## List vs. Set in Java

*Important point to note is that both List and Set are derived from Collection Interface*. In short main difference between List and Set in Java is that List is an **ordered collection** which **allows duplicates** while ***Set*** is an ***unordered collection*** which *doesn't allow duplicates*.

Both List and Set are two of most important Collection classes Java Program use along with various Map implementation. Basic feature of List and Set are abstracted in List and Set interface in Java and then various implementation of List and Set adds specific feature on top of that e.g. [ArrayList in Java](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html) is a List implementation backed by Array while LinkedList is anotherList implementation which works like linked list data-structure.

Both of them are used to store objects and provide convenient API to insert, remove and retrieve elements, along with to support Iteration over 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 calledSortedSet which can store Set elements 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.

3) Set uses [equals() method](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html) to check uniqueness of elements stored in Set, while SortedSet uses [compareTo() method](http://javarevisited.blogspot.com/2011/11/how-to-override-compareto-method-in.html) to implement natural sorting order of elements. In order for an element to behave properly in Set and SortedSet, [equals and compareTo must be consistent](http://javarevisited.blogspot.com/2011/11/how-to-override-compareto-method-in.html) to each other.

4) Popular implementation of List interface in Java includes ArrayList, Vector and LinkedList. While popular implementation of Set interface includes HashSet, TreeSet and LinkedHashSet.

When to use List and Set in Java  
  
its pretty clear that if you need to maintain insertion order or object and you collection can contain duplicates than List is a way to go. On the other hand if your requirement is to maintain unique collection without any duplicates than Set is the way to go.

Read more: <http://javarevisited.blogspot.com/2012/04/difference-between-list-and-set-in-java.html#ixzz2U6Av4Yhh>

**LinkedList vs ArrayList in Java**

LinkedList and [ArrayList](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html) both implement List Interface but how they work internally is where the differences lies. Main difference between ArrayList and LinkedList is that ArrayList is implemented using **re sizable array** while LinkedList is implemented using **doubly LinkedList**.

All the differences between LinkedList and ArrayList have there root on difference between [Array](http://javarevisited.blogspot.com/2012/01/anonymous-array-example-java-create.html) and LinkedList data-structure.



1) Since Array is an index based data-structure searching or getting element from Array with index is pretty fast. Array provides O(1) performance for get(index) method but remove is costly in ArrayList as you need to rearrange all elements. On the Other hand LinkedList doesn't provide Random or index based access and you need to iterate over linked list to retrieve any element which is of order O(n).

2) Insertions  are easy and fast in LinkedList as compared to ArrayList because there is no risk of resizing array

and copying content to new array if array gets full which makes adding into ArrayList of O(n) in worst case, while adding is O(1) operation in LinkedList in Java. ArrayList also needs to update its index if you insert something anywhere except at the end of array.

3) Removal is like insertions better in LinkedList than ArrayList.

4) 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.

## 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.

Use ArrayList in Java for all there 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.  
  
Read more: <http://javarevisited.blogspot.com/2012/02/difference-between-linkedlist-vs.html#ixzz2U674bkcX>

## Vector vs ArrayList in Java

ArrayList and Vector are two of most used class on java collection package   
Similarities between these two and why we can use ArrayList in place of Vector on certain scenario.

[ArrayList vs Vector in Java](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html)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.

### Differences are based upon properties like synchronization, thread safety, speed, performance, navigation and Iteration over List etc. 1) Synchronization and thread-safety

First and foremost difference between Vector and ArrayList is that **Vector is synchronized** and ArrayList is not, what it means is that all the method which structurally modifies Vector e.g. add () or remove () are [synchronized](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html)which makes it [thread-safe](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html)and allows it to be used safely in a multi-threaded and concurrent environment. On the other hand ArrayList methods are not synchronized thus not suitable for use in multi-threaded environment. This is also a [popular interview question on thread](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html), where people ask why ArrayList can not be shared between multiple threads.

### 2) Speed and Performance

ArrayList is way faster than Vector**.**Since Vector is synchronized and thread-safe it pays price of synchronization which makes it little slow. On the other hand ArrayList is not synchronized and fast which makes it obvious choice in a single-threaded access environment. You can also use ArrayList in a multi-threaded environment if multiple threads are only reading values from ArrayList or you can create [read only ArrayList](http://javarevisited.blogspot.sg/2011/06/converting-array-to-arraylist-in-java.html) as well.

### 3) Capacity

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.

### 4) Enumeration and Iterator

Vector can return enumeration of items it hold by calling elements () method which is not [*fail-fast*](http://javarevisited.blogspot.sg/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html)*a*s opposed to Iterator and ListIterator returned by ArrayList. I have discussed this point in detail on my post [What is difference between Iterator and Enumeration](http://javarevisited.blogspot.sg/2010/10/what-is-difference-between-enumeration.html), you can also look there.

### 5) Legacy

Another point worth to remember is Vector is one of those classes which comes with JDK 1.0 and initially not part of Collection framework but in later version it's been re-factored to implement List interface so that it could become part of collection framework

After considering these points about both Vector and ArrayList , my conclusion is use ArrayList wherever possible and avoids use of Vector until you have no choice. Think for CopyOnWriteArrayList over Vector, if you have multiple readers and few writers because it can provide thread-safety without impacting performance too much.

Read more: <http://javarevisited.blogspot.com/2011/09/difference-vector-vs-arraylist-in-java.html#ixzz2U6HclAvH>

**Enumeration vs iterator**

Important point to note is that both Iterator and Enumeration provides way to traverse or navigate through entire collection in java

Between Enumeration and Iterator, Enumeration is older and its there from JDK1.0 while iterator was introduced later. Iterator can be used with [Java arraylist](http://javarevisited.blogspot.com/2011/06/converting-array-to-arraylist-in-java.html),  [java hashmap keyset](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html)  and with any other collection classes.

Another similarity between Iterator and Enumeration in Java is that functionality *of Enumeration interface is duplicated by the Iterator interface*.  
  
Only major difference between Enumeration and iterator is Iterator has a remove() method while Enumeration doesn't. Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects, where as by using Iterator we can manipulate the objects like adding and removing the objects from collection e.g. Arraylist.

Also **Iterator**is more secure and safe as compared to **Enumeration**because it  does not allow other thread to modify the collection object while some thread is iterating over it and throws **ConcurrentModificationException.**This is by far most important fact for me for deciding between Iterator vs Enumeration in Java.

In Summary both Enumeration and Iterator will give successive elements, but Iterator is new and improved version where method names are shorter, and has new method called remove. Here is a short comparison:

**Enumeration**

hasMoreElement()

nextElement()

N/A

**Iterator**

hasNext()

next()

remove()

So **Enumeration** is used whenever we want to make Collection objects as Read-only.

Readmore: <http://javarevisited.blogspot.com/2010/10/what-is-difference-between-enumeration.html#ixzz2U6JSJYEG>

**Fail-safe vs Fail-fast Iterator in Java**

## Concept of fail-safe iterator are relatively new in Java and first introduced with Concurrent Collections in Java 5 like ConcurrentHashMap and CopyOnWriteArrayList.

## Fail-fast Iterators in Java:

[Difference between fail-safe vs fail-fast iterator in java](http://2.bp.blogspot.com/-wrzDeQGAe1I/TWu8pLuLr4I/AAAAAAAAADE/V017G-6Q61w/s1600/java_logo_50_50.jpg)As name suggest **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.

Java doc says this is not a guaranteed behavior instead its done of "best effort basis", So application programming can not  rely on this behavior. Also since multiple threads are involved while updating and checking modification count and this check  is done without synchronization, there is a chance that Iteration thread still sees a stale value and might not be able to detect any change done by parallel threads. Iterators returned by most of JDK1.4 collection are fail-fast including Vector, [ArrayList](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html), HashSet etc. to read more about Iterator see my post [What is Iterator in Java](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html).

**Fail-safe Iterator in java:**

Contrary to fail-fast Iterator, **fail-safe iterator** doesn't throw any Exception if Collection is modified structurally while one thread is Iterating over it 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 in Java.

Read more: <http://javarevisited.blogspot.com/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html#ixzz2U6Lll0s3>

***ConcurrentHashMap VS Hashtable VS synchronizedMap***

### *Why we 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**.

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.

### Difference between ConcurrentHashMap and Collections.synchronizedMap

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

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

Read more: <http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html#ixzz2U6rLqr1F>

## HashMap and Hashtable

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](http://javarevisited.blogspot.sg/2012/03/simpledateformat-in-java-is-not-thread.html), [synchronization](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html)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](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. Another significant difference between [HashMap vs Hashtable](http://java67.blogspot.sg/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html) is that Iterator in the HashMap is  a [fail-fast iterator](http://javarevisited.blogspot.sg/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) 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.

Some Important Terms

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.

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);

Read more: <http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html#ixzz2U6vgZHcW>

**HashMap vs Hashset**

**HashSet** is implementation of **Set** Interface which does not allow duplicate value all the methods which are in **Collection** **Framework** are also in Set Interface by default but when we are talking about Hash set the main thing is objects which are going to be stored in HashSet must [override **equals**()](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html) and **hashCode**() method so that we can check for equality and no duplicate value are stored in our set.if we have created our own objects we need to implement **hashCode**() and **equal**() in such a manner that will be able to compare objects correctly when storing in a set so that duplicate objects are not stored,if we have not override this method objects will take default implementation of this method.

**public boolean add(Object o)**Method is used to add element in a set which returns false if it’s a duplicate value in case of  HashSet otherwise returns true if added successfully.

**HashMap** is a implementation of Map Interface, which maps a key to value.Duplicate keys are not allowed in a map.Basically map Interface has two implementation classes HashMap and TreeMap the main difference is TreeMap maintains order of the objects but HashMap will not.HashMap allows null values and null keys.[HashMap is not synchronized](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html),but collection framework provide methods so that we can make them synchronized if multiple threads are going to access our hashmap and one thread is structurally change our map.

**public Object put(Object Key,Object value)**method is used to add element in map.

### Difference between HashSet and HashMap in Java

Following are some differences between HashMap and Hashset:

|  |  |
| --- | --- |
| **Hash Map** | **Hash Set** |
| HashMap  is a implementation of Map interface | HashSet is an implementation of Set Interface |
| HashMap Stores data in form of  key value pair | HashSet Store only objects |
| Put method is used to add element in map | Add method is used to add element is Set |
| In hash map hashcode value is calculated using key object | Here member object is used for calculating hashcode value which can be same for two objects so equal () method is used to check for equality if it returns false that means two objects are different. |
| HashMap is faster than hashset because unique key is used to access object | HashSet is slower than Hashmap |

Read more: <http://javarevisited.blogspot.com/2011/09/difference-hashmap-vs-hashset-java.html#ixzz2U6we3olH>

**How HashMap works in Java**

[HashMap accept null while Hashtable doesn't](http://www.blogger.com/), [HashMap is not synchronized](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html), HashMap is fast and so on along with basics like its stores key and value pairs etc.

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.

**"Do you Know how HashMap works in Java”** or **"How does get () method of HashMap works in Java"**

And then you get answers like I don't bother its standard Java API, you better look code on Java source or Open JDK; I can find it out in Google at any time etc. But some interviewee definitely answer this and will say **"HashMap works on principle of hashing**, we have put(key, value) and get(key) method for storing and retrieving Objects from HashMap. When we pass Key and Value object  to put() method on Java HashMap, HashMap implementation calls [hashCode method](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html)on Key object and applies returned hashcode into its own hashing function to find a bucket location for storing Entry object, important point to mention is that HashMap in Java stores both key and value object as Map.Entry in bucket which is essential to understand the retrieving logic. If people fails to recognize this and say it only stores Value in the bucket they will fail to explain the retrieving logic of any object stored in Java HashMap . This answer is very much acceptable and does make sense that interviewee has fair bit of knowledge on how hashing works and how HashMap  works in Java. But this is just start of story and confusion increases when you put interviewee on scenarios faced by Java developers on day by day basis. Next question could be about collision detection and collision resolution in Java HashMap  e.g.

**"What will happen if two different objects have same hashcode?”**

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

**"How will you retrieve Value object  if two Keys will have same hashcode?”**

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

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

**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 :

Read more: <http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html#ixzz2UCa1Iowq>