**Difference Between String , StringBuilder and StringBuffer Classes with Example : Java**

Today we are going to understand the difference between **String , StringBuilder and StringBuffer** . As you will find that there are minor differences between the above mentioned classes.  
  
**String**  
  
Stringis *immutable*  ( once created can not be changed )object  . The object created as a String is stored in the  **Constant String Pool**  .   
Every immutable object in Java is thread safe ,that implies String is also thread safe . String can not be used by two threads simultaneously.  
String  once assigned can not be changed.  
  
String  demo = " hello " ;  
// The above object is stored in constant string pool and its value can not be modified.  
  
  
demo="Bye" ;     //new "Bye" string is created in constant pool and referenced by the demo variable              
 // "hello" string still exists in string constant pool and its value is not overrided but we lost reference to the  "hello"string    
  
**StringBuffer**  
  
StringBufferis mutable means one can change the value of the object . The object created through StringBuffer is stored in the heap . StringBuffer  has the same methods as the StringBuilder , but **each method in StringBuffer is synchronized** that is **StringBuffer is thread safe** .   
  
Due to this it does not allow  two threads to simultaneously access the same method . Each method can be accessed by one thread at a time .  
  
But being thread safe has disadvantages too as the performance of the StringBuffer hits due to thread safe property . Thus  StringBuilder is faster than the StringBuffer when calling the same methods of each class.  
  
StringBuffer value can be changed , it means it can be assigned to the new value . Nowadays its a most common interview question ,the differences between the above classes .  
String Buffer can be converted to the string by using   
toString() method.  
  
StringBuffer demo1 = new StringBuffer("Hello") ;  
// The above object stored in heap and its value can be changed .

demo1=new StringBuffer("Bye");  
// Above statement is right as it modifies the value which is allowed in the StringBuffer  
  
**StringBuilder**  
  
StringBuilder  is same as the StringBuffer , that is it stores the object in heap and it can also be modified . The main difference between the StringBuffer and StringBuilder is that **StringBuilder is also not thread safe.**  
StringBuilder is fast as it is not thread safe .    
  
  
StringBuilder demo2= new StringBuilder("Hello");  
// The above object too is stored in the heap and its value can be modified  
demo2=new StringBuilder("Bye");   
// Above statement is right as it modifies the value which is allowed in the StringBuilder  
  
  
----------------------------------------------------------------------------------  
                                  ***String***                  ***StringBuffer***        ***StringBuilder***  
----------------------------------------------------------------------------------                   
**Storage Area** | Constant String Pool           Heap                       Heap   
**Modifiable**     |  No (immutable)            Yes( mutable )          Yes( mutable )  
**Thread Safe**   |           Yes                                  Yes                              No  
**Performance** |         Fast                                Very slow                    Fast  
-----------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# How to create Immutable class?

|  |
| --- |
| There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members as the example given below: |

### Example to create Immutable class

|  |
| --- |
| In this example, we have created a final class named Employee. It have one final datamember, a parameterized constructor and getter method. |

public final class Employee{

final String pancardNumber;

public Employee(String pancardNumber){

this.pancardNumber=pancardNumber;

}

public String getPancardNumber(){

return pancardNumber;

}

}

|  |
| --- |
| The above class is immutable because:   * The instance variable of the class is final i.e. we cannot change the value of it after creating an object. * The class is final so we cannot create the subclass. * There is no setter methods i.e. we have no option to change the value of the instance variable. |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Java Collections Interview Questions Answers

### 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-api-example-tutorial) 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-for-developers-lambdas-functional-interface-stream-and-time-api#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-lambda-expressions-and-functional-interfaces-example-tutorial) because it’s argument Consumer is a [function interface](http://www.journaldev.com/2763/java-8-lambda-expressions-and-functional-interfaces-example-tutorial).
  3. Miscellaneous Collection API improvements such as forEachRemaining(Consumer action) method in Iterator interface, Map replaceAll(), compute(), merge() methods.

### 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

### 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-tutorial-example-class-interface-methods-wildcards-and-much-more) to understand generics in a better way.

### What are the basic interfaces of Java Collections Framework?

[Collection](http://www.journaldev.com/1260/java-collections-framework-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/java-collections-framework-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/java-collections-framework-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/java-collections-framework-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/java-collections-framework-tutorial#queue-interface), [Dequeue](http://www.journaldev.com/1260/java-collections-framework-tutorial#dequeue-interface), [Iterator](http://www.journaldev.com/1260/java-collections-framework-tutorial#iterator-interface), [SortedSet](http://www.journaldev.com/1260/java-collections-framework-tutorial#sortedset-interface), [SortedMap](http://www.journaldev.com/1260/java-collections-framework-tutorial#sortedmap-interface) and [ListIterator](http://www.journaldev.com/1260/java-collections-framework-tutorial#listiterator-interface).

### 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.

### 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.

### 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-in-java-example-tutorial).

### 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.

### 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.

### 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.

### 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.

### 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | List&lt;String&gt; strList = new ArrayList&lt;&gt;();  //using for-each loop  for(String obj : strList){      System.out.println(obj);  }  //using iterator  Iterator&lt;String&gt; 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.

### 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.

### 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/java-arraylist-vs-copyonwritearraylist-and-exploring-iterator).

### 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/hashmap-vs-concurrenthashmap-%E2%80%93-example-and-exploring-iterator).

### 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.

### What is UnsupportedOperationException?

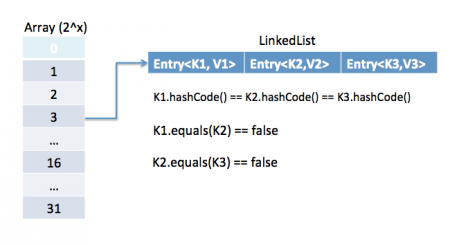
UnsupportedOperationException is the exception used to indicate that the operation is not supported. It’s used extensively in [JDK](http://www.journaldev.com/546/difference-between-jdk-jre-and-jvm-in-java) classes, in collections framework java.util.Collections.UnmodifiableCollection throws this exception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

### 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://1988780851.rsc.cdn77.org/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.

### 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.

### 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | //MyKey name argument passed is used for equals() and hashCode()  MyKey key = new MyKey("Pankaj"); //assume hashCode=1234  myHashMap.put(key, "Value");    // Below code will change the key hashCode() and equals()  // but it's location is not changed.  key.setName("Amit"); //assume new hashCode=7890    //below will return null, because HashMap will try to look for key  //in the same index as it was stored but since key is mutated,  //there will be no match and it will return null.  myHashMap.get(new MyKey("Pankaj")); |

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

### 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.

[sociallocker id=”2713″]

### 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.

### 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.

### 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.

### 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<>>

### 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.

### 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://1988780851.rsc.cdn77.org/wp-content/uploads/2013/01/java-collections-framework.pdf) for more information.

### 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-examples-with-benefits-and-class-usage).

### 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.

### 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.util packages 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/how-to-avoid-concurrentmodificationexception-when-using-an-iterator)
  2. [CopyOnWriteArrayList Example](http://www.journaldev.com/1289/java-arraylist-vs-copyonwritearraylist-and-exploring-iterator)
  3. [HashMap vs ConcurrentHashMap](http://www.journaldev.com/122/hashmap-vs-concurrenthashmap-%e2%80%93-example-and-exploring-iterator)

### 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-implementing-producer-consumer-problem).

### 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.

### 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.

### 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/java-comparable-and-comparator-example-to-sort-objects).

### 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.

### 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.

### 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.

### 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.

### 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.

### 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.

### 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.

### 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/java-comparable-and-comparator-example-to-sort-objects) 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/java-synchronization-and-thread-safety-tutorial-with-examples) 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).

### 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.

|  |  |
| --- | --- |
| 1  2  3  4 | 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.

### 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.

|  |  |
| --- | --- |
| 1  2  3  4  5 | 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.

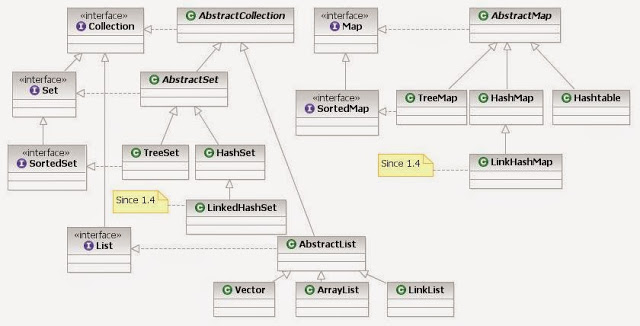
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Top 50 Java Collections Interview Questions and Answers

We have already shared the most frequently asked java interview questions for experience candidates. Also shared the tricky coding interview questions in our previous posts. Today , we will learn about the top 50 java collections interview questions and answers. We will divide this post into three categories :  
  
Beginner level (0-1 year experience (Freshers)) ,  
  
Intermediate level (1-3 years experienced Java Developers)  
  
Advanced level(3+ Experienced) java collections interview questions and answers  
  
Note : Please prepare all the below questions . Interviewer may choose to ask any question.   
 ***Beginner Level (0-1 yr): Java Collections Interview Questions  and Answers***  
  
  
**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)

source of image :  By Ervinn at en.wikibooks [CC BY-SA 3.0 ], from Wikimedia Commons  
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%. find detailed explanation   [ArrayList vs Vector](http://javahungry.blogspot.co.uk/2013/12/difference-between-arraylist-and-vector-in-java-collection-interview-question.html)  .

**Q17 What is the difference between HashMap and Hashtable ?**  
  
It is one of the most popular collections interview question for java developer . Make sure you go through this once before appearing for the interview .  
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 .  
find detailed explanation here [Hashtable vs HashMap in Java](http://javahungry.blogspot.co.uk/2014/03/hashmap-vs-hashtable-difference-with-example-java-interview-questions.html)  
  
**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 ?**  
  
This question checks whether student understand the concept of static and dynamic array. Some main differences between Array and ArrayList are :  
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.

find detailed explanation [ArrayList vs Array in Java](http://javahungry.blogspot.ca/2015/03/difference-between-array-and-arraylist-in-java-example.html)

**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.  
find detailed explanation here [TreeSet vs HashSet in Java](http://javahungry.blogspot.co.uk/2014/03/difference-between-hashset-and-treeset-similarities-and-example.html)  
  
  
**Q22 Write java code showing insertion,deletion and retrieval of HashMap object ?**  
  
Do it yourself (DIY) , if found any difficulty or doubts then please mention in the comments.  
  
**Q23 What is the difference between HashMap and ConcurrentHashMap ?**  
  
This is also one of the most popular java collections interview question . Make sure this question is in your to do list before appearing for the interview .  
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 .  
find detailed explanation here [ConcurrentHashMap vs HashMap in Java](http://javahungry.blogspot.co.uk/2014/02/hashmap-vs-concurrenthashmap-java-collections-interview-question.html)  
  
**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 . find the detailed explanation here [ArrayList vs LinkedList in java](http://javahungry.blogspot.co.uk/2015/04/difference-between-arraylist-and-linkedlist-in-java-example.html).  
  
  
  
**Q27 What are Comparable and Comparator interfaces ? List the difference between them ?**

We already explained what is comparable and comparator interface in detail along with examples here,  [Comparable vs Comparator in Java](http://javahungry.blogspot.com/2013/08/difference-between-comparable-and.html)

**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  
  
Coding part : Do it  yourself (DIY) , in case of any doubts or difficulty please mention in the comments .

***Advance Level (3+ yrs): Java Collections Interview Questions  and Answers***

**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. We have already discussed in detail [hashset internal working in java](http://javahungry.blogspot.co.uk/2013/08/how-sets-are-implemented-internally-in.html).  
  
**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 .

**Q33  How HashMap works in Java ?**  
  
We are repeating this question , as it is one of the most important question for java developer.HashMap works on the principle of Hashing . please find the detailed answer here [hashmap internal working in java](http://javahungry.blogspot.co.uk/2013/08/hashing-how-hash-map-works-in-java-or.html) .  
  
**Q34 How remove(key) method works in HashMap ?**  
  
This is the new question which is getting popular among java interviewers . We have shared the detailed explanation here [how remove method works internally in java](http://javahungry.blogspot.co.uk/2015/03/how-remove-method-internally-works-in-hashmap-java.html).  
  
**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. Please find the detailed answers here [internal implementation of TreeMap in java](http://javahungry.blogspot.co.uk/2014/06/how-treemap-works-ten-treemap-java-interview-questions.html) .  
  
**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)  
  
**Q38 What is the difference between Fail- fast iterator and Fail-safe iterator ?**  
This is one  of the most popular interview question for the higher experienced java developers .  
Main differences between Fail-fast and Fail-safe iterators are :  
a. Fail-fast throw ConcurrentModificationException while Fail-safe does not.  
b. Fail-fast does not clone the original collection list of objects while Fail-safe creates a copy of the original collection list of objects.  
The difference is explained in detail here [fail-safe vs fail-fast iterator in java](http://javahungry.blogspot.co.uk/2014/04/fail-fast-iterator-vs-fail-safe-iterator-difference-with-example-in-java.html).  
  
  
**Q39 How ConcurrentHashMap works internally in Java?**  
  
The detailed answer is already explained here  [internal implementation of concurrenthashmap](http://javahungry.blogspot.co.uk/2015/02/how-concurrenthashmap-works-in-java-internal-implementation.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)  
 **Q43 What is EnumSet in Java ?**  
[EnumSet](http://docs.oracle.com/javase/7/docs/api/java/util/EnumSet.html)  is a specialized Set implementation for 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.  
The iterator never throws ConcurrentModificationException and is weakly consistent.  
Advantage over HashSet:  
All basic operations of EnumSet execute in constant time . It is most likely to be much faster than HashSet counterparts.  
It is a part of Java Collections Framework since jdk 1.5.  
  
**Q44 What are concurrentCollectionClasses?**   
In jdk1.5 , Java Api developers had introduced new package called java.util.concurrent that have thread-safe collection classes as they allow collections to be modified while iterating . The iterator is fail-fast that is it will throw ConcurrentModificationException.  
Some examples of concurrentCollectionClasses are :  
a. CopyOnWriteArrayList  
b. ConcurrentHashMap  
  
**Q45 How do you convert a given Collection to SynchronizedCollection ?**  
One line code :    Collections.synchronizedCollection(Collection collectionObj) will convert a given collection to synchronized collection.  
  
**Q46  What is IdentityHashMap ?**  
 **IdentityHashMap**  
[IdentityHashMap](http://docs.oracle.com/javase/7/docs/api/java/util/IdentityHashMap.html) is a class present in java.util package. It implements the Map interface with a hash table , using [reference equality instead of object equality](http://javahungry.blogspot.co.uk/2013/06/difference-between-equals-and-double-equals-method-with-example-java-collections-interview-question.html) when comparing keys and values.In other words , in IdentityHashMap two keys k1 and k2 are considered equal if only if (k1==k2).  
IdentityHashMap is not synchronized.  
Iterators returned by the iterator() method are fail-fast , hence , will throw ConcurrentModificationException.   
  
**Q47 What is  WeakHashMap ?**  
 **WeakHashMap :**  
[WeakHashMap](http://docs.oracle.com/javase/7/docs/api/java/util/WeakHashMap.html) is a class present in java.util package similar to IdentityHashMap. It is a Hashtable based implementation of Map interface with weak keys. An entry in WeakHashMap will automatically be removed when its key is no longer in ordinary use. More precisely the presence of a mapping for a given key will not prevent the key from being discarded by the garbage collector.  
It permits null keys and null values.  
Like most collection classes this class is not synchronized.A synchronized WeakHashMap may be constructed using the Collections.synchronizedMap() method.  
Iterators returned by the iterator() method are fail-fast , hence , will throw ConcurrentModificationException.

**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)

**Q49  What is UnsupportedOperationException?**  
This exception is thrown to indicate that the requested operation is not supported.  
Example of UnsupportedOperationException:  
In other words, if you call add() or remove() method on the readOnly collection . We know readOnly collection can not be modified . Hence , UnsupportedOperationException will be thrown.  
 **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)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*