, but Iterator is improved in such a way so the method names are shorter, and has an additional remove method. Here is a side-by-side comparison:

Enumeration Iterator

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

hasMoreElement() hasNext()

nextElement() next()

N/A remove()

As also mentioned in the Java API Specifications, for newer programs, Iterator should be preferred over Enumeration, as "Iterator takes the place of Enumeration in the Java collections framework." (From the [Iterator](http://java.sun.com/javase/6/docs/api/java/util/Iterator.html) specifications.)

Iterators are **fail-fast** . i.e. when one thread changes the collection by add / remove operations , while another thread is traversing it through an Iterator using hasNext() or next() method, the iterator fails quickly by throwing ConcurrentModificationException . The fail-fast behavior of iterators can be used only to detect bugs. The Enumerations returned by the methods of classes like Hashtable, Vector are not fail-fast that is achieved by synchronizing the block of code inside the nextElement() method that locks the current Vector object which costs lots of time.

ArrayList synchronization:

1. Using CopyOnWriteArrayList
2. Collections.synchronizedList()

What is CopyOnWriteArrayList in Java - Example Tutorial

**CopyOnWriteArrayList vs Array List in Java**

CopyOnWriteArrayList is a concurrent Collection class introduced in Java5 Concurrency API along with its popular cousin [ConcurrentHashMap](http://java67.blogspot.sg/2012/08/difference-between-hashmap-and-concurrentHashMap-java-collection.html) in Java.CopyOnWriteArrayList implements List interface like [ArrayList](http://javarevisited.blogspot.sg/2011/05/example-of-arraylist-in-java-tutorial.html), [Vector](http://javarevisited.blogspot.sg/2011/09/difference-vector-vs-arraylist-in-java.html)and [LinkedList](http://javarevisited.blogspot.sg/2012/02/difference-between-linkedlist-vs.html) but it’s a thread-safe collection and it achieves its [thread-safety](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html)in a slightly different way than Vector or other thread-safe collection class. As name suggest CopyOnWriteArrayList creates copy of underlying [ArrayList](http://javarevisited.blogspot.sg/2011/06/converting-array-to-arraylist-in-java.html) with every mutation operation e.g. add or set. Normally CopyOnWriteArrayList is very expensive because it involves **costly Array copy** with every write operation but its very efficient if you have a [List](http://javarevisited.blogspot.sg/2012/04/difference-between-list-and-set-in-java.html) where Iteration outnumber mutation e.g. you mostly need to [iterate the ArrayList](http://java67.blogspot.sg/2012/08/how-to-traverse-iterate-or-loop-ArrayList-in-java-example-tutorial.html) and don't modify it too often. Iterator of CopyOnWriteArrayList is [fail-safe](http://javarevisited.blogspot.sg/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) and doesn't throw ConcurrentModificationException even if underlying CopyOnWriteArrayList is modified once Iteration begins because Iterator is operating on separate copy of ArrayList. Consequently all the updates made on CopyOnWriteArrayList is not available to Iterator. In this Java Collection tutorial we will see *What is CopyOnWriteArrayList in Java*, *Difference between ArrayList and CopyOnWriteArrayList in Java* and One simple Java programexample on How to use CopyOnWriteArrayList in Java.

Difference between CopyOnWriteArrayList and ArrayList in Java.

In last section we have seen What is CopyOnWriteArrayList in Java and How it achieves [thread-safety](http://javarevisited.blogspot.sg/2012/03/simpledateformat-in-java-is-not-thread.html) by creating a separate [copy of List](http://java67.blogspot.sg/2012/07/copy-elements-from-list-to-set-in-java-collection-example.html) for each write operation. Now let's see Some **difference between ArrayList and CopyOnWriteArrayList in Java** , which is another implementation of List interface :

1) First and foremost difference between CopyOnWriteArrayList and ArrayList in Java is that CopyOnWriteArrayList is a [thread-safe collection](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html) while ArrayList is not thread-safe and cannot be used in multi-threaded environment.

2) Second difference between ArrayList and CopyOnWriteArrayList is that [Iterator of ArrayList is fail-fast](http://javarevisited.blogspot.sg/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) and throw ConcurrentModificationException once detect any modification in List once iteration begins but Iterator of CopyOnWriteArrayList is fail-safe and doesn't throw ConcurrentModificationException.

3) Third difference between CopyOnWriteArrayList vs ArrayList is that [Iterator](http://javarevisited.blogspot.sg/2011/10/java-iterator-tutorial-example-list.html) of former doesn't support remove operation while Iterator of later supports remove() operation.

**CopyOnWriteArrayList Example in Java**

[CopyOnWriteArrayList Example in Java - Difference with ArrayList ](http://javarevisited.blogspot.sg/2012/03/how-to-loop-arraylist-in-java-code.html)Here is a complete code Example of CopyOnWriteArrayList which demonstrate that [Iterator ofCopyOnWriteArrayList](http://javarevisited.blogspot.sg/2011/10/java-iterator-tutorial-example-list.html) doesn't support remove() operation.

**import** java.util.Iterator;  
**import** java.util.concurrent.CopyOnWriteArrayList;  
  
/\*\*  
 \*  
 \* Java program to demonstrate What is CopyOnWriteArrayList in Java,

 \* Iterator of CopyOnWriteArrayList  
 \* doesn’t support add, remove or any modification operation.  
 \*  
 \* @author Java67  
 \*/  
**public** **class** CopyOnWriteArrayListExample{  
  
    **public** **static** **void** main(**String** args[]) {  
        
        **CopyOnWriteArrayList**<**String**> threadSafeList = **new** **CopyOnWriteArrayList**<**String**>();  
        threadSafeList.add("Java");  
        threadSafeList.add("J2EE");  
        threadSafeList.add("Collection");  
        
        *//add, remove operator is not supported by CopyOnWriteArrayList iterator*  
        **Iterator**<**String**> failSafeIterator = threadSafeList.iterator();  
        while(failSafeIterator.hasNext()){  
            **System**.out.printf("Read from CopyOnWriteArrayList : %s %n", failSafeIterator.next());  
            failSafeIterator.remove(); *//not supported in CopyOnWriteArrayList in Java*  
        }  
    }  
}  
  
Output:  
Read from **CopyOnWriteArrayList** : Java  
Read from **CopyOnWriteArrayList** : J2EE  
Read from **CopyOnWriteArrayList** : **Collection**

If we uncomment, commented code in this [Java program](http://javarevisited.blogspot.sg/2011/11/run-java-program-from-command-prompt.html) which modifies CopyOnWriteArrayList using Iterator then we will get following [Exception](http://javarevisited.blogspot.sg/2011/12/checked-vs-unchecked-exception-in-java.html):

Read from CopyOnWriteArrayList : Java

Exception in thread "main" java.lang.UnsupportedOperationException

        at java.util.concurrent.CopyOnWriteArrayList$COWIterator.remove(CopyOnWriteArrayList.java:1004)

        at test.CollectionTest.main(CollectionTest.java:29)

Java Result: 1

That's all on What is CopyOnWriteArrayList, **Difference between CopyOnWriteArrayList and ArrayList in Java** and an Exmaple of CopyOnWriteArrayList. In Summary use CopyOnWriteArrayList if you mostly require to Iterate over list without modifying it.

**[ordering a hashset example?](http://stackoverflow.com/questions/3380312/ordering-a-hashset-example)**

If you want to convert a HashSet to a TreeSet, then do so:

Set<YourObject> hashSet = getItSomehow();

Set<YourObject> treeSet = new TreeSet<YourObject>(new YourComparator());

treeSet.addAll(hashSet);

// Now it's sorted based on the logic as implemented in YourComparator.

If the items you have itself already implements [Comparable](http://download.oracle.com/javase/6/docs/api/java/lang/Comparable.html) and its default ordering order is already what you want, then you basically don't need to supply a Comparator. You could then construct the TreeSet directly based on the HashSet. E.g.

What is the difference between them? I know that

A LinkedHashSet is an ordered version of HashSet that maintains a doubly-linked List across all elements. Use this class instead of HashSet when you care about the iteration order. When you iterate through a HashSet the order is unpredictable, while a LinkedHashSet lets you iterate through the elements in the order in which they were inserted.

Difference between HashSet and LinkedHashSet:

A HashSet is unordered and unsorted Set. LinkedHashSet is the ordered version of HashSet.

The only difference between HashSet and LinkedHashSet is that LinkedHashSet maintains the insertion order. When we iterate through a HashSet, the order is unpredictable while it is predictable in case of LinkedHashSet.

The reason why LinkedHashSet maintains insertion order is because the underlying data structure is a doubly-linked list.

When to use LinkedHashSet over HashSet:

LinkedHashSet should be used in preference to HashSet when we want to iterate over the elements as per the insertion order. If order does not matter, then we can go for HashSet.

## Difference between TreeSet, LinkedHashSet and HashSet in Java

[TreeSet vs HashSet vs LinkedHashSet in Java with Example](http://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)TreeSet, LinkedHashSet and HashSet in Java are three Set implementation in collection framework and like many others they are also used to store objects. Main feature of TreeSet is sorting,  LinkedHashSet is insertion order and HashSet is just general purpose collection for storing object. HashSet is implemented using[HashMap in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html) while TreeSet is implemented using [TreeMap](http://javarevisited.blogspot.sg/2011/12/treemap-java-tutorial-example-program.html).  TreeSet is a SortedSet implementation which allows it to keep elements in the sorted order defined by either [Comparable or Comparator interface](http://javarevisited.blogspot.sg/2011/06/comparator-and-comparable-in-java.html). Comparable is used for natural order sorting and Comparator for [custom order sorting](http://java67.blogspot.sg/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html) of objects, which can be provided while creating instance of TreeSet. Anyway before seeing difference between TreeSet, LinkedHashSet and HashSet, let's see some similarities between them:

1) **Duplicates** : All three implements Set interface means they are not allowed to store duplicates.

2) **Thread safety**: HashSet, TreeSet and LinkedHashSet are not [thread-safe](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html), if you use them in multi-threading environment where at least one Thread  modifies Set you need to externally synchronize them.

3) **Fail-Fast Iterator** : Iterator returned by TreeSet, LinkedHashSet and HashSet are fail-fast Iterator. i.e. If Iterator is modified after its creation by any way other than Iterators remove() method, it will throw ConcurrentModificationException with best of effort. read more about [fail-fast vs fail-safe Iterator](http://javarevisited.blogspot.sg/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) here

Now let’s see **difference between HashSet, LinkedHashSet and TreeSet in Java** :

**Performance** and **Speed**: First difference between them comes in terms of  speed.  HashSet is fastest, LinkedHashSet is second on performance or almost similar to HashSet but TreeSet is bit slower because of sorting operation it needs to perform on each insertion. TreeSet provides guaranteed O(log(n)) time for common operations like add, remove and contains, while HashSet and LinkedHashSet offer constant time performance e.g. O(1) for add, contains and remove given hash function uniformly distribute elements in bucket.

**Ordering :** HashSet does not maintain any order while LinkedHashSet maintains insertion order of elements much like List interface and TreeSet maintains sorting order or elements.

**Internal Implementation :** HashSet is backed by an HashMap instance, LinkedHashSet is implemented using HashSet and LinkedList while TreeSet is backed up by NavigableMap in Java and by default it uses TreeMap.

**null** : Both HashSet and LinkedHashSet allows null but TreeSet doesn't allow null but TreeSet doesn't allow null and throw [java.lang.NullPointerException](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html) when you will insert null into TreeSet. Since TreeSet uses [compareTo() method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) of respective elements to compare them  which throws NullPointerException while comparing with null, here is an example:

TreeSet cities

Exception in thread "main" java.lang.NullPointerException

        at java.lang.String.compareTo(String.java:1167)

        at java.lang.String.compareTo(String.java:92)

        at java.util.TreeMap.put(TreeMap.java:545)

        at java.util.TreeSet.add(TreeSet.java:238)

**Comparison :** HashSet and LinkedHashSet uses [equals() method in Java](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) for comparison but TreeSet uses [compareTo() method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) for maintaining ordering. That's why compareTo() should be consistent to equals in Java. failing to do so break general contact of Set interface i.e. it can permit duplicates.

Read more: <http://javarevisited.blogspot.com/2012/11/difference-between-treeset-hashset-vs-linkedhashset-java.html#ixzz30aHaqfHC>

**Clone() method copies and it does not impact original object if modified**

## When to use LinkedList and ArrayList in Java

As I said LinkedList is not as popular as ArrayList but still there are situation where a LinkedList is better choice than ArrayList in Java. Use LinkedList in Java if:

1) Your application can live without Random access. Because if you need nth element in LinkedList you need to first traverse up to nth element O(n) and than you get data from that node.

2) Your application is more insertion and deletion driver and you insert or remove more than retrieval. Since insertion or

removal doesn't involve resizing its much faster than ArrayList.

That’s all on **difference between ArrayList and LinkedList in Java**. Use ArrayList in Java for all their situation where you need a **non-synchronized index based access**. ArrayList is fast and easy to use, just try to minimize array resizing by constructing arraylist with proper initial size.

# <http://www.tutorialspoint.com/java/java_using_comparator.htm>

# Java - How to use Comparator?

The Comparator interface defines two methods: compare( ) and equals( ). The compare( ) method, shown here, compares two elements for order:

Example:

class Dog implements Comparator<Dog>, Comparable<Dog>{

private String name;

private int age;

Dog(){

}

Dog(String n, int a){

name = n;

age = a;

}

public String getDogName(){

return name;

}

public int getDogAge(){

return age;

}

// Overriding the compareTo method

public int compareTo(Dog d){

return (this.name).compareTo(d.name);

}

// Overriding the compare method to sort the age

public int compare(Dog d, Dog d1){

return d.age - d1.age;

}

}

public class Example{

public static void main(String args[]){

// Takes a list o Dog objects

List<Dog> list = new ArrayList<Dog>();

list.add(new Dog("Shaggy",3));

list.add(new Dog("Lacy",2));

list.add(new Dog("Roger",10));

list.add(new Dog("Tommy",4));

list.add(new Dog("Tammy",1));

Collections.sort(list);// Sorts the array list

for(Dog a: list)//printing the sorted list of names

System.out.print(a.getDogName() + ", ");

// Sorts the array list using comparator

Collections.sort(list, new Dog());

System.out.println(" ");

for(Dog a: list)//printing the sorted list of ages

System.out.print(a.getDogName() +" : "+

a.getDogAge() + ", ");

}

}

This would produce the following result:

Lacy, Roger, Shaggy, Tammy, Tommy,

Tammy : 1, Lacy : 2, Shaggy : 3, Tommy : 4, Roger : 10,

**Convert List to Set**

new HashSet<Foo>(myList);

will give you an unsorted set which doesn't have duplicates.

1. An alternative that gives a sorted set is:

new TreeSet<Foo>(myList);

This works if Foo implements Comparable. If it doesn't then you may want to use a comparator:

Set<Foo> lSet = new TreeSet<Foo>(someComparator);

lSet.addAll(myList);

1. Collections.*addAll*(hs, a);

Here hs is HashSet to which elements “a” need to be added, here a can be array or any other Object,list..

Other: Collections.copy(dest,src) , but here both *dest* and *src* should be of same type.

**"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.

1. 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,
2. 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.
3. If people fail to recognize this and say it only stores Value in the bucket they will fail to explain the retrieving logic of any object stored in Java HashMap
4. . 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?”

**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 .

Read more: <http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html#ixzz35KhB7m4d>

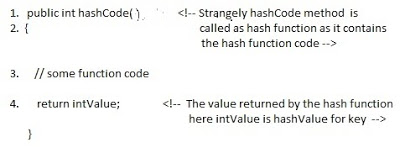
Now from here onwards real confusion starts, Some time candidate will say that since hashcode is equal, both objects are equal and HashMap  will throw exception or not store them again etc, Then you might want to remind them about [equals() and hashCode() contract](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html)that two unequal object in Java can have same hashcode. Some will give up at this point and few will move ahead and say "Since hashcode is same, bucket location would be same and collision will occur in HashMap, Since HashMap use LinkedList to store object, this entry (object of Map.Entry comprise key and value )  will be stored in [LinkedList](http://javarevisited.blogspot.sg/2012/02/difference-between-linkedlist-vs.html). Great this answer make sense though there are many collision resolution methods available this is simplest and HashMap in Java does follow this. But story does not end here and interviewer asks  
  
Read more: <http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html#ixzz35KhGFomz>

Good link: http://javahungry.blogspot.com/2013/08/hashing-how-hash-map-works-in-java-or.html

### ashing :How hash map works in java or How get() method works internally

One of the most darling question of the core java interviewers is How hash map works in java or internal.implementation of hashmap. Most of the candidates rejection chances increases if the candidate do not give the satisfactory explanation . This question shows that candidate has good knowledge of Collection . So this question should be in your to do list before appearing for the interview .  
  
Read also  [How Hashset works in java or How it ensures uniqueness in java](http://javahungry.blogspot.com/2013/08/how-sets-are-implemented-internally-in.html)  
  
**How Hashmap works in Java**  
  
HashMap works on the principle of Hashing .  To understand Hashing , we should understand the three terms first   i.e  *Hash Function , Hash Value and Bucket .*  
  
**What is Hash Function , Hash Value  and Bucket ?**  
  
hashCode() function  which returns an integer value is the **Hash function**. The important point to note that ,  this method is present in [Object class ( Mother of all class )](http://javahungry.blogspot.com/2013/06/object-class-and-methods-in-java-example-explanation.html) .  
  
This is the code for the hash function(also known as hashCode method) in Object Class :  
  
    public native int hashCode();

The most important point to note from the above line :  hashCode method return  int value .  
So the **Hash value**isthe int value returned by the hash function**.**  
  
  
    So summarize the terms in the diagram below :  
                   

[](http://3.bp.blogspot.com/-ohpWRYtP3N8/UgVGHwEk3YI/AAAAAAAAAbQ/K5BKPJ8dfmQ/s1600/How+hash+map+works+.jpg)

**What is bucket ?**   
A bucket is used to store key value pairs . A bucket can have multiple key-value pairs . In hash map, bucket used simple linked list to store objects .  
  
After understanding the terms we are ready to move next step , **How hash map works in java** **or How get() works internally in java .**  
  
  
  
**Code inside Java Api (HashMap class internal implementation) for HashMap get(Obejct key) method**

**1**. Public V get(Object key)

{

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

**3**. //Some code

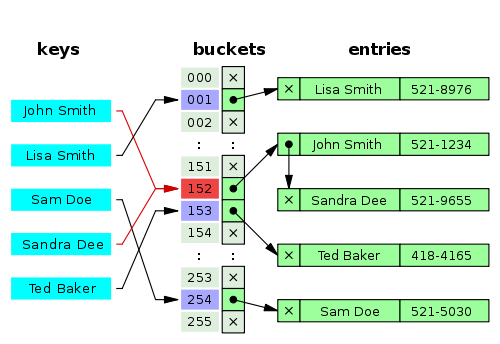
**4**. **int** hash = hash(key.hashCode());

**5**. // if key found in hash table then return value

**6**. // else return null

}

**Hash map** **works on the principle of hashing**  
  
HashMap get(Key k) method calls hashCode method on the key object and applies returned hashValue to its own static hash function to find a bucket location(backing array) where keys and values are stored in form of a**nested class called Entry (Map.Entry)**. So you have concluded that from the previous line that**Both key and value is stored in the bucket as a form of  Entry object**. So thinking that Only value is stored  in the bucket is not correct and will not give a good impression on the interviewer .  
  
\* Whenever we call get( Key k )  method on the HashMap object . First it checks that whether key is null or not .  Note that **there can only be one null key in HashMap .**  
 **If key is null , then Null keys always map to hash 0, thus index 0.**  
  
If key is not null then , it will call hashfunction on the key object , see line 4 in above method i.e. key.hashCode()  ,so after key.hashCode() returns hashValue , line 4 looks like  
  
4.                int hash = hash(hashValue)  
  
 , and now ,it applies returned hashValue into its own hashing function .  
  
**We might wonder why we are calculating the hashvalue again using hash(hashValue).** Answer is ,It defends against poor quality hash functions.  
  
Now step 4 final  hashvalue is used to find the bucket location at which the Entry object is stored .**Entry object stores in the bucket like this (hash,key,value,bucketindex) .**  
  
**Interviewer:    What if  when two different keys have the same hashcode ?**  
Solution, [equals() method](http://javahungry.blogspot.com/2013/06/difference-between-equals-and-double-equals-method-with-example-java-collections-interview-question.html) comes to rescue.Here candidate gets puzzled. Since bucket is one and we have two objects with the same hashcode .Candidate usually forgets that bucket is a simple linked list.  
  
**The bucket is the linked list effectively . Its not a LinkedList as in a java.util.LinkedList - It's a separate (simpler) implementation just for the map .**  
  
**So we traverse through linked list , comparing keys in each entries using keys.equals() until it return true.  Then the corresponding entry object Value is returned .**

[](http://4.bp.blogspot.com/-unPwpp8AJTA/U0e9S0F5ljI/AAAAAAAAAUo/xMnUVRO5fyY/s1600/how+hashmap+works+internally+in+java+.png)

One of  our readers Jammy  asked a very good  question   
  
**When the functions 'equals' traverses through the linked list does it traverses from start to end one by one...in other words brute method. Or the linked list is sorted based on key and then it traverses?**  
  
Answer is when an element is added/retrieved, same procedure follows:  
  
  
a. Using key.hashCode() [ see above step 4],determine initial hashvalue for the key  
  
b. Pass intial hashvalue as hashValue  in    hash(hashValue) function, to calculate the final hashvalue.  
  
c. Final hash value is then passed as a first parameter in the indexFor(int ,int )method .  
    The second parameter is length which is a constant in HashMap Java Api , represented by                             DEFAULT\_INITIAL\_CAPACITY  
  
    The default  value of DEFAULT\_INITIAL\_CAPACITY is 16 in HashMap Java Api .  
  
 indexFor(int,int) method  returns the first entry in the appropriate bucket. The linked list in the bucket is then iterated over - (the end is found and the element is added or the key is matched and the value is returned )  
  
  
Explanation about indexFor(int,int) is below :

/\*\*

\* Returns index for hash code h.

\*/

**static** **int** **indexFor**(**int** h, **int** length) {

**return** h & (length-**1**);

}

The above function indexFor() works because Java HashMaps always have a capacity, i.e. number of buckets, as a power of 2.  
 Let's work with a capacity of 256,which is 0x100, but it could work with any power of 2. Subtracting 1  
from a power of 2 yields the exact bit mask needed to bitwise-and with the hash to get the proper bucket index, of range 0 to length - 1.  
256 - 1 = 255  
0x100 - 0x1 = 0xFF  
E.g. a hash of 257 (0x101) gets bitwise-anded with 0xFF to yield a bucket number of 1.

