COLLECTIONS FRAMEWORK INTERVIEW QUESTIONS

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **Vector** |
| 1) | ArrayList is not synchronized. | Vector is synchronized. |
| 2) | ArrayList is not a legacy class. | Vector is a legacy class. |
| 3) | ArrayList increases its size by 50% of the array size. | Vector increases its size by doubling the array size. |

2) What is the difference between ArrayList and LinkedList?

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **LinkedList** |
| 1) | ArrayList uses a dynamic array. | LinkedList uses doubly linked list. |
| 2) | ArrayList is not efficient for manipulation because a lot of shifting is required. | LinkedList is efficient for manipulation. |
| 3) | ArrayList is better to store and fetch data. | LinkedList is better to manipulate data. |

3) What is the difference between Iterator and ListIterator?

Iterator traverses the elements in forward direction only whereas ListIterator traverses the elements in forward and backward direction.

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **ListIterator** |
| 1) | Iterator traverses the elements in forward direction only. | ListIterator traverses the elements in backward and forward directions both. |
| 2) | Iterator can be used in List, Set and Queue. | ListIterator can be used in List only. |

5) What is the difference between List and Set?

List can contain duplicate elements whereas Set contains only unique elements.

6) What is the difference between HashSet and TreeSet?

HashSet maintains **no order** whereas TreeSet maintains **ascending order**.

7) What is the difference between Set and Map?

Set contains values only whereas Map contains key and values both.

8) What is the difference between HashSet and HashMap?

HashSet contains only values whereas HashMap contains entry(key,value). HashSet can be iterated but HashMap need to convert into Set to be iterated.

9) What is the difference between HashMap and TreeMap?

HashMap maintains **no order** but TreeMap maintains **ascending order**.

10) What is the difference between HashMap and Hashtable?

|  |  |  |
| --- | --- | --- |
| **No.** | **HashMap** | **Hashtable** |
| 1) | HashMap is not synchronized. | Hashtable is synchronized. |
| 2) | HashMap can contain one null key and multiple null values. | Hashtable cannot contain any null key or null value. |

|  |  |  |
| --- | --- | --- |
| **No.** | **Comparable** | **Comparator** |
| 1) | Comparable provides only one sort of sequence. | Comparator provides multiple sort of sequences. |
| 2) | It provides one method named compareTo(). | It provides one method named compare(). |
| 3) | It is found in java.lang package. | it is found in java.util package. |
| 4) | If we implement Comparable interface, actual class is modified. | Actual class is not modified. |

11) What is the difference between Collection and Collections?

Collection is an interface whereas Collections is a class. Collection interface provides normal functionality of data structure to List, Set and Queue. But, Collections class is to sort and synchronize collection elements.

12) What is the difference between Comparable and Comparator?

13) What is the advantage of Properties file?

If you change the value in properties file, you don't need to recompile the java class. So, it makes the application easy to manage.

Use hardwired constants in your Java code when *you don't want* users / deployers / testers / tests changing them.

Use a properties file when *you do want* this to be a possibility.

The point is that changing a hard-wired constant in your application's source code entails editing the source code, rebuilding and redeploying. By contrast, changing a properties file may be as simple as firing up NotePad.

# Properties

*Properties* are configuration values managed as *key/value pairs*. In each pair, the key and value are both [String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) values. The key identifies, and is used to retrieve, the value, much as a variable name is used to retrieve the variable's value. For example, an application capable of downloading files might use a property named "download.lastDirectory" to keep track of the directory used for the last download.

To manage properties, create instances of [java.util.Properties](https://docs.oracle.com/javase/8/docs/api/java/util/Properties.html" \t "_blank). This class provides methods for the following:

* loading key/value pairs into a Properties object from a stream,
* retrieving a value from its key,
* listing the keys and their values,
* enumerating over the keys, and
* saving the properties to a stream.

For an introduction to streams, refer to the section [I/O Streams](http://docs.oracle.com/javase/tutorial/essential/io/streams.html) in the [Basic I/O](http://docs.oracle.com/javase/tutorial/essential/io/index.html) lesson.

Properties extends [java.util.Hashtable](https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html" \t "_blank). Some of the methods inherited from Hashtable support the following actions:

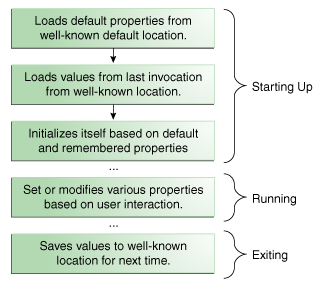
* testing to see if a particular key or value is in the Properties object,
* getting the current number of key/value pairs,
* removing a key and its value,
* adding a key/value pair to the Properties list,
* enumerating over the values or the keys,
* retrieving a value by its key, and
* finding out if the Properties object is empty.

**Security Considerations:** Access to properties is subject to approval by the current security manager. The example code segments in this section are assumed to be in standalone applications, which, by default, have no security manager. The same code in an applet may not work depending on the browser in which it is running. See [What Applets Can and Cannot Do](http://docs.oracle.com/javase/tutorial/deployment/applet/security.html) in the [Java Applets](http://docs.oracle.com/javase/tutorial/deployment/applet/index.html) lesson for information about security restrictions on applets.

The System class maintains a Properties object that defines the configuration of the current working environment. For more about these properties, see [System Properties](http://docs.oracle.com/javase/tutorial/essential/environment/sysprop.html). The remainder of this section explains how to use properties to manage application configuration.

## Properties in the Application Life Cycle

The following figure illustrates how a typical application might manage its configuration data with a Properties object over the course of its execution.



* Starting Up  
  The actions given in the first three boxes occur when the application is starting up. First, the application loads the default properties from a well-known location into a Properties object. Normally, the default properties are stored in a file on disk along with the .class and other resource files for the application.  
  Next, the application creates another Properties object and loads the properties that were saved from the last time the application was run. Many applications store properties on a per-user basis, so the properties loaded in this step are usually in a specific file in a particular directory maintained by this application in the user's home directory. Finally, the application uses the default and remembered properties to initialize itself.  
  The key here is consistency. The application must always load and save properties to the same location so that it can find them the next time it's executed.
* Running  
  During the execution of the application, the user may change some settings, perhaps in a Preferences window, and the Properties object is updated to reflect these changes. If the users changes are to be remembered in future sessions, they must be saved.
* Exiting  
  Upon exiting, the application saves the properties to its well-known location, to be loaded again when the application is next started up.

## Setting Up the Properties Object

The following Java code performs the first two steps described in the previous section: loading the default properties and loading the remembered properties:

. . .

// create and load default properties

Properties defaultProps = new Properties();

FileInputStream in = new FileInputStream("defaultProperties");

defaultProps.load(in);

in.close();

// create application properties with default

Properties applicationProps = new Properties(defaultProps);

// now load properties

// from last invocation

in = new FileInputStream("appProperties");

applicationProps.load(in);

in.close();

. . .

First, the application sets up a default Properties object. This object contains the set of properties to use if values are not explicitly set elsewhere. Then the load method reads the default values from a file on disk named defaultProperties.

Next, the application uses a different constructor to create a second Properties object, applicationProps, whose default values are contained in defaultProps. The defaults come into play when a property is being retrieved. If the property can't be found in applicationProps, then its default list is searched.

Finally, the code loads a set of properties into applicationProps from a file named appProperties. The properties in this file are those that were saved from the application the last time it was invoked, as explained in the next section.

## Saving Properties

The following example writes out the application properties from the previous example using Properties.store. The default properties don't need to be saved each time because they never change.

FileOutputStream out = new FileOutputStream("appProperties");

applicationProps.store(out, "---No Comment---");

out.close();

The store method needs a stream to write to, as well as a string that it uses as a comment at the top of the output.

## Getting Property Information

Once the application has set up its Properties object, the application can query the object for information about various keys and values that it contains. An application gets information from a Properties object after start up so that it can initialize itself based on choices made by the user. The Properties class has several methods for getting property information:

* contains(Object value) and containsKey(Object key)  
  Returns true if the value or the key is in the Properties object. Properties inherits these methods from Hashtable. Thus they accept Object arguments, but only String values should be used.
* getProperty(String key) and getProperty(String key, String default)  
  Returns the value for the specified property. The second version provides for a default value. If the key is not found, the default is returned.
* list(PrintStream s) and list(PrintWriter w)  
  Writes all of the properties to the specified stream or writer. This is useful for debugging.
* elements(), keys(), and propertyNames()  
  Returns an Enumeration containing the keys or values (as indicated by the method name) contained in the Properties object. The keys method only returns the keys for the object itself; the propertyNames method returns the keys for default properties as well.
* stringPropertyNames()  
  Like propertyNames, but returns a Set<String>, and only returns names of properties where both key and value are strings. Note that the Set object is not backed by the Properties object, so changes in one do not affect the other.
* size()  
  Returns the current number of key/value pairs.

## Setting Properties

A user's interaction with an application during its execution may impact property settings. These changes should be reflected in the Properties object so that they are saved when the application exits (and calls the store method). The following methods change the properties in a Properties object:

* setProperty(String key, String value)  
  Puts the key/value pair in the Properties object.
* remove(Object key)  
  Removes the key/value pair associated with key.

**Note:** Some of the methods described above are defined in Hashtable, and thus accept key and value argument types other than String. Always use Strings for keys and values, even if the method allows other types. Also do not invoke Hashtable.set or Hastable.setAll on Properties objects; always use Properties.setProperty.

|  |  |
| --- | --- |
|  | A .properties file is a simple collection of key-value pairs that can be parsed by the [java.util.Properties](http://java.sun.com/javase/6/docs/api/java/util/Properties.html) class.  Properties files are widely used for many purposes in all kinds of Java applications, often to store configuration or localization data. |

14) What does the hashCode() method?

The hashCode() method returns a hash code value (an integer number).

The hashCode() method returns the same integer number, if two keys (by calling equals() method) are same.

But, it is possible that two hash code numbers can have different or same keys.

15) Why we override equals() method?

The equals method is used to check whether two objects are same or not. It needs to be overridden if we want to check the objects based on property.

For example, Employee is a class that has 3 data members: id, name and salary. But, we want to check the equality of employee object on the basis of salary. Then, we need to override the equals() method.

16) How to synchronize List, Set and Map elements?

Yes, Collections class provides methods to make List, Set or Map elements as synchronized:

|  |
| --- |
| public static List synchronizedList(List l){} |
| public static Set synchronizedSet(Set s){} |
| public static SortedSet synchronizedSortedSet(SortedSet s){} |
| public static Map synchronizedMap(Map m){} |
| public static SortedMap synchronizedSortedMap(SortedMap m){} |

17) What is the advantage of generic collection?

If we use generic class, we don't need typecasting. It is typesafe and checked at compile time.

18) What is hash-collision in Hashtable and how it is handled in Java?

Two different keys with the same hash value is known as hash-collision. Two different entries will be kept in a single hash bucket to avoid the collision.

19) What is the Dictionary class?

The Dictionary class provides the capability to store key-value pairs.

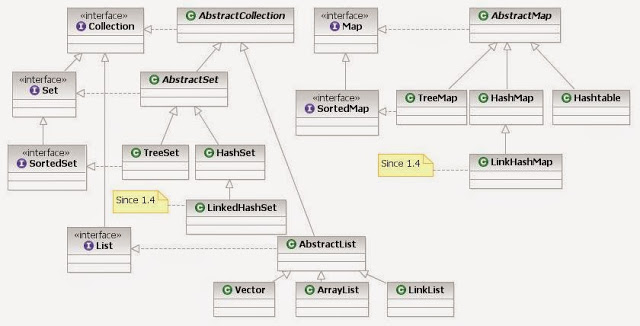
20) What is the default size of load factor in hashing based collection?

The default size of load factor is **0.75**. The default capacity is computed as initial capacity \* load factor. For example, 16 \* 0.75 = 12. So, 12 is the default capacity of Map.

**Q1  What is Collection ? What is a Collections Framework ? What are the benefits of Java Collections Framework ?**  
  
**Collection :** A collection (also called as container) is an object  that groups multiple elements into a single unit.  
  
**Collections Framework :**Collections framework provides unified architecture for manipulating and representing collections.  
  
**Benefits of Collections Framework :**  
1. Improves program quality and speed  
2. Increases the chances of reusability of software  
3. Decreases programming effort.  
  
**Q2 What is the root interface in collection hierarchy ?**  
  
Root interface in collection hierarchy is **Collection interface .**Few interviewer may argue that   
**Collection interface extends Iterable interface. So iterable should be the root interface. But you should reply iterable interface present in java.lang package not in java.util package .**It is clearly mentioned in [Oracle Collection  docs](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) , that Collection interface is a member of the Java Collections framework.  For [Iterable interface Oracle doc](https://docs.oracle.com/javase/7/docs/api/java/lang/Iterable.html) , iterable interface is not mentioned as a part of the Java Collections framework .So if the question includes  collection hierarchy , then you should answer the question as Collection interface (which is found in java.util package).  
 **Q3 What is the difference between Collection and Collections ?**  
  
Collection is  an interface while Collections is a java class , both are present in java.util package and  part of java collections framework.

**Q4 Which collection classes are synchronized or thread-safe ?**  
  
Stack, Properties , Vector and Hashtable can be used in multi threaded environment because they are synchronized classes (or thread-safe).

**Q5 Name the core Collection  interfaces ?**

[](http://1.bp.blogspot.com/-ifC30f-ZZ7M/VU7wWRwqo3I/AAAAAAAAAmQ/b1CIhOP5NU8/s1600/Java_collection_framework.jpg)

The list of core collection interfaces are : just mention the important ones  
  
Important : Collection , Set , Queue , List , Map  
  
Other interface also in the list :  SortedSet, SortedMap , Deque, ListIterator etc.  
 **Q6 What is the difference between List and Set ?**  
  
Set contain only unique elements while List can contain duplicate elements.  
Set is unordered while List is ordered .

List maintains the order in which the objects are added .  
  
**Q7 What is the difference between Map and Set ?**  
  
Map object has unique keys each containing some value, while **Set contain only unique values.**  
**Q8 What are the classes implementing List and Set interface ?**  
  
***Class implementing List interface :***  ArrayList , Vector , LinkedList ,  
  
***Class implementing Set interface :***HashSet , TreeSet

**Q9 What is an iterator ?**  
  
Iterator is an interface . It is found in java.util package. It provides methods to iterate over any Collection.  
  
  
**Q10 What is the difference between Iterator and Enumeration ?**  
  
The main difference between Iterator and Enumeration is that Iterator has remove() method while Enumeration doesn't.  
Hence , using Iterator we can manipulate objects by adding and removing the objects from the collections. Enumeration behaves like a read only interface as it can only traverse the objects and fetch it .  
  
**Q11 Which design pattern followed by Iterator ?**  
  
It follows iterator design pattern. Iterator design pattern provides us to navigate through the collection of objects by using a common interface without letting us know about the underlying implementation.  
  
Enumeration is an example of Iterator design pattern.  
 **Q12 Which methods you need to override to use any object as key in HashMap ?**  
  
To use any object as key in HashMap , it needs to implement equals() and hashCode() method .  
  
**Q13  What is the difference between Queue and Stack ?**  
  
Queue is a data structure which is based on FIFO ( first in first out ) property . An example of Queue in real world is buying movie tickets in the multiplex or cinema theaters.  
  
Stack is a data structure which is based on LIFO (last in first out) property . An example of Stack in real world is  insertion or removal of CD  from the CD case.  
  
**Q14 How to reverse the List in Collections ?**  
  
There is a built in reverse method in Collections class . reverse(List list) accepts list as parameter.  
  
**Collections.reverse(listobject);**  
  
**Q15 How to convert the array of strings into the list ?**  
  
Arrays class of java.util package contains the method asList() which accepts the array as parameter.  
So,  
  
**String[]  wordArray =  {"Love Yourself"  , "Alive is Awesome" , "Be in present"};**  
**List wordList =  Arrays.asList(wordArray);**  
  
  
***Intermediate Level (1-3 yrs): Java Collections Interview Questions  and Answers***

**Q16 What is the difference between ArrayList and Vector ?**

It is one of the frequently asked collection interview question , the main differences are  
Vector is synchronized while ArrayList is not . Vector is slow while ArrayList is fast . Every time when needed, Vector increases the capacity twice of its initial size while ArrayList increases its ArraySize by 50%.

**Q17 What is the difference between HashMap and Hashtable ?**  
  
Main differences between HashMap and Hashtable are :  
  
a. HashMap allows one null key and any number of null values while Hashtable does not allow null keys and null values.  
b. HashMap is not synchronized or thread-safe while Hashtable is synchronized or thread-safe .  
  
  
**Q18 What is the difference between peek(),poll() and remove() method of the Queue interface ?**  
  
Both poll() and remove() method is used to remove head object of the Queue. The main difference lies when the Queue is empty().  
**If Queue is empty then poll() method will return null . While in similar case , remove() method will throw NoSuchElementException** .  
peek() method retrieves but does not remove the head of the Queue. If queue is empty then peek() method also returns null.

**Q19 What is the difference between Iterator and ListIterator.**  
  
Using Iterator we can traverse the list of objects in forward direction . But ListIterator can traverse the collection in both directions that is forward as well as backward.  
  
**Q20 What is the difference between Array and ArrayList in Java ?**  
  
a. Array is static in size while ArrayList is dynamic in size.  
b. Array can contain primitive data types while ArrayList can not contain primitive data types.

**Q21 What is the difference between HashSet and TreeSet ?**  
  
Main differences between HashSet and TreeSet are :  
a.  HashSet maintains the inserted elements in random order while TreeSet maintains elements in the sorted order  
b. HashSet can store null object while TreeSet can not store null object.  
  
  
**Q22 Write java code showing insertion,deletion and retrieval of HashMap object ?**  
**Q23 What is the difference between HashMap and ConcurrentHashMap ?**  
  
Main differences between HashMap and ConcurrentHashMap are :  
a. HashMap is not synchronized while ConcurrentHashMap is synchronized.  
b. HashMap can have one null key and any number of null values while ConcurrentHashMap does not allow null keys and null values .  
**Q24 Arrange the following in the ascending order (performance):**  
**HashMap , Hashtable , ConcurrentHashMap and Collections.SynchronizedMap**  
  
Hashtable  <  Collections.SynchronizedMap  <  ConcurrentHashMap  <  HashMap

**Q25 How HashMap works in Java ?**  
  
This is one of the most important question for java developers. HashMap  works on the principle of Hashing . Find detailed information here to understand [what is hashing and how hashmap works in java](http://javahungry.blogspot.co.uk/2013/08/hashing-how-hash-map-works-in-java-or.html) .  
  
**Q26 What is the difference between LinkedList and ArrayList in Java ?**  
  
Main differences between LinkedList and ArrayList are :  
a. LinkedList is the doubly linked list implementation of list interface , while , ArrayList is the resizable array implementation of list interface.  
b. LinkedList can be traversed in the reverse direction using descendingIterator() method  provided by the Java Api developers , while , we need to implement our own method to traverse ArrayList in the reverse direction .   
  
  
**Q28 Why Map interface does not extend the Collection interface in Java Collections Framework ?**  
  
One liner answer : **Map interface is not compatible with the Collection interface.**  
Explanation : Since Map requires key as well as value , for example , if we want to add key-value pair then we will use put(Object key , Object value) . So there are two parameters required to add element to the HashMap object  . In Collection interface add(Object o) has only one parameter.   
The other reasons are Map supports valueSet , keySet as well as other appropriate methods which have just different views from the Collection interface.  
  
**Q29 When to use ArrayList and when to use LinkedList in application?**  
  
ArrayList has constant time search operation O(1) .Hence, ArrayList is preferred when there are more get() or search operation .  
  
Insertion , Deletion operations take constant time O(1) for LinkedList. Hence, LinkedList is preferred when there are more insertions or deletions involved in the application.  
  
  
**Q30 Write the code for iterating the list in different ways in java ?**  
There are two ways to iterate over the list in java :  
a. using Iterator  
b. using for-each loop  
**Q31 How HashSet works internally in java ?**  
  
This is one of the popular interview question .

HashSet internally uses HashMap to maintain the uniqueness of elements.

**Q32 What is CopyOnWriteArrayList ?  How it is different from  ArrayList in Java?**  
  
[CopyOnWriteArrayList](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) is a thread safe variant of ArrayList   in which all mutative operations like add , set are implemented by creating a fresh copy of the underlying array.  
It guaranteed not to throw ConcurrentModificationException.  
It permits all elements including null. It is introduced in jdk 1.5 .

**Q35 What is BlockingQueue in Java Collections Framework?**  
  
[BlockingQueue](https://docs.oracle.com/javase/6/docs/api/java/util/concurrent/BlockingQueue.html) implements the java.util.Queue interface . BlockingQueue 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 .  
It does not accept null elements.  
Blocking queues are primarily designed for the producer-consumer problems.  
BlockingQueue implementations are thread-safe and can also be used in inter-thread communications.  
This concurrent Collection class was added in jdk 1.5

**Q36 How TreeMap works in Java ?**  
  
TreeMap internally uses Red-Black tree to sort the elements in natural order.

**Q37 All the questions related to HashSet class can be found here** ,  [frequently asked HashSet interview questions](http://javahungry.blogspot.co.uk/2014/04/top-10-hashset-java-interview-questions-collection.html)  
  
**Q40 How do you use a custom object as key in Collection  classes like HashMap ?**  
  
If one is using the custom object as key then one needs to override equals() and hashCode() method  
and one also need to fulfill the contract.  
If you want to store the custom object in the SortedCollections like SortedMap then one needs to make sure that equals() method is consistent to the compareTo() method. If inconsistent , then collection will not follow their contracts ,that is , Sets may allow duplicate elements.

**Q41 What is hash-collision in Hashtable ? How it was handled in Java?**  
  
In Hashtable , if two different keys have the same hash value then it lead to hash -collision. A bucket of type linkedlist used to hold the different keys of same hash value.  
  
**Q42 Explain the importance of hashCode() and equals() method ? Explain the contract also ?**  
HashMap object uses Key object hashCode() method and equals() method to find out the index to put the key-value pair. If we want to get value from the HashMap same both methods are used . Somehow, if both methods are not implemented correctly , it will result in two keys producing the same hashCode() and equals() output. The problem will arise that HashMap will treat both output same instead of different and overwrite the most recent key-value pair with the previous key-value pair.  
Similarly all the collection classes that does not allow the duplicate values use hashCode() and equals() method to find the duplicate elements.So it is very important to implement them correctly.  
  
**Contract of hashCode() and equals() method**  
a.If  object1.equals(object2) , then  object1.hashCode() == object2.hashCode() should always be true.  
  
b. If object1.hashCode() == object2.hashCode() is true does not guarantee object1.equals(object2)

**Q48 How will you make Collections readOnly ?**

We can make the Collection readOnly by using the following lines code:

General : Collections.unmodifiableCollection(Collection c)  
  
Collections.unmodifiableMap(Map m)

Collections.unmodifiableList(List l)

Collections.unmodifiableSet(Set s)

**Q50 Suppose there is an Employee class. We add Employee class objects to the ArrayList. Mention the steps need to be taken , if I want to sort the objects in ArrayList using the employeeId attribute present  in Employee class.**  
a. Implement the Comparable interface for the Employee class and now to compare the objects by employeeId we will override the emp1.compareTo(emp2)  
b. We will now call Collections class sort method and pass the list as argument , that is ,  
     Collections.sort(empList)

**4. How do you remove an entry from a Collection? and subsequently what is the difference between the remove() method of Collection and remove() method of Iterator, which one you will use while removing elements during iteration?**  
  
Collection interface defines remove(Object obj) method to remove objects from Collection. List interface adds another method remove(int index), which is used to remove object at specific index. You can use any of these method to remove an entry from Collection, while not iterating. Things change, when you iterate. Suppose you are traversing a List and removing only certain elements based on logic, then you need to use Iterator's remove() method. This method removes current element from Iterator's perspective. If you use Collection's or List's remove() method during iteration then your code will throw ConcurrentModificationException. That's why it's advised to use Iterator remove() method to remove objects from Collection.

**11. What is the difference between Set and List in Java? (**[**answer**](http://java67.blogspot.com/2012/08/difference-between-list-and-set-in-java.html)**)**

Another classical Java Collection interviews popular on telephonic round or the first round of interview. Most of Java programmer knows that Set doesn't allowed duplicate while List does and List maintains insertion order while Set doesn't. What is key here is to show the interviewer that you can decide which collection is more suited based on requirements.

**12. How do you Sort objects on the collection? (**[**solution**](http://java67.blogspot.com/2012/07/sort-list-ascending-descending-order-set-arraylist.html)**)**

This Collection interview question serves two purpose it not only test an important programming concept Sorting but also utility class like Collections which provide several methods for creating synchronized collection and sorting. **Sorting is implemented using Comparable and Comparator in Java and when you call Collections.sort() it gets sorted based on the natural order specified in compareTo() method while Collections.sort(Comparator) will sort objects based on compare() method of Comparator.**

**13. What is the difference between Vector and ArrayList? (**[**answer**](http://java67.blogspot.com/2012/09/arraylist-vs-vector-in-java-interview.html)**)**

[ArrayList in Java](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html) is one of the most used Collection class and the most interviewers asked questions on ArrayList. See Difference between Vector and ArrayList for the answer to this interview question.

**14. What is the difference between HashMap and HashSet? (**[**answer**](http://java67.blogspot.com/2012/08/difference-between-hashset-and-hashmap.html)**)**

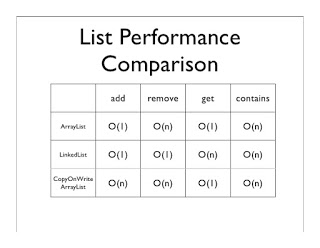
This collection interview questions is asked in conjunction with HashMap vs Hashtable. HashSet implements java.util.Set interface and that's why only contains unique elements, while HashMap allows duplicate values.  In fact, HashSet is actually implemented on top of java.util.HashMap. If you look internal implementation of java.util.HashSet, you will find that it adds element as key on internal map with same values. For a more detailed answer, see [HashMap vs HashSet](http://javarevisited.blogspot.com/2011/09/difference-hashmap-vs-hashset-java.html).  
  
  
  
**15) What is NavigableMap in Java? What is a benefit over Map? (**[**answer**](http://javarevisited.blogspot.com/2013/01/what-is-navigablemap-in-java-6-example-submap-head-tail.html)**)**  
NavigableMap Map was added in Java 1.6, it adds navigation capability to Map data structure. It provides methods like lowerKey() to get keys which is less than specified key, floorKey() to return keys which is less than or equal to specified key, ceilingKey() to get keys which is greater than or equal to specified key and higherKey() to return keys which is greater specified key from a Map. It also provide similar methods to get entries e.g. lowerEntry(), floorEntry(), ceilingEntry() and higherEntry(). Apart from navigation methods, it also provides utilities to create sub-Map e.g. creating a Map from entries of an exsiting Map like tailMap, headMap and subMap. headMap() method returns a NavigableMap whose keys are less than specified, tailMap() returns a NavigableMap whose keys are greater than the specified and subMap() gives a NavigableMap between a range, specified by toKey to fromKey.    
  
  
  
**16) Which one you will prefer between Array and ArrayList for Storing object and why? (**[**answer**](http://java67.blogspot.com/2012/12/difference-between-array-vs-arraylist-java.html)**)**  
Though ArrayList is also backed up by array, it offers some usability advantage over array in Java. Array is fixed length data structure, once created you can not change it's length. On the other hand, ArrayList is dynamic, it automatically allocate a new array and copies content of old array, when it resize. Another reason of using ArrayList over Array is support of Generics. Array doesn't support Generics, and if you store an Integer object on a String array, you will only going to know about it at runtime, when it throws ArrayStoreException. On the other hand, if you use ArrayList, compiler and IDE will catch those error on the spot. So if you know size in advance and you don't need re-sizing than use array, otherwise use ArrayList.

**17) Can we replace Hashtable with ConcurrentHashMap? (**[**answer**](http://java67.blogspot.com/2014/07/21-frequently-asked-java-interview-questions-answers.html)**)**

Answer 3: Yes we can replace Hashtable with ConcurrentHashMap and that's what suggested in Java documentation of ConcurrentHashMap. but you need to be careful with code which relies on locking behavior of Hashtable. Since Hashtable locks whole Map instead of a portion of Map, compound operations like if(Hashtable.get(key) == null) put(key, value) works in Hashtable but not in concurrentHashMap. instead of this use putIfAbsent() method of ConcurrentHashMap

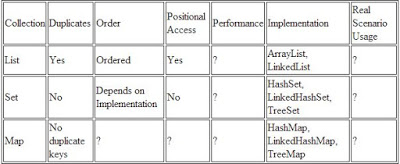
**18) What is CopyOnWriteArrayList, how it is different than ArrayList and Vector? (**[**answer**](http://java67.blogspot.com/2015/06/difference-between-synchronized-arraylist-and-copyOnWriteArrayList-java.html)**)**

Answer: CopyOnWriteArrayList is new List implementation introduced in Java 1.5 which provides better concurrent access than Synchronized List. better concurrency is achieved by Copying ArrayList over each write and replace with original instead of locking. Also CopyOnWriteArrayList doesn't throw any ConcurrentModification Exception. Its different than ArrayList because its thread-safe and ArrayList is not thread-safe and it's different than Vector in terms of Concurrency. CopyOnWriteArrayList provides better Concurrency by reducing contention among readers and writers. Here is a nice table which compares performance of three of popular List implementation ArrayList, LinkedList and CopyOnWriteArrayList in Java:

[](http://3.bp.blogspot.com/-C3omsD-5Dmk/VmQRsg6xYjI/AAAAAAAAEQk/Q2h35mRULLc/s1600/ArrayList+LinkedList+and+CopyOnWriteArrayList.jpg)

**21) Difference between Set, List and Map Collection classes? (**[**answer**](http://java67.blogspot.com/2013/01/difference-between-set-list-and-map-in-java.html)**)**

java.util.Set, java.util.List and java.util.Map defines three of most popular data structure support in Java. Set provides uniqueness guarantee i.e.g you can not store duplicate elements on it, but it's not ordered. On the other hand List is an ordered Collection and also allowes duplicates. Map is based on hashing and stores key and value in an Object called entry. It provides O(1) performance to get object, if you know keys, if there is no collision. Popular impelmentation of Set is HashSet, of List is ArrayList and LinkedList, and of Map are HashMap, Hashtable and ConcurrentHashMap. Another key difference between Set, List and Map are that Map doesn't implement Collection interface, while other two does. For a more detailed answer, see Set vs List vs Map in Java

[](http://3.bp.blogspot.com/-6H0yWpx3nEQ/VmQRFIW7GxI/AAAAAAAAEQM/cYFACp53rj0/s1600/List+vs+Map+vs+Set+in+Java.jpg)

**22) What is BlockingQueue, how it is different than other collection classes? (**[**answer**](http://javarevisited.blogspot.com/2012/12/blocking-queue-in-java-example-ArrayBlockingQueue-LinkedBlockingQueue.html)**)**

BlockingQueue is a Queue implementation available in java.util.concurrent package. It's one of the concurrent Collection class added on Java 1.5, main difference between BlockingQueue and other collection classes is that apart from storage, it also provides flow control. It can be used in inter-thread communication and also provides built-in thread-safety by using happens-before guarantee. You can use BlockingQueue to solve Producer Consumer problem, which is what is needed in most of concurrent applications.

**Q1) What is difference between an ArrayList and a vector?**

Ans)

* Synchronization - ArrayList is not thread-safe whereas Vector is thread-safe. In Vector class each method like add(), get(int i) is surrounded with a synchronized block, thus making Vector class thread-safe.
* Data growth - Internally, both the ArrayList and Vector hold onto their contents using an Array. When an element is inserted into an ArrayList or a Vector, the object will need to expand its internal array if it runs out of room. A Vector defaults to doubling the size of its array, while the ArrayList increases its array size by 50 percent.
* Performance - Since vector is thread-safe, the performance is slower than ArrayList.

**Q2) How can an Arraylist be synchronized without using Vector?**

Ans) Arraylist can be synchronized using:

Collections.synchronizedList(List list)

Other collections can be synchronized:

Collections.synchronizedMap(Map map)

Collections.synchronizedCollection(Collection c)

**Q3) If an Employee class is present and its objects are added in an arrayList. Now I want the list to be sorted on the basis of the employeeID of Employee class. What are the steps?**

Ans)

* Implement Comparable interface for the Employee class and override the compareTo(Object obj) method in which compare the employeeID
* Now call Collections.sort() method and pass the list as an argument.

*Now consider that Employee class is a jar file.*

* 1) Since Comparable interface cannot be implemented, create Comparator and override the compare(Object obj, Object obj1) method .
* 2) Call Collections.sort() on the list and pass comparator as an argument.

**Q4)What is difference between a HashMap and a HashTable?**

Ans) Both collections implements Map. Both collections store value as key-value pairs. The key differences between the two are:

1. Hashmap is not synchronized in nature but hashtable is.
2. Another difference is that iterator in the HashMap is fail-safe while the enumerator for the Hashtable isn't.**Fail-safe** -if the Hashtable is structurally modified at any time after the iterator is created, in any way except through the iterator's own remove method, the iterator will throw a ConcurrentModificationException?
3. HashMap permits null values and only one null key, while Hashtable doesn't allow key or value as null.

**Q5) What are the classes implementing the List interface?**

Ans) There are three implementation of List interface:

1. **ArrayList** : It is a resizable array implementation. The size of the ArrayList can be increased dynamically also operations like add,remove and get can be formed once the object is created. It also ensures that the data is retrieved in the manner it was stored. The ArrayList is not thread-safe.
2. **Vector**: It is thread-safe implementation of ArrayList. The methods are wrapped around a synchronized block.
3. **LinkedList**: the LinkedList implements Queue interface too and provide FIFO (First In First Out) operation for add operation. It is faster than ArrayList if its mainly used forinsertion and deletion of elements.

**Q6) Which all classes implement Set interface ?**

Ans) A Set is a collection that contains no duplicate elements. More formally, a set contains no pair of elements e1 and e2 such that e1.equals(e2), and at most one null element. **HashSet,SortedSet and TreeSet** are the commonly used class which implements Set interface.

* **SortedSet** - It is an interface which extends Set. A the name suggest, the interface allows the data to be iterated in the ascending order or sorted on the basis of Comparator or Comparable interface. All elements inserted into the interface must implement Comparable or Comparator interface.
* **TreeSet** - It is the implementation of SortedSet interface. This implementation provides guaranteed log(n) time cost for the basic operations (add, remove and contains). The class is not synchronized. The class uses Red-Black tree data structure.
* **HashSet:** This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element. This class offers constant time performance for the basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets

**Q7) What is difference between List and a Set?**

Ans)

1. List can contain duplicate values but Set doesn't allow.
2. List allows retrieval of data to be in same order in the way it is inserted but Set doesnt ensures the sequence in which data can be retrieved.(Except HashSet)

**Q8) What is difference between Arrays and ArrayList ?**

Ans)

* Arrays are created of fix size whereas ArrayList is dynamic in nature and can vary its length. Also the size of array cannot be incremented or decremented. But with arrayList the size is variable.
* Once the array is created elements cannot be added or deleted from it. But with ArrayList the elements can be added and deleted at runtime.
* List list = new ArrayList();
* list.add(1);
* list.add(3);

list.remove(0) // will remove the element from the 1st location.

* ArrayList is one dimensional but array can be multidimensional.

int[][][] intArray= new int[3][2][1]; // 3 dimensional array

* Array can contain objects of a single data type or class. ArrayList if not used with generic can contain objects of different classes

**Q9) When to use ArrayList or LinkedList ?**

Ans)

1. Adding new elements is pretty fast for either type of list. Inserting element to nth location in arraylist and to first location in linkedlist takes O(1).
2. For the ArrayList, doing random lookup using "get" is faster O(1), but for LinkedList O(n), it's slow. It's slow because there's no efficient way to index into the middle of a linked list. Linkedlist lookup always start from 1st location.
3. When removing elements, using ArrayList is slow. This is because all remaining elements in the underlying array of Object instances must be shifted down for each remove operation. But LinkedList is fast, because deletion can be done simply by changing a couple of links.

So an ArrayList works best for cases where you're doing random access on the list and a LinkedList works better if you're doing a lot of editing in the middle of the list.

Source : [Read More - from java.sun](http://java.sun.com/developer/TechTips/1999/tt0809.html)

**Q11) What are advantages of iterating a collection using iterator?**

Ans) For loop does not allow updating the colection(add or remove) whereas Iterator does. Also Iterator can be used where there is no clue what type of collections will be used because all collections implement Iterator interface.

#### **Limitations of Arrays:**

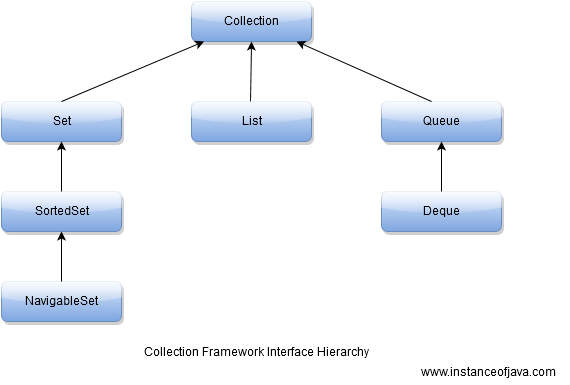
* Arrays are fixed in size. need to estimate the size of an array while declaration itself. once array created we can not increase the size of an array.
* Arrays can hold only homogeneous data elements. Means we can add same type of elements in an array. While declaring an array we need to mention the data type.
* int a[]= new int[10];
* By using Object array we can add heterogeneous elements to the array.
* Object o= new Object[10];
* There is no underlying data structure for arrays.
* Arrays are not recommended to use with respect to memory.
* Performance wise arrays are good to use.

#### **Collections**

* Java Collection framework added in J2SE 1.2 release.
* Collections are set of classes and interfaces.
* By using Collections we can store and manipulate objects easily.
* Collections can hold heterogeneous data elements.
* Collections are no fixed in size and dynamically increase in size.
* Collection of objects. No primitives.
* All collection classes having underlying data structure.
* Collections are good with respect to memory. Bad with respect to performance.

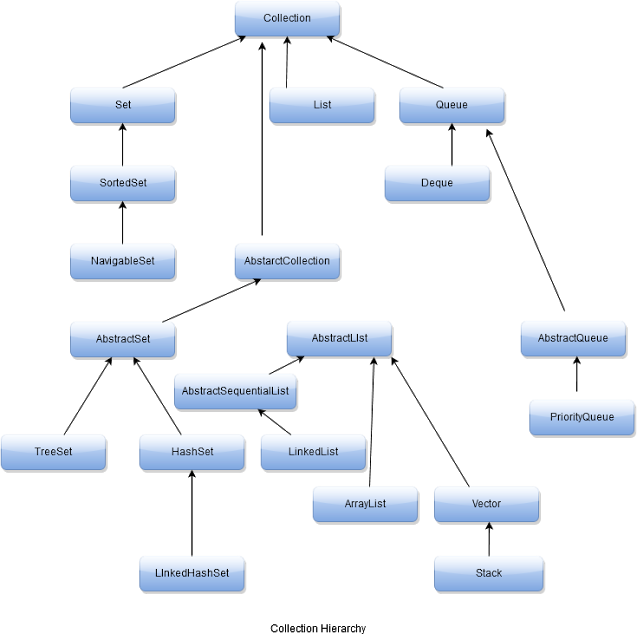
#### **Collection interface Hierarchy:**

* java.util package contains all collections classes and interfaces.
* Lets see collection interface hierarchy.
* under Collection. Set , List , Queue are the sub interfaces.

[](http://3.bp.blogspot.com/-VLI7AOnPWtE/VdmZSNA8eWI/AAAAAAAAAco/EmXs2GbyqAk/s1600/instanceofjavainterface2.png)

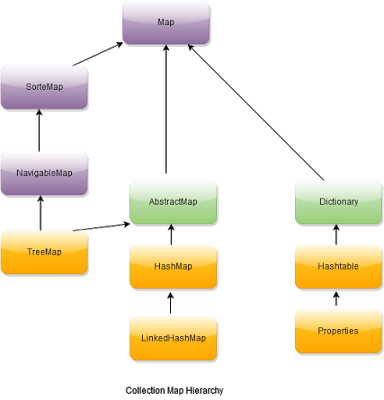
#### **Collections Abstract classes and classes:**

* Let us see  all the abstract classes implementing all the above interfaces and classes which extending these abstract classes.
* To Collect Objects in array format we choose Collection hierarchy classes.
* Main abstract class is AbstractCollection.
* AbstractSet
* AbstractList
* AbstractQueue
* AbstractSequentialList
* All the classes in Collection hierarchy are
* TreeSet
* HashSet
* LinkedHashSet
* LinkedList
* ArrayList
* Vector
* Stack
* PriorityQueue
* To collect unique elements we must choose Set implemented classes
* To collect unique and duplicate elements in indexed order we choose List implemented classes.
* To retrieve elements in FIFO manner we choose Queue implemented classes.

[](http://1.bp.blogspot.com/-SobxEfNQb88/Vdm5p5lSrxI/AAAAAAAAAc0/rkm8XWysTeY/s1600/instanceofjavacollectionmin.png)

#### **Map Hierarchy:**

* In this hierarchy Hashtable and properties classes are avilable since java 1.0.
* LinkedHashMap class is available since java 1.4
* NavigableMap is available since java 6 and all other classes available since java 1.2.
* SortedMap and NavigableMap are two main interfaces.
* TreeMap
* HashMap
* LinkedHashMap
* Hashtable
* Properties are the classes.
* To collect objects in key, value pair format we choose Map hierarchy classes.

[](http://2.bp.blogspot.com/-8W0Xt0rYilA/VdoNTw4MG6I/AAAAAAAAAdI/CC1NjDyucCQ/s1600/map.png)

* Collection is the main interface.
* There are many methods declared in Collection interface. These are the common methods for all collections to perform different operations.

1. public interface Collection<E>
2. extends Iterable<E>

#### **Methods in Collection Interface:**

 1.public Boolean add(Object obj)

* Used to add element in to the collection

 2.public Boolean addAll(Collection c)

* Adds all the elements of c to the invoking collection. Returns true if the operation succeeded else returns false.

 3.public boolean isEmpty()

* Returns true if collection is empty otherwise returns false.

 4.public boolean remove(Object obj)

* Remove element from collection. Returns true if successful deletion.

5.public boolean  removeAll(Collection c)

* Removes all elements of a collection.

 6.public void clear()

* Deletes total elements from collection

 7.public boolean contains(Object obj)

* This method used to search an element

 8.public boolean containsAll(Collection c)

* This method used to search an element in a collection

 9.public boolean retianAll(Collection c)

* Used to delete all elements from a collection except specified one.

10.public int size()

* Returns total numbers of elements in a collection

11.public Iterator iterator()

* Returns iterator for the collection

12.public boolean equals(Object obj)

* Compares two collections

13.public int hashCode()

* Returns hashcode

14.public Object[] toArray()

* This method used to convert collection into array

15.public Object[] toArray(Object[] obj)

* Returns an array containing only those collection elements whose type matches that of array.

#### **Difference Between Collection and Collections:**

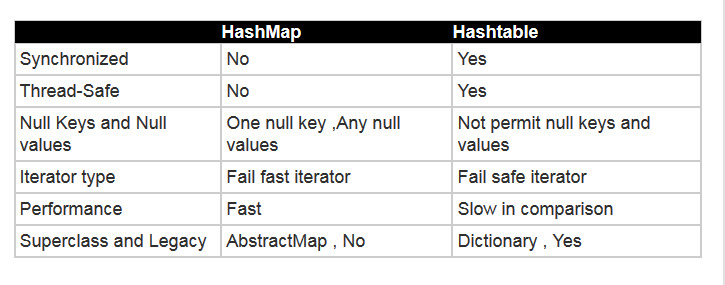
* Collection is the base interface for most of the classes.
* Collections is the utility class.

#### **1.Why Map interface doesn’t extend Collection interface?**

* Set is unordered collection and does not allows duplicate elements.
* List is ordered collection allows duplicate elements.
* Where as Map is key-value pair.
* It is viewed as set of keys and collection of values.
* Map is a collection of key value pairs so by design they separated from collection interface.

#### **2.What is difference between HashMap and Hashtable?**

* Synchronization or Thread Safe
* Null keys and null values
* Iterating the values
* Default Capacity

[](https://2.bp.blogspot.com/-3WTfMcf4CG0/VwX2BAuSxZI/AAAAAAAAAv0/8rX_HZvjVKgZLaWbeL0HVUJrv27GUwhtg/s1600/Hashmap+vs+hashtable.jpg)

#### **3.Differences between comparable and comparator?**

* Comparable Interface is actually from java.lang package.
* It will have a method compareTo(Object obj)to sort objects
* Comparator Interface is actually from java.util package.
* It will have a method compare(Object obj1, Object obj2)to sort objects

#### **4.How can we sort a list of Objects?**

* To sort the array of objects we will use  Arrays.sort() method.
* If we need to sort collection of object we will use Collections.sort().

#### **5.What is difference between fail-fast and fail-safe?**

* Fail fast is nothing but immediately report any failure. whenever a problem occurs fail fast system fails.
* in java Fail fast iterator while iterating through collection of objects sometimes concurrent modification exception will come there are two reasons for this.
* If one thread is iterating a collection and another thread trying to modify the collection.
* And after remove() method call if we try to modify collection object

#### 

#### [**What is difference between Iterator ,ListIterator and Enumeration?**](http://www.instanceofjava.com/2014/12/difference-between-enumeration-and.html)

#### **7.What is difference between Set and List in Java?**

* A set is a collection that allows unique elements.
* Set does not allow duplicate elements
* Set allows only one null value.
* Set having classes like :
* HashSet
* LinkedHashSet
* TreeSet
* List having index. and ordered  collection
* List allows n number of null values.
* List will display Insertion order with index.
* List having classes like :
* Vector
* ArrayList
* LinkedList

#### **8.Differences between arraylist and vector?**

* Vector was introduced in  first version of java . that's the reason only vector is legacy class.
* ArrayList was introduced in java version1.2, as part of java collections framework.
* Vector is  synchronized**.**
* ArrayList is not synchronized.

Read more: [Differences between arraylist and vector](http://www.instanceofjava.com/2014/12/difference-between-arraylist-and-vector.html)

#### **9.What are the classes implementing List interface?**

* ArrayList
* LinkedList
* Vector

#### **10. Which all classes implement Set interface ?**

* HashSet
* LinkedHashSet
* TreeSet

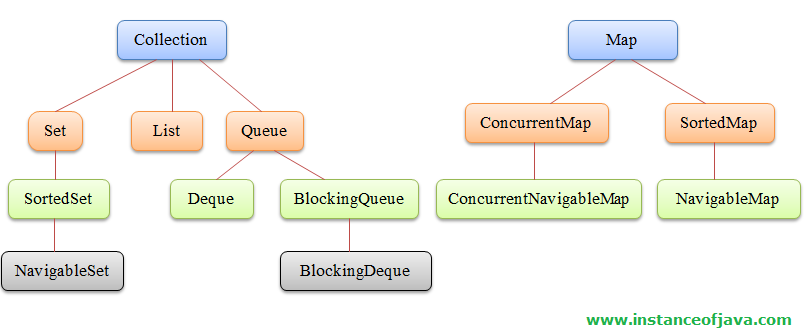
#### **11.How to make a collection thread safe?**

* Vector, Hashtable, Properties and Stack are synchronized classes, so they are thread-safe and can be used in multi-threaded environment.
* By using Collections.synchronizedList(list)) we can make list classes thread safe.
* By using    
  java.util.Collections.synchronizedSet()  we can make set classes thread safe.

#### **12.Can a null element added to a TreeSet or HashSet?**

* One null element can be added to hashset.
* TreeSet does not allow null values

#### **13. Explain Collection’s interface hierarchy?**

[](http://1.bp.blogspot.com/-lFxZCOe3z9M/VbBf7zrmlBI/AAAAAAAAAbI/ZlPRT6-cYVY/s1600/InterfacesHierarchy.png)

#### **14.Which design pattern Iterator follows?**

* Iterator design pattern

#### **15.Which data structure HashSet implements**

* Hashset implements hashmap internally.

#### **16.Why doesn't Collection extend Cloneable and Serializable?**

* List and Set and queue extends Collection interface.
* SortedMap extends Map interface.

#### **17.What is the importance of hashCode() and equals() methods? How they are used in Java?**

* equals() and hashcode() methods defined in "object" class.
* If equals() method return true on comparing two objects then hashcode() of those two objects must be same.

#### **18.What is difference between array & arraylist?**

* Array is collection of similar type of objects and fixed in size.
* Arraylist is collection of homogeneous and heterogeneous elements.

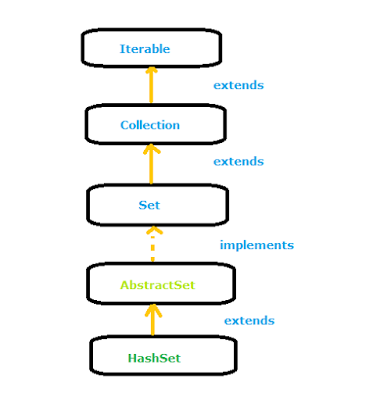
#### **19.What is the Properties class?**

* **Properties is a subclass of Hashtable. It is used to maintain lists of values in which the key and the value is String.**

#### **20.How to convert a string array to arraylist?**

* ArrayList al=new ArrayList( Arrays.asList( new String[]{"java", "collection"} ) );
* arrayList.toArray(); from list to array

#### **Hierarchy of HashSet class:**

[](http://4.bp.blogspot.com/-WKdWiQYncUs/VlAcEuLqJTI/AAAAAAAAAiA/RNRYP4n_5ac/s1600/hashset.png)

#### 

**public class HashSet<E>**

**extends AbstractSet<E>   
           implements Set<E>, Cloneable, java.io.Serializable**

#### **Key points:**

* HashSet is sub class of AbstractSet class.
* Underlying data structure of HashSet is HashTable.
* HashSet implements Serializable and Cloneable interfaces
* HashSet does not allow duplicate elements.
* Insertion order is not preserved. No order(Based on hashcode of objects)

#### **Constructors of HashSet:**

**1.HashSet( )**

* Creates an empty HashSet object with default initial capacity 16.  Fill ratio or load factor 0.75

**2.HashSet(Collection obj)**

* This constructor initializes the hash set by using the elements of the collection **obj**.

**3.HashSet(int capacity)**

* Creates an empty HashSet object with given capacity.

**4.HashSet(int capacity, float fillRatio)**

* This constructor initializes both the capacity and the fill ratio (also called load capacity) of the hash set from its arguments (fill ratio 0.1 to 1.0)

**Basic java example program to get size of  hashset**

* **int size()**  This method is used get get size of  hashset

1. package com.getSizehashset;
3. import java.util.HashSet;
4. import java.util.Iterator;
6. public class HashsetExample{
8. public static void main(String[] args) {
10. //create object of HashSet
11. HashSet<Integer> hashSet = new HashSet();
13. //add elements to HashSet object
14. hashSet.add(1);
15. hashSet.add(2);
16. hashSet.add(3);
17. hashSet.add(4);
18. hashSet.add(5);
19. hashSet.add(6);
20. hashSet.add(7);
21. hashSet.add(8);
23. System.out.println("Size of HashSet after addition : " + hashSet.size());
25. System.out.println("Hashset contains");
27. Iterator it=hashSet.iterator();
29. while(it.hasNext()){
30. System.out.println(it.next());
32. }
34. }
36. }

**Output:**

1. Size of HashSet after addition
2. 8
3. hashset contains
4. 1
5. 2
6. 3
7. 4
8. 5
9. 6
10. 7
11. 8

**Basic java example program to check particular element is exists in hashset**

* **boolean contains(Object o)**  This method Returns true if this set contains the specified element

1. package com.checkelementhashset;
3. import java.util.HashSet;
4. import java.util.Iterator;
6. public class HashsetExample{
8. public static void main(String[] args) {
10. //create object of HashSet
11. HashSet<Integer> hashSet = new HashSet();
13. //add elements to HashSet object
14. hashSet.add(1);
15. hashSet.add(2);
16. hashSet.add(3);
17. hashSet.add(4);
18. hashSet.add(5);
20. /\*
21. To check whether a particular value exists in HashSetwe need to use
22. boolean contains(Object value) method of HashSet class.
23. this method returns true if the HashSet contains the value, otherwise returns false.
24. \*/
26. boolean isExists = hashSet.contains(5);
27. System.out.println("5 exists in HashSet ? : " + isExists);

30. Iterator it=hashSet.iterator();
32. while(it.hasNext()){
33. System.out.println(it.next());
35. }
37. }
39. }

**Output:**

1. 3 exists in HashSet ? : true
2. 1
3. 2
4. 3
5. 4
6. 5

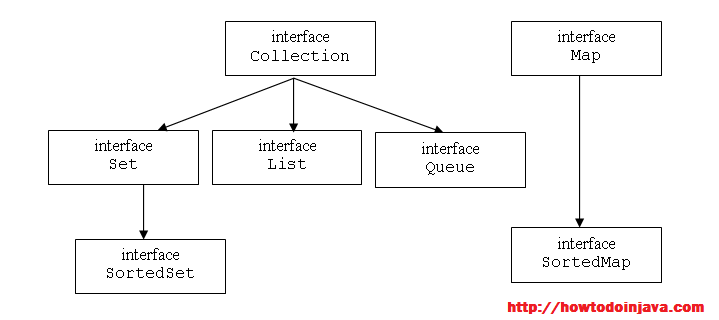
##### 1) What is the Java Collection framework? List down its advantages?

By definition, a collection is **an object that represents a group of objects**. Like in set theory, a set is group of elements. Easy enough !!  
Prior to JDK 1.2, JDK has some utility classes such as Vector and HashTable, but there was no concept of Collection framework. Later from JDK 1.2 onwards, JDK felt the need of having a consistent support for reusable data structures. Finally, the collections framework was designed and developed primarily by Joshua Bloch, and was **introduced in JDK 1.2**.

Its most **noticeable advantages** can be listed as:

* Reduced programming effort due to ready to use code
* Increased performance because of high-performance implementations of data structures and algorithms
* Provides interoperability between unrelated APIs by establishing a common language to pass collections back and forth
* Easy to learn APIs by learning only some top level interfaces and supported operations

##### 2) Explain Collection’s hierarchy?

Java Collection Hierarchy

As shown in above image, collection framework has one interface at top i.e. **Collection**. It is **extended by Set, List and Queue interfaces**. Then there are loads of other classes in these 3 branches which we will learn in following questions.

Remember the signature of Collection interface. It will help you in many question.

|  |
| --- |
| public interface Collection extends Iterable {  //method definitions  } |

Framework also consist of Map interface, which is part of collection framework. but it does not extend Collection interface. We will see the reason in 4th question in this question bank.

##### 3) Why Collection interface does not extend Cloneable and Serializable interface?

Well, simplest answer is “**there is no need to do it**“. Extending an interface simply means that you are creating a subtype of interface, in other words a more specialized behavior and Collection interface is not expected to do what Cloneable and Serializable interfaces do.

Another reason is that not everybody will have a reason to have Cloneable collection because if it has very large data, then every **unnecessary clone operation will consume a big memory**. Beginners might use it without knowing the consequences.

Another reason is that **Cloneable and Serializable are very specialized behavior** and so should be implemented only when required. For example, many concrete classes in collection implement these interfaces. So if you want this feature. use these collection classes otherwise use their alternative classes.

##### 4) Why Map interface does not extend Collection interface?

A good answer to this interview question is “**because they are incompatible**“. Collection has a method add(Object o). Map can not have such method because it need key-value pair. There are other reasons also such as Map supports keySet, valueSet etc. Collection classes does not have such views.

Due to such big differences, Collection interface was not used in Map interface, and it was build in separate hierarchy.

## **List interface related**

##### 5) Why we use List interface? What are main classes implementing List interface?

A java list is a **“ordered” collection of elements**. This ordering is a **zero based index**. It does not care about duplicates. Apart from methods defined in Collection interface, it does **have its own methods** also which are largely to manipulate the collection **based on index location of element**. These methods can be grouped as search, get, iteration and range view. All above operations support index locations.

The main classes implementing List interface are: **Stack, Vector, ArrayList and LinkedList**. Read more about them in java documentation.

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

This is more of a programmatic question which is seen at beginner level. The intent is to check the knowledge of applicant in Collection utility classes. For now, lets learn that there are two utility classes in Collection framework which are mostly seen in interviews i.e. **Collections and Arrays**.

Collections class provides some static functions to perform specific operations on collection types. And Arrays provide utility functions to be performed on array types.

|  |
| --- |
| //String array  String[] words = {"ace", "boom", "crew", "dog", "eon"};  //Use Arrays utility class  List wordList = Arrays.asList(words);  //Now you can iterate over the list |

Please not that this function is not specific to String class, it will return List of element of any type, of which the array is. e.g.

|  |
| --- |
| //String array  Integer[] nums = {1,2,3,4};  //Use Arrays utility class  List numsList = Arrays.asList(nums); |

##### 7) How to reverse the list?

This question is just like above to test your knowledge of **Collections** utility class. Use it **reverse()** method to reverse the list.

|  |
| --- |
| Collections.reverse(list); |

## **Set interface related**

##### 8) Why we use Set interface? What are main classes implementing Set interface?

It **models the mathematical set in set theory**. Set interface is like List interface but with some differences. First, it is **not ordered collection**. So no ordering is preserved while adding or removing elements. The main feature it does provide is “**uniqueness of elements**“. It does not support duplicate elements.

Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ. Two Set instances are equal if they contain the same elements.

Based on above reasons, it **does not have operations based on indexes of elements like List**. It only has methods which are inherited by Collection interface.

Main classes implementing Set interface are :**EnumSet, HashSet, LinkedHashSet, TreeSet**. Read more on related java documentation.

##### 9) How HashSet store elements?

You must know that HashMap store key-value pairs, with one condition i.e. keys will be unique. HashSet uses Map’s this feature to ensure uniqueness of elements. In HashSet class, a map declaration is as below:

|  |
| --- |
| private transient HashMap<E,Object> map;    //This is added as value for each key  private static final Object PRESENT = new Object(); |

So **when you store a element in HashSet, it stores the element as key in map and “PRESENT” object as value**. (See declaration above).

|  |
| --- |
| public boolean add(E e) {  return map.put(e, PRESENT)==null;  } |

I will highly suggest you to read this post: [**How HashMap works in java?**](http://howtodoinjava.com/core-java/collections/how-hashmap-works-in-java/) This post will help you in answering all the HashMap related questions very easily.

##### 10) Can a null element added to a TreeSet or HashSet?

As you see, There is no null check in add() method in previous question. And HashMap also allows one null key, so **one “null” is allowed in HashSet**.

TreeSet uses the same concept as HashSet for internal logic, but uses NavigableMap for storing the elements.

|  |
| --- |
| private transient NavigableMap<E,Object> m;    // Dummy value to associate with an Object in the backing Map  private static final Object PRESENT = new Object(); |

NavigableMap is subtype of SortedMap which does not allow null keys. So essentially,**TreeSet also does not support null keys**. It will throw NullPointerException if you try to add null element in TreeSet.

## **Map interface related**

##### 11) Why we use Map interface? What are main classes implementing Map interface?

Map interface is a special type of collection which is **used to store key-value pairs**. It does not extend Collection interface for this reason. This interface provides methods to add, remove, search or iterate over various views of Map.

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

##### 12) 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** 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.

##### 13) Explain ConcurrentHashMap? How it works?

Taking from java docs:

**A hash table supporting full concurrency of retrievals and adjustable expected concurrency for updates**. This class obeys the same functional specification as Hashtable, and includes versions of methods corresponding to each method of Hashtable. However, even though all operations are thread-safe, retrieval operations do not entail locking, and there is not any support for locking the entire table in a way that prevents all access. This class is fully interoperable with Hashtable in programs that rely on its thread safety but not on its synchronization details.

Read more about how [**concurrent hashmap works and related interview questions**](http://howtodoinjava.com/core-java/collections/popular-hashmap-and-concurrenthashmap-interview-questions/).

##### 14) How hashmap works?

The **most important question** which is most likely to be seen in every level of job interviews. You must be very clear on this topic., not only because it is most asked question but also it will open up your mind in further questions related to collection APIs.

Answer to this question is very large and you should read it my post: [**How HashMap works?**](http://howtodoinjava.com/core-java/collections/how-hashmap-works-in-java/) For now, lets remember that HashMap works **on principle of Hashing**. A map by definition is : “An object that maps keys to values”. To store such structure, **it uses an inner class Entry**:

|  |
| --- |
| static class Entry implements Map.Entry  {  final K key;  V value;  Entry next;  final int hash;  ...//More code goes here  } |

Here key and value variables are used to store key-value pairs. Whole entry object is stored in an array.

|  |
| --- |
| /\*\*  \* The table, re-sized as necessary. Length MUST Always be a power of two.  \*/  transient Entry[] table; |

The index of array is calculated on basis on hashcode of Key object. Read more of linked topic.

##### 15) How to design a good key for hashmap?

Another good question usually followed up after answering how hashmap works. Well, the most important constraint is **you must be able to fetch the value object back in future**. Otherwise, there is no use of having such a data structure. If you understand the working of hashmap, you will find it largely depends on hashCode() and equals() method of Key objects.

So a good key object**must provide same hashCode() again and again**, no matter how many times it is fetched. Similarly, same keys**must return true when compare with equals() method and different keys must return false**.

For this reason,**immutable classes are considered best candidate for HashMap keys**.

Read more : [**How to design a good key for HashMap?**](http://howtodoinjava.com/core-java/collections/how-to-design-a-good-key-for-hashmap/)

##### 16) 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.

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

HashMap is well known class and all of us know that. So, I will leave this part by saying that it is used to store key-value pairs and allows to perform many operations on such collection of pairs.

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.

## **Tell the difference questions**

##### 18) Difference between Set and List?

The most noticeable differences are :

* Set is unordered collection where List is ordered collection based on zero based index.
* List allow duplicate elements but Set does not allow duplicates.
* List does not prevent inserting null elements (as many you like), but Set will allow only one null element.

##### 19) Difference between List and Map?

Perhaps most easy question. **List is collection of elements where as map is collection of key-value pairs**. There is actually lots of differences which originate from first statement. They have**separate top level interface, separate set of generic methods, different supported methods and different views of collection**.

I will take much time hear as answer to this question is enough as first difference only.

##### 20) Difference between HashMap and HashTable?

There are several differences between HashMap and Hashtable in Java:

* Hashtable is synchronized, whereas HashMap is not.
* Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.
* The third significant difference between HashMap vs Hashtable is that Iterator in the HashMap is a fail-fast iterator while the enumerator for the Hashtable is not.

##### 21) Difference between Vector and ArrayList?

Lets note down the differences:

* All the methods of Vector is synchronized. But, the methods of ArrayList is not synchronized.
* Vector is a Legacy class added in first release of JDK. ArrayList was part of JDK 1.2, when collection framework was introduced in java.
* By default, Vector doubles the size of its array when it is re-sized internally. But, ArrayList increases by half of its size when it is re-sized.

##### 22) 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.

##### 23) 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.

##### 24) 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.

##### 25) Difference between TreeSet and SortedSet?

SortedSet is an interface which TreeSet implements. That’ it !!

##### 26) 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.

## **More questions**

##### 27) How to make a collection read only?

Use following methods:

* Collections.unmodifiableList(list);
* Collections.unmodifiableSet(set);
* Collections.unmodifiableMap(map);

These methods takes collection parameter and return a new read-only collection with same elements as in original collection.

##### 28) How to make a collection thread safe?

Use below methods:

* Collections.synchronizedList(list);
* Collections.synchronizedSet(set);
* Collections.synchronizedMap(map);

Above methods take collection as parameter and return same type of collection which are synchronized and thread safe.

##### 29) Why there is not method like Iterator.add() to add elements to the collection?

The sole purpose of an Iterator is to enumerate through a collection. All collections contain the add() method to serve your purpose. There would be no point in adding to an Iterator because the **collection may or may not be ordered**. And **add() method can not have same implementation for ordered and unordered collections**.

##### 30) What are different ways to iterate over a list?

You can iterate over a list using following ways:

* Iterator loop
* For loop
* For loop (Advance)
* While loop

Read more : <http://www.mkyong.com/java/how-do-loop-iterate-a-list-in-java/>

##### 31) 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.

##### 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.

##### 33) How to avoid ConcurrentModificationException while iterating a collection?

You should first try to **find another alternative iterator which are fail-safe**. For example if you are using List and you can use ListIterator. If it is legacy collection, you can use enumeration.

If above options are not possible then you can use one of three changes:

* If you are using JDK1.5 or higher then you can use ConcurrentHashMap and CopyOnWriteArrayList classes. It is the recommended approach.
* You can convert the list to an array and then iterate on the array.
* You can lock the list while iterating by putting it in a synchronized block.

Please note that last two approaches will cause a performance hit.

##### 34) What is UnsupportedOperationException?

This exception is thrown **on invoked methods which are not supported by actual collection type**.

For example, if you make a read-only list list using “Collections.unmodifiableList(list)” and then call add() or remove() method, what should happen. It should clearly throw UnsupportedOperationException.

##### 35) Which collection classes provide random access of it’s elements?

ArrayList, HashMap, TreeMap, Hashtable classes provide random access to it’s elements.

##### 36) 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.

Read the example usage of blocking queue in post : [**How to use blocking queue?**](http://howtodoinjava.com/java-5/how-to-use-blockingqueue-and-threadpoolexecutor-in-java/)

##### 37) What is Queue and Stack, list down their differences?

**A collection designed for holding elements prior to processing.** Besides basic Collection operations, queues provide additional insertion, extraction, and inspection operations.  
**Queues typically, but do not necessarily, order elements in a FIFO (first-in-first-out) manner.**

**Stack is also a form of Queue but one difference, it is LIFO (last-in-first-out).**

Whatever the ordering used, the head of the queue is that element which would be removed by a call to remove() or poll(). Also note that Stack and Vector are both synchronized.

**Usage:** Use a queue if you want to process a stream of incoming items in the order that they are received.Good for work lists and handling requests.  
Use a stack if you want to push and pop from the top of the stack only. Good for recursive algorithms.

##### 38) 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.

##### 39) What are Collections and Arrays classes?

**Collections and Arrays classes are special utility classes to support collection framework core classes.** They provide utility functions to get read-only/ synchronized collections, sort the collection on various ways etc.

Arrays also helps array of objects to convert in collection objects. Arrays also have some functions which helps in copying or working in part of array objects.

1. **What are Collection related features in Java 8?**

Java 8 has brought major changes in the Collection API. Some of the changes are:

* 1. [Java Stream API](http://www.journaldev.com/2774/java-8-stream) for collection classes for supporting sequential as well as parallel processing
  2. [Iterable interface is extended with forEach()](http://www.journaldev.com/2389/java-8-features-with-examples#iterable-forEach) default method that we can use to iterate over a collection. It is very helpful when used with [lambda expressions](http://www.journaldev.com/2763/java-8-functional-interfaces) because it’s argument Consumer is a [function interface](http://www.journaldev.com/2763/java-8-functional-interfaces).
  3. Miscellaneous Collection API improvements such as forEachRemaining(Consumer action)method in Iterator interface, Map replaceAll(), compute(), merge() methods.

1. **What is Java Collections Framework? List out some benefits of Collections framework?**

Collections are used in every programming language and initial java release contained few classes for collections: **Vector**, **Stack**, **Hashtable**, **Array**. But looking at the larger scope and usage, Java 1.2 came up with Collections Framework that group all the collections interfaces, implementations and algorithms.  
Java Collections have come through a long way with usage of Generics and Concurrent Collection classes for thread-safe operations. It also includes blocking interfaces and their implementations in java concurrent package.  
Some of the benefits of collections framework are;

* 1. Reduced development effort by using core collection classes rather than implementing our own collection classes.
  2. Code quality is enhanced with the use of well tested collections framework classes.
  3. Reduced effort for code maintenance by using collection classes shipped with JDK.
  4. Reusability and Interoperability

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. I would highly recommend to go through [**Java Generic Tutorial**](http://www.journaldev.com/1663/java-generics-example-method-class-interface) to understand generics in a better way.

1. **What are the basic interfaces of Java Collections Framework?**

[Collection](http://www.journaldev.com/1260/collections-in-java-tutorial#collection-interface) is the root of the collection hierarchy. A collection represents a group of objects known as its elements. The Java platform doesn’t provide any direct implementations of this interface.

[Set](http://www.journaldev.com/1260/collections-in-java-tutorial#set-interface) is a collection that cannot contain duplicate elements. This interface models the mathematical set abstraction and is used to represent sets, such as the deck of cards.

[List](http://www.journaldev.com/1260/collections-in-java-tutorial#list-interface) is an ordered collection and can contain duplicate elements. You can access any element from it’s index. List is more like array with dynamic length.

A [Map](http://www.journaldev.com/1260/collections-in-java-tutorial#map-interface) is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.

Some other interfaces are [Queue](http://www.journaldev.com/1260/collections-in-java-tutorial#queue-interface), [Dequeue](http://www.journaldev.com/1260/collections-in-java-tutorial#dequeue-interface), [Iterator](http://www.journaldev.com/1260/collections-in-java-tutorial#iterator-interface), [SortedSet](http://www.journaldev.com/1260/collections-in-java-tutorial#sortedset-interface), [SortedMap](http://www.journaldev.com/1260/collections-in-java-tutorial#sortedmap-interface) and [ListIterator](http://www.journaldev.com/1260/collections-in-java-tutorial#listiterator-interface).

1. **Why Collection doesn’t extend Cloneable and Serializable interfaces?**

Collection interface specifies group of Objects known as elements. How the elements are maintained is left up to the concrete implementations of Collection. For example, some Collection implementations like List allow duplicate elements whereas other implementations like Set don’t.  
A lot of the Collection implementations have a public clone method. However, it does’t really make sense to include it in all implementations of Collection. This is because Collection is an abstract representation. What matters is the implementation.  
The semantics and the implications of either cloning or serializing come into play when dealing with the actual implementation; so concrete implementation should decide how it should be cloned or serialized, or even if it can be cloned or serialized.  
So mandating cloning and serialization in all implementations is actually less flexible and more restrictive. The specific implementation should make the decision as to whether it can be cloned or serialized.

1. **Why Map interface doesn’t extend Collection interface?**

Although Map interface and it’s implementations are part of Collections Framework, Map are not collections and collections are not Map. Hence it doesn’t make sense for Map to extend Collection or vice versa.  
If Map extends Collection interface, then where are the elements? Map contains key-value pairs and it provides methods to retrieve list of Keys or values as Collection but it doesn’t fit into the “group of elements” paradigm.

1. **What is an Iterator?**

Iterator interface provides methods to iterate over any Collection. We can get iterator instance from a Collection using *iterator()* method. Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection during the iteration. Java Collection iterator provides a generic way for traversal through the elements of a collection and implements [**Iterator Design Pattern**](http://www.journaldev.com/1716/iterator-design-pattern-java).

1. **What is difference between Enumeration and Iterator interface?**

Enumeration is twice as fast as Iterator and uses very less memory. Enumeration is very basic and fits to basic needs. But Iterator is much safer as compared to Enumeration because it always denies other threads to modify the collection object which is being iterated by it.  
Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection that is not possible with Enumeration. Iterator method names have been improved to make it’s functionality clear.

1. **Why there is not method like Iterator.add() to add elements to the collection?**

The semantics are unclear, given that the contract for Iterator makes no guarantees about the order of iteration. Note, however, that ListIterator does provide an add operation, as it does guarantee the order of the iteration.

1. **Why Iterator don’t have a method to get next element directly without moving the cursor?**

It can be implemented on top of current Iterator interface but since it’s use will be rare, it doesn’t make sense to include it in the interface that everyone has to implement.

1. **What is different between Iterator and ListIterator?**
   1. We can use Iterator to traverse Set and List collections whereas ListIterator can be used with Lists only.
   2. Iterator can traverse in forward direction only whereas ListIterator can be used to traverse in both the directions.
   3. ListIterator inherits from Iterator interface and comes with extra functionalities like adding an element, replacing an element, getting index position for previous and next elements.
2. **What are different ways to iterate over a list?**

We can iterate over a list in two different ways – using iterator and using for-each loop.

List<String> strList = new ArrayList<>();

//using for-each loop

for(String obj : strList){

System.out.println(obj);

}

//using iterator

Iterator<String> it = strList.iterator();

while(it.hasNext()){

String obj = it.next();

System.out.println(obj);

}

Using iterator is more thread-safe because it makes sure that if underlying list elements are modified, it will throw ConcurrentModificationException.

1. **What do you understand by iterator fail-fast property?**

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.

1. **What is difference between fail-fast and fail-safe?**

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/copyonwritearraylist-java).

1. **How to avoid ConcurrentModificationException while iterating a collection?**

We can use concurrent collection classes to avoid ConcurrentModificationException while iterating over a collection, for example CopyOnWriteArrayList instead of ArrayList.  
Check this post for [ConcurrentHashMap Example](http://www.journaldev.com/122/java-concurrenthashmap-example-iterator).

1. **Why there are no concrete implementations of Iterator interface?**

Iterator interface declare methods for iterating a collection but it’s implementation is responsibility of the Collection implementation classes. Every collection class that returns an iterator for traversing has it’s own Iterator implementation nested class.  
This allows collection classes to chose whether iterator is fail-fast or fail-safe. For example ArrayList iterator is fail-fast whereas CopyOnWriteArrayList iterator is fail-safe.

1. **What is UnsupportedOperationException?**

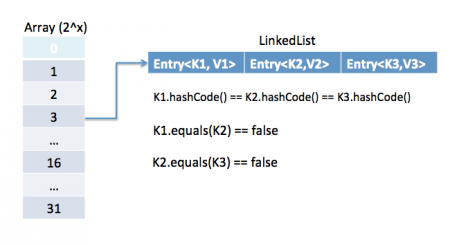
UnsupportedOperationException is the exception used to indicate that the operation is not supported. It’s used extensively in [JDK](http://www.journaldev.com/546/difference-jdk-vs-jre-vs-jvm) classes, in collections framework java.util.Collections.UnmodifiableCollection throws this exception for all add and removeoperations.

1. **How HashMap works in Java?**

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 get methods.

When we call put method 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.

[](http://cdn.journaldev.com/wp-content/uploads/2013/01/java-hashmap-entry-impl.png)

The other important things to know about HashMap are capacity, load factor, threshold resizing. HashMap initial default capacity is **16** and load factor is 0.75. Threshold is capacity multiplied by load factor and whenever we try to add an entry, if map size is greater than threshold, HashMap rehashes the contents of map into a new array with a larger capacity. The capacity is always power of 2, so if you know that you need to store a large number of key-value pairs, for example in caching data from database, it’s good idea to initialize the HashMap with correct capacity and load factor.

1. **What is the importance of hashCode() and equals() methods?**

HashMap uses Key object hashCode() and equals() method to determine the index to put the key-value pair. These methods are also used when we try to get value from HashMap. If these methods are not implemented correctly, two different Key’s might produce same hashCode() and equals() output and in that case rather than storing it at different location, HashMap will consider them same and overwrite them.

Similarly all the collection classes that doesn’t store duplicate data use hashCode() and equals() to find duplicates, so it’s very important to implement them correctly. The implementation of equals() and hashCode() should follow these rules.

* 1. If o1.equals(o2), then o1.hashCode() == o2.hashCode()should always be true.
  2. If o1.hashCode() == o2.hashCode is true, it doesn’t mean that o1.equals(o2) will be true.

1. **Can we use any class as Map key?**

We can use any class as Map Key, however following points should be considered before using them.

* 1. If the class overrides equals() method, it should also override hashCode() method.
  2. The class should follow the rules associated with equals() and hashCode() for all instances. Please refer earlier question for these rules.
  3. If a class field is not used in equals(), you should not use it in hashCode() method.
  4. Best practice for user defined key class is to make it immutable, so that hashCode() value can be cached for fast performance. Also immutable classes make sure that hashCode() and equals() will not change in future that will solve any issue with mutability.  
     For example, let’s say I have a class MyKey that I am using for HashMap key.
  5. //MyKey name argument passed is used for equals() and hashCode()
  6. MyKey key = new MyKey("Pankaj"); //assume hashCode=1234
  7. myHashMap.put(key, "Value");
  8. // Below code will change the key hashCode() and equals()
  9. // but it's location is not changed.
  10. key.setName("Amit"); //assume new hashCode=7890
  11. //below will return null, because HashMap will try to look for key
  12. //in the same index as it was stored but since key is mutated,
  13. //there will be no match and it will return null.

myHashMap.get(new MyKey("Pankaj"));

This is the reason why String and Integer are mostly used as HashMap keys.

1. **What are different Collection views provided by Map interface?**

Map interface provides three collection views:

* 1. **Set keySet()**: Returns a Set view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator’s own remove operation), the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Set.remove, removeAll, retainAll, and clear operations. It does not support the add or addAll operations.
  2. **Collection values()**: Returns a Collection view of the values contained in this map. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa. If the map is modified while an iteration over the collection is in progress (except through the iterator’s own remove operation), the results of the iteration are undefined. The collection supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Collection.remove, removeAll, retainAll and clear operations. It does not support the add or addAll operations.
  3. **Set<Map.Entry<K, V>> entrySet()**: Returns a Set view of the mappings contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator’s own remove operation, or through the setValue operation on a map entry returned by the iterator) the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Set.remove, removeAll, retainAll and clear operations. It does not support the add or addAll operations.

1. **What is difference between HashMap and Hashtable?**

HashMap and Hashtable both implements Map interface and looks similar, however there are following difference between HashMap and Hashtable.

* 1. HashMap allows null key and values whereas Hashtable doesn’t allow null key and values.
  2. Hashtable is synchronized but HashMap is not synchronized. So HashMap is better for single threaded environment, Hashtable is suitable for multi-threaded environment.
  3. LinkedHashMap was introduced in Java 1.4 as a subclass of HashMap, so incase you want iteration order, you can easily switch from HashMap to LinkedHashMap but that is not the case with Hashtable whose iteration order is unpredictable.
  4. HashMap provides Set of keys to iterate and hence it’s fail-fast but Hashtable provides Enumeration of keys that doesn’t support this feature.
  5. Hashtable is considered to be legacy class and if you are looking for modifications of Map while iterating, you should use ConcurrentHashMap.

1. **How to decide between HashMap and TreeMap?**

For inserting, deleting, and locating elements in a Map, the HashMap offers the best alternative. If, however, you need to traverse the keys in a sorted order, then TreeMap is your better alternative. Depending upon the size of your collection, it may be faster to add elements to a HashMap, then convert the map to a TreeMap for sorted key traversal.

1. **What are similarities and difference between ArrayList and Vector?**

ArrayList and Vector are similar classes in many ways.

* 1. Both are index based and backed up by an array internally.
  2. Both maintains the order of insertion and we can get the elements in the order of insertion.
  3. The iterator implementations of ArrayList and Vector both are fail-fast by design.
  4. ArrayList and Vector both allows null values and random access to element using index number.

These are the differences between ArrayList and Vector.

* 1. Vector is synchronized whereas ArrayList is not synchronized. However if you are looking for modification of list while iterating, you should use CopyOnWriteArrayList.
  2. ArrayList is faster than Vector because it doesn’t have any overhead because of synchronization.
  3. ArrayList is more versatile because we can get synchronized list or read-only list from it easily using Collections utility class.

1. **What is difference between Array and ArrayList? When will you use Array over ArrayList?**

Arrays can contain primitive or Objects whereas ArrayList can contain only Objects.  
Arrays are fixed size whereas ArrayList size is dynamic.  
Arrays doesn’t provide a lot of features like ArrayList, such as addAll, removeAll, iterator etc.

Although ArrayList is the obvious choice when we work on list, there are few times when array are good to use.

* 1. If the size of list is fixed and mostly used to store and traverse them.
  2. For list of primitive data types, although Collections use autoboxing to reduce the coding effort but still it makes them slow when working on fixed size primitive data types.
  3. If you are working on fixed multi-dimensional situation, using [][] is far more easier than List<List<>>

1. **What is difference between ArrayList and LinkedList?**

ArrayList and LinkedList both implement List interface but there are some differences between them.

* 1. ArrayList is an index based data structure backed by Array, so it provides random access to it’s elements with performance as O(1) but LinkedList stores data as list of nodes where every node is linked to it’s previous and next node. So even though there is a method to get the element using index, internally it traverse from start to reach at the index node and then return the element, so performance is O(n) that is slower than ArrayList.
  2. Insertion, addition or removal of an element is faster in LinkedList compared to ArrayList because there is no concept of resizing array or updating index when element is added in middle.
  3. LinkedList consumes more memory than ArrayList because every node in LinkedList stores reference of previous and next elements.

1. **Which collection classes provide random access of it’s elements?**

ArrayList, HashMap, TreeMap, Hashtable classes provide random access to it’s elements. Download [java collections pdf](http://cdn.journaldev.com/wp-content/uploads/2013/01/java-collections-framework.pdf) for more information.

1. **What is EnumSet?**

java.util.EnumSet is Set implementation to use with enum types. All of the elements in an enum set must come from a single enum type that is specified, explicitly or implicitly, when the set is created. EnumSet is not synchronized and null elements are not allowed. It also provides some useful methods like copyOf(Collection c), of(E first, E… rest) and complementOf(EnumSet s).

Check this post for [java enum tutorial](http://www.journaldev.com/716/java-enum).

1. **Which collection classes are thread-safe?**

Vector, Hashtable, Properties and Stack are synchronized classes, so they are thread-safe and can be used in multi-threaded environment. Java 1.5 Concurrent API included some collection classes that allows modification of collection while iteration because they work on the clone of the collection, so they are safe to use in multi-threaded environment.

1. **What are concurrent Collection Classes?**

Java 1.5 Concurrent package (java.util.concurrent) contains thread-safe collection classes that allow collections to be modified while iterating. By design Iterator implementation in java.utilpackages are fail-fast and throws ConcurrentModificationException. But Iterator implementation in java.util.concurrent packages are fail-safe and we can modify the collection while iterating. Some of these classes are CopyOnWriteArrayList, ConcurrentHashMap, CopyOnWriteArraySet.

Read these posts to learn about them in more detail.

* 1. [Avoid ConcurrentModificationException](http://www.journaldev.com/378/java-util-concurrentmodificationexception)
  2. [CopyOnWriteArrayList Example](http://www.journaldev.com/1289/copyonwritearraylist-java)
  3. [HashMap vs ConcurrentHashMap](http://www.journaldev.com/122/java-concurrenthashmap-example-iterator)

1. **What is BlockingQueue?**

java.util.concurrent.BlockingQueue is a Queue that supports operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element.

BlockingQueue interface is part of java collections framework and it’s primarily used for implementing producer consumer problem. We don’t need to worry about waiting for the space to be available for producer or object to be available for consumer in BlockingQueue as it’s handled by implementation classes of BlockingQueue.

Java provides several BlockingQueue implementations such as ArrayBlockingQueue, LinkedBlockingQueue, PriorityBlockingQueue, SynchronousQueue etc.  
Check this post for use of BlockingQueue for [producer-consumer problem](http://www.journaldev.com/1034/java-blockingqueue-example).

1. **What is Queue and Stack, list their differences?**

Both Queue and Stack are used to store data before processing them. java.util.Queue is an interface whose implementation classes are present in java concurrent package. Queue allows retrieval of element in First-In-First-Out (FIFO) order but it’s not always the case. There is also Deque interface that allows elements to be retrieved from both end of the queue.  
Stack is similar to queue except that it allows elements to be retrieved in Last-In-First-Out (LIFO) order.  
Stack is a class that extends Vector whereas Queue is an interface.

1. **What is Collections Class?**

java.util.Collections is a utility class consists exclusively of static methods that operate on or return collections. It contains polymorphic algorithms that operate on collections, “wrappers”, which return a new collection backed by a specified collection, and a few other odds and ends.

This class contains methods for collection framework algorithms, such as binary search, sorting, shuffling, reverse etc.

1. **What is Comparable and Comparator interface?**

Java provides Comparable interface which should be implemented by any custom class if we want to use Arrays or Collections sorting methods. Comparable interface has compareTo(T obj) method which is used by sorting methods. We should override this method in such a way that it returns a negative integer, zero, or a positive integer if “this” object is less than, equal to, or greater than the object passed as argument.

But, in most real life scenarios, we want sorting based on different parameters. For example, as a CEO, I would like to sort the employees based on Salary, an HR would like to sort them based on the age. This is the situation where we need to use Comparator interface because Comparable.compareTo(Object o) method implementation can sort based on one field only and we can’t chose the field on which we want to sort the Object.

Comparator interface compare(Object o1, Object o2) method need to be implemented that takes two Object argument, it should be implemented in such a way that it returns negative int if first argument is less than the second one and returns zero if they are equal and positive int if first argument is greater than second one.

Check this post for use of Comparable and Comparator interface to [sort objects](http://www.journaldev.com/780/comparable-and-comparator-in-java-example).

1. **What is difference between Comparable and Comparator interface?**

Comparable and Comparator interfaces are used to sort collection or array of objects.

Comparable interface is used to provide the natural sorting of objects and we can use it to provide sorting based on single logic.  
Comparator interface is used to provide different algorithms for sorting and we can chose the comparator we want to use to sort the given collection of objects.

1. **How can we sort a list of Objects?**

If we need to sort an array of Objects, we can use Arrays.sort(). If we need to sort a list of objects, we can use Collections.sort(). Both these classes have overloaded sort() methods for natural sorting (using Comparable) or sorting based on criteria (using Comparator).  
Collections internally uses Arrays sorting method, so both of them have same performance except that Collections take sometime to convert list to array.

1. **While passing a Collection as argument to a function, how can we make sure the function will not be able to modify it?**

We can create a read-only collection using Collections.unmodifiableCollection(Collection c)method before passing it as argument, this will make sure that any operation to change the collection will throw UnsupportedOperationException.

1. **How can we create a synchronized collection from given collection?**

We can use Collections.synchronizedCollection(Collection c) to get a synchronized (thread-safe) collection backed by the specified collection.

1. **What are common algorithms implemented in Collections Framework?**

Java Collections Framework provides algorithm implementations that are commonly used such as sorting and searching. Collections class contain these method implementations. Most of these algorithms work on List but some of them are applicable for all kinds of collections.  
Some of them are sorting, searching, shuffling, min-max values.

1. **What is Big-O notation? Give some examples?**

The Big-O notation describes the performance of an algorithm in terms of number of elements in a data structure. Since Collection classes are actually data structures, we usually tend to use Big-O notation to chose the collection implementation to use based on time, memory and performance.

Example 1: ArrayList get(index i) is a constant-time operation and doesn’t depend on the number of elements in the list. So it’s performance in Big-O notation is O(1).  
Example 2: A linear search on array or list performance is O(n) because we need to search through entire list of elements to find the element.

1. **What are best practices related to Java Collections Framework?**
   1. Chosing the right type of collection based on the need, for example if size is fixed, we might want to use Array over ArrayList. If we have to iterate over the Map in order of insertion, we need to use TreeMap. If we don’t want duplicates, we should use Set.
   2. Some collection classes allows to specify the initial capacity, so if we have an estimate of number of elements we will store, we can use it to avoid rehashing or resizing.
   3. Write program in terms of interfaces not implementations, it allows us to change the implementation easily at later point of time.
   4. Always use Generics for type-safety and avoid ClassCastException at runtime.
   5. Use immutable classes provided by JDK as key in Map to avoid implementation of hashCode() and equals() for our custom class.
   6. Use Collections utility class as much as possible for algorithms or to get read-only, synchronized or empty collections rather than writing own implementation. It will enhance code-reuse with greater stability and low maintainability.
2. **What is Java Priority Queue?**

PriorityQueue is an unbounded queue based on a priority heap and the elements are ordered in their natural order or we can provide [Comparator](http://www.journaldev.com/780/comparable-and-comparator-in-java-example) for ordering at the time of creation. PriorityQueue doesn’t allow null values and we can’t add any object that doesn’t provide natural ordering or we don’t have any comparator for them for ordering. Java PriorityQueue is not [thread-safe](http://www.journaldev.com/1061/thread-safety-in-java) and provided O(log(n)) time for enqueing and dequeing operations. Check this post for [java priority queue example](http://www.journaldev.com/1642/java-priority-queue-priorityqueue-example).

1. **Why can’t we write code as List<Number> numbers = new ArrayList<Integer>();?**

Generics doesn’t support sub-typing because it will cause issues in achieving type safety. That’s why List<T> is not considered as a subtype of List<S> where S is the super-type of T. To understanding why it’s not allowed, let’s see what could have happened if it has been supported.

List<Long> listLong = new ArrayList<Long>();

listLong.add(Long.valueOf(10));

List<Number> listNumbers = listLong; // compiler error

listNumbers.add(Double.valueOf(1.23));

As you can see from above code that IF generics would have been supporting sub-typing, we could have easily add a Double to the list of Long that would have caused ClassCastException at runtime while traversing the list of Long.

1. **Why can’t we create generic array? or write code as List<Integer>[] array = new ArrayList<Integer>[10];**

We are not allowed to create generic arrays because array carry type information of it’s elements at runtime. This information is used at runtime to throw ArrayStoreException if elements type doesn’t match to the defined type. Since generics type information gets erased at runtime by Type Erasure, the array store check would have been passed where it should have failed. Let’s understand this with a simple example code.

List<Integer>[] intList = new List<Integer>[5]; // compile error

Object[] objArray = intList;

List<Double> doubleList = new ArrayList<Double>();

doubleList.add(Double.valueOf(1.23));

objArray[0] = doubleList; // this should fail but it would pass because at runtime intList and doubleList both are just List

Arrays are covariant by nature i.e S[] is a subtype of T[] whenever S is a subtype of T but generics doesn’t support covariance or sub-typing as we saw in last question. So if we would have been allowed to create generic arrays, because of type erasure we would not get array store exception even though both types are not related.

**Difference between HashMap and HashTable / HashMap vs HashTable**    
  
**1. Synchronization or Thread Safe :**  This is the most important difference between two . HashMap is non synchronized and not thread safe.On the other hand, HashTable is thread safe and synchronized.  
When to use HashMap ?  answer is if your application do not require any multi-threading task, in other words hashmap is better for non-threading applications. HashTable should be used in multithreading applications.   
  
**2. Null keys and null values :**  Hashmap allows one null key and any number of null values, while Hashtable do not allow null keys and null values in the HashTable object.  
  
  
  
**3. Iterating the values:**  Hashmap object values are iterated by using iterator .HashTable is the only class other than vector which uses enumerator to iterate the values of HashTable object.

[](http://3.bp.blogspot.com/-BvvI4qSJ5gs/UymE9OXgBGI/AAAAAAAAASA/yXv2COAHm_U/s1600/difference+between+hashmap+and+hashtable.jpg)

**4.  Fail-fast iterator**  : The iterator in Hashmap is fail-fast iterator while the enumerator for Hashtable is not.  
According to [Oracle Docs](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html),  if the Hashtable is structurally modified at any time after the iterator is created in any way except the iterator's own remove method , then the iterator will throw ConcurrentModification Exception.  
Structural modification means adding or removing elements from the Collection object (here hashmap or hashtable) . Thus the enumerations returned by the Hashtable keys and elements methods are not fail fast.We have already explained the[difference between iterator and enumeration](http://javahungry.blogspot.com/2013/06/difference-between-iterator-and-enumeration-collections-java-interview-question-with-example.html).  
  
  
**5. Performance :**  Hashmap is much faster and uses less memory than Hashtable as former is unsynchronized . Unsynchronized objects are often much better in performance in compare to synchronized  object like Hashtable in single threaded environment.  
  
**6. Superclass and Legacy :**  Hashtable is a subclass of Dictionary class which is now obsolete in Jdk 1.7 ,so ,it is not used anymore. It is better off externally synchronizing a HashMap or using a ConcurrentMap implementation (e.g ConcurrentHashMap).HashMap is the subclass of the AbstractMap class. Although Hashtable and HashMap has different superclasses but they both are implementations of the *"Map"*  abstract data type.  
  
  
**Example of HashMap  and HashTable** 

**import** **java.util.Hashtable**;

**public** **class** **HashMapHashtableExample** {

**public** **static** **void** **main**(String[] args) {

 Hashtable<String,String> hashtableobj = **new** Hashtable<String, String>();

hashtableobj.put("Alive is ", "awesome");

hashtableobj.put("Love", "yourself");

System.out.println("Hashtable object output :"+ hashtableobj);

HashMap hashmapobj = new HashMap();

hashmapobj.put("Alive is ", "awesome");

hashmapobj.put("Love", "yourself");

System.out.println("HashMap object output :"+hashmapobj);

 }

}

**Output :***Hashtable object output :****{Love=yourself, Alive is =awesome}***  
                 *HashMap object output :****{Alive is =awesome, Love=yourself}***  
      
  
**Similarities Between HashMap and Hashtable**  
  
**1. Insertion Order :**   Both HashMap and Hashtable  does not guarantee that  the order of the map will remain constant over time. Instead use LinkedHashMap, as the order remains constant over time.  
  
**2. Map interface :**   Both HashMap and Hashtable implements Map interface .  
  
**3. Put and get method :**  Both HashMap and Hashtable provides constant time performance for put and get methods assuming that the objects are distributed uniformly across the bucket.  
  
**4. Internal working :**  Both HashMap and Hashtable works on the Principle of Hashing . We have already discussed [how hashmap works in java](http://javahungry.blogspot.com/2013/08/hashing-how-hash-map-works-in-java-or.html) .  
  
  
**When to use HashMap and Hashtable?**  
  
*1. Single Threaded Application*  
  
HashMap should be preferred over Hashtable for the non-threaded applications. In simple words , use HashMap in unsynchronized or single threaded applications .  
  
*2. Multi Threaded Application*  
  
We should avoid using Hashtable, as the class is now obsolete in latest Jdk 1.8 . Oracle has provided a better replacement of Hashtable named ConcurrentHashMap. For multithreaded  application prefer ConcurrentHashMap instead of Hashtable.  
  
  
**Recap  : Difference between HashMap and Hashtable in Java** 

|  |  |  |
| --- | --- | --- |
|  | **HashMap** | **Hashtable** |
|  |  |  |
| Synchronized | No | Yes |
|  |  |  |
| Thread-Safe | No | Yes |
|  |  |  |
| Null Keys and Null values | One null key ,Any null values | Not permit null keys and values |
|  |  |  |
| Iterator type | Fail fast iterator | Fail safe iterator |
|  |  |  |
| Performance | Fast | Slow in comparison |
|  |  |  |
| Superclass and Legacy | AbstractMap , No | Dictionary , Yes |

## **1) How to design a good key for HashMap**

The very basic need for designing a good key is that “we should be able to retrieve the value object back from the map without failure”, right?? Otherwise no matter how fancy data structure you build, it will be of no use. To decide that we have created a good key, we MUST know that “[**how HashMap works?**](http://howtodoinjava.com/core-java/collections/how-hashmap-works-in-java/)“. I will leave, how hashmap works, part on you to read from linked post, but in summary it works on principle of Hashing.

Key’s hash code is used primarily in conjunction to its equals() method, for putting a key in map and then searching it back from map. So if hash code of key object changes after we have put a key-value pair in map, then its almost impossible to fetch the value object back from map. It is a case of memory leak. To avoid this, map **keys should be immutable**. These are few things to [**create an immutable of class**](http://howtodoinjava.com/core-java/related-concepts/how-to-make-a-java-class-immutable/).

This is the main reason why immutable classes like String, Integer or other wrapper classes are a good key object candidate.

But remember that **immutability is recommended and not mandatory**. If you want to make a mutable object as key in hashmap, then you have to make sure that **state change for key object does not change the hash code of object**. This can be done by overriding the hashCode() method. Also, key class must honor the [**hashCode() and equals() methods contract**](http://howtodoinjava.com/core-java/related-concepts/working-with-hashcode-and-equals-methods-in-java/) to avoid the undesired and surprising behavior on run time. Read more about this contract in linked post.

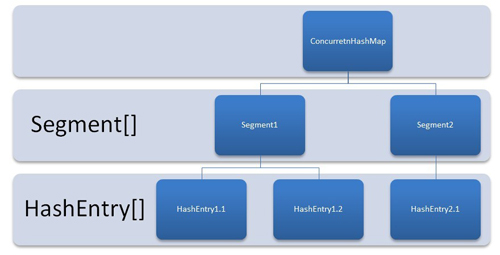
A more detailed information is available in [**here**](http://howtodoinjava.com/core-java/collections/how-to-design-a-good-key-for-hashmap/).

## **2) Difference between HashMap and ConcurrentHashMap**

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.

I will go further deeper into this concept in my coming post. I will suggest you to subscribe email updates to get notified.

## **3) Difference between HashMap and Collections.synchronizedMap(HashMap)**

It’s easy question, right !! HashMap is non-synchronized and Collections.synchronizedMap() returns a wrapped instance of HashMap which has all get, put methods synchronized.

Essentially, **Collections.synchronizedMap() returns the reference of internally created inner-class “SynchronizedMap”**, which contains key-value pairs of input HashMap, passed as argument.

This instance of inner class has nothing to do with original parameter HashMap instance and is completely independent.

## **4) Difference between ConcurrentHashMap and Collections.synchronizedMap( HashMap )**

This one is slightly tougher. Both are synchronized version of HashMap, with difference in their core functionality and internal structure.

As stated above, ConcurrentHashMap is consist of internal segments which can be viewed as independent HashMaps, conceptually. All such segments can be locked by separate threads in high concurrent executions. In this way, **multiple threads can get/put key-value pairs from ConcurrentHashMap without blocking/waiting for each other**.

In Collections.synchronizedMap(), we get a synchronized version of HashMap and **it is accessed in blocking manner**. This means if multiple threads try to access synchronizedMap at same time, they will be allowed to get/put key-value pairs one at a time in synchronized manner.

## **5) Difference between HashMap and HashTable**

It is also very easy question. The major difference is that **HashTable is synchronized and HashMap is not**.

If asked for other reasons, tell them, **HashTable is legacy class** (part of JDK 1.0) which was promoted into collections framework by implementing Map interface later. It still has some **extra features like Enumerator** with it, which HashMap lacks.

Another minor reason can be: **HashMap supports null key** (mapped to zero bucket), HashTable does not support null keys and throws NullPointerException on such attempt.

## **6) Difference between HashTable and Collections.synchronized(HashMap)**

So far you must have got the core idea of the similarities between them. Both are synchronized version of collection. Both have synchronized methods inside class. Both are blocking in nature i.e. multiple threads will need to wait for getting the lock on instance before putting/getting anything out of it.

So what is the difference. Well, **NO major difference** for above said reasons. Performance is also same for both collections.

Only thing which separates them is the fact **HashTable is legacy** class promoted into collection framework. It got its own extra features like enumerators.

## **7) Impact of random/fixed hashcode() value for key**

The impact of both cases (fixed hashcode or random hashcode for keys) will have same result and that is “**unexpected behavior**“. The very basic need of hashcode in HashMap is to identify the bucket location where to put the key-value pair, and from where it has to be retrieved.

If the hashcode of key object changes every time, the exact location of key-value pair will be calculated different, every time. This way, one object stored in HashMap will be lost forever and there will be very minimum possibility to get it back from map.

For this same reason, key are suggested to be immutable, so that they return a unique and same hashcode each time requested on same key object.

## **8) Using HashMap in non-synchronized code in multi-threaded application**

In normal cases, it **can leave the hashmap in inconsistent state** where key-value pairs added and retrieved can be different. Apart from this, other surprising behavior like NullPointerException can come into picture.

In worst case,**It can cause infinite loop**. YES. You got it right. It can cause infinite loop. What did you asked, How?? Well, here is the reason.

HashMap has the concept of rehashing when it reaches to its upper limit of size. This rehashing is the process of creating a new memory area, and copying all the already present key-value pairs in new memory are. Lets say Thread A tried to put a key-value pair in map and then rehashing started. At the same time, thread B came and started manipulating the buckets using put operation.

Here while rehashing process, there are chances to generate the cyclic dependency where one element in linked list [in any bucket] can point to any previous node in same bucket. This will result in infinite loop, because rehashing code contains a “while(true) { //get next node; }” block and in cyclic dependency it will run infinite.

To watch closely, look art source code of transfer method which is used in rehashing:

|  |
| --- |
| public Object get(Object key) {      Object k = maskNull(key);      int hash = hash(k);      int i = indexFor(hash, table.length);      Entry e = table[i];        //While true is always a bad practice and cause infinite loops        while (true) {          if (e == null)              return e;          if (e.hash == hash &amp;&amp; eq(k, e.key))              return e.value;          e = e.next;      }  } |

A **hash table** (also called a **hash**, **hash map**, **map** or **dictionary**) is a data structure that pairs keys to values.

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

lightbulbToHoursOfLight.put("incandescent", 1200);

lightbulbToHoursOfLight.put("compact fluorescent", 10000);

lightbulbToHoursOfLight.put("LED", 50000);



Hash tables:

* take *on average* **O(1)*O*(1) time for insertions and lookups**
* are **unordered** (the keys are not guaranteed to stay in the same order)
* can use **many types of objects as keys** (commonly strings)

Hash tables can be thought of as arrays, if you think of array indices as keys!

In fact, hash tables are *built on* arrays. So if you ever want to use a hash table but know your keys will be sequential integers (like 1..1001..100), you can probably save time and space by just using an array instead.

**Note:** hash tables have an **average case** insertion and lookup cost of O(1)*O*(1). In industry, we often confuse the average-case cost with *worst case* cost, but they're not really the same. Because of hash collisions and rebalancing, a hash table insertion or lookup can cost as much as O(n)*O*(*n*) time in the worst case. But usually in industry we assume hashing and resizing algorithms are clever enough that collisions are rare and cheap.