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

[Java Stream API](https://www.journaldev.com/2774/java-8-stream) for collection classes for supporting sequential as well as parallel processing

[Iterable interface is extended with forEach()](https://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](https://www.journaldev.com/2763/java-8-functional-interfaces) because it’s argument Consumer is a [function interface](https://www.journaldev.com/2763/java-8-functional-interfaces).

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?**

initial java release contained few classes for collections: **Vector**, **Stack**, **Hashtable**, **Array**. 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**](https://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](https://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](https://www.journaldev.com/1260/collections-in-java-tutorial#set-interface)  => cannot contain duplicate elements. This interface models the mathematical set abstraction and is used to represent sets, such as the deck of cards.

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

[Map](https://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](https://www.journaldev.com/1260/collections-in-java-tutorial#queue-interface), [Dequeue](https://www.journaldev.com/1260/collections-in-java-tutorial#dequeue-interface), [Iterator](https://www.journaldev.com/1260/collections-in-java-tutorial#iterator-interface), [SortedSet](https://www.journaldev.com/1260/collections-in-java-tutorial#sortedset-interface), [SortedMap](https://www.journaldev.com/1260/collections-in-java-tutorial#sortedmap-interface) and [ListIterator](https://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.  
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](https://www.journaldev.com/1260/collections-in-java-tutorial). **Iterators allow the caller to remove elements from the underlying collection during the iteration.**

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.**, 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 ,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?**

|  |  |
| --- | --- |
| 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.

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.

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 choose 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](https://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 remove operations.

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.

HashMap initial default capacity is **16** and load factor is 0.75. Threshold is **capacity \* 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.

If o1.equals(o2), then o1.hashCode() == o2.hashCode()should always be true.

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.
  14. 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:

**Set<K> 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.

**Collection<V> 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.

**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** | **HashTable** |
| Allows null key | Doesn’t allow null key , legacy class |
| Not synchronized | Synchronized |
| 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 | Hash table iteration order is unpredictable |
| Hash Map provides Set of keys to iterate and hence it’s fail-fast | Hash table provides Enumeration of keys that doesn’t support this feature. |

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?**
   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](https://www.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](https://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](https://www.journaldev.com/378/java-util-concurrentmodificationexception)
  2. [CopyOnWriteArrayList Example](https://www.journaldev.com/1289/copyonwritearraylist-java)
  3. [HashMap vs ConcurrentHashMap](https://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](https://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](https://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](https://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](https://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](https://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 compile time 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.

**2. What is the difference between poll() and remove() method of Queue interface? (answer)**  
Though both poll() and remove() method from Queue is used to remove the object and returns the head of the queue, there is a subtle difference between them. If Queue is empty() then a call to remove() method will throw Exception, while a call to poll() method returns null. By the way, exactly which element is removed from the queue depends upon queue's ordering policy and varies between different implementation, for example, PriorityQueue keeps the lowest element as per Comparator or Comparable at head position.   
  
**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'sremove() method during iteration then your code will throwConcurrentModificationException. That's why it's advised to use Iterator remove() method to remove objects from Collection.

**5. What is the difference between Synchronized Collection and Concurrent Collection? (**[**answer**](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html)**)**

e.g.ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue etc, which has made Interview questions on Java Collection even trickier. Java Also provided a way to get Synchronized copy of collection e.g. ArrayList, HashMap by usingCollections.synchronizedMap() Utility function.One Significant difference is that Concurrent Collections has better performance than synchronized Collection because they lock only a portion of Map to achieve concurrency and Synchronization. See the difference between Synchronized Collection and Concurrent Collection in Java for more details.

**7. How does HashSet is implemented in Java, How does it use Hashing? (**[**answer**](http://java67.blogspot.com/2014/01/how-hashset-is-implemented-or-works-internally-java.html)**)**  
This is a tricky question in Java because for hashing you need both key and value and there is no key for the store it in a bucket, then how exactly HashSetstore element internally. Well, HashSet is built on top of HashMap. If you look at source code of java.util.HashSet class, you will find that that it uses a HashMap with same values for all keys, as shown below:

private transient HashMap map;  
// Dummy value to associate with an Object in the backing Map  
private static final Object PRESENT = new Object();  
When you call add() method of HashSet, it put entry in HashMap :  
public boolean add(E e) {  
  return map.put(e, PRESENT)==null;  
}  
Since keys are unique in a HashMap, it provides uniqueness guarantee of Set interface.

**8. What do you need to do to use a custom object as a key in Collection classes like Map or Set? (**[**answer**](http://javarevisited.blogspot.com/2015/01/why-override-equals-hashcode-or-tostring-java.html)**)**  
The answer is: If you are using any custom object in Map as key, you need to override equals() and hashCode() method, and make sure they follow their contract. On the other hand if you are storing a custom object in Sorted Collection e.g. SortedSet or SortedMap, you also need to make sure that your equals() method is consistent to compareTo() method, otherwise that collection will not follow there contacts e.g. Set may allow duplicates.  
  
**10. When do you use ConcurrentHashMap in Java**

This is another advanced level collection interview questions in Java which normally asked to check whether the interviewer is familiar with optimization done onConcurrentHashMap or not. ConcurrentHashMap is better suited for situation where you have multiple readers and one

Writer or fewer writers since Map gets locked only during the write operation. If you have an equal number of reader and writer than [ConcurrentHashMap](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) will perform in the line of Hashtable or synchronized HashMap.

**12. How do you Sort objects on the collection?**

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.

**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 vsHashtable. 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?**   
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: 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:

**19) Why ListIterator has added() method but Iterator doesn't or Why to add() method is declared in ListIterator and not on Iterator.**

Answer: ListIterator has added() method because of its ability to traverse or iterate in both direction of the collection. it maintains two pointers in terms of previous and next call and in a position to add a new element without affecting current iteration.

**20) When does ConcurrentModificationException occur on iteration**

When you remove object using Collection's or List's remove method e.g.remove(Object element) or remove(int index), instead of Iterator's remove() method than ConcurrentModificationException occurs. As per Iterator's contract, if it detect any structural change in Collection e.g. adding or removing of the element, once Iterator begins, it can throw ConcurrentModificationException.  Here are some tips to avoid ConcurrentModification in Java.

**21) Difference between Set, List and Map Collection classes?**

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

**22) What is BlockingQueue, how it is different than other collection classes?**

BlockingQueue is a Queue implementation available in java.util.concurrentpackage. 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.

**What is the difference between Hashtable and ConcurrentHashMap in Java?**

Another frequently asked Java collection interview question post-Java 5 world which introduced Concurrent Collection classes like ConcurrentHashMap and CopyOnWriteArrayList along with Concurrency utilities e.g. [CyclicBarrier](http://javarevisited.blogspot.sg/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html) and [CountDownLatch](http://javarevisited.blogspot.sg/2012/07/countdownlatch-example-in-java.html). Well, both Hashtable and ConcurrentHashMap are thread-safe here but later provides more scalability than former.

**What is the difference between ArrayList and LinkedList in Java?**

A follow-up question which is asked in response to previous Java collection interview question. Here also both LinkedList and ArrayList are List implementation but their internal data-structure is different, one is derived from Array while other is derived from LinkedList. See [LinkedList vs ArrayList in Java](http://javarevisited.blogspot.sg/2012/02/difference-between-linkedlist-vs.html) to answer this Java Collection interview question.



**Q1  What is Collection ? What is a Collections Framework ? What are the benefits of Java Collections Framework ?**  
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" \t "_blank) , 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 ?**

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.  
  
**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);**  
  
**Q16 What is the difference between ArrayList and Vector ?**

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" \t "_blank)  .

**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 .  
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" \t "_blank)  
  
**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.

**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.  
  
**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 .  
  
**Q24 Arrange the following in the ascending order (performance):**  
**HashMap , Hashtable , ConcurrentHashMap and Collections.SynchronizedMap**  
Hashtable  <  Collections.SynchronizedMap  <  ConcurrentHashMap  <  HashMap

**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" \t "_blank).  
  
**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.  
  
**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 .

**Q34 How remove(key) method works in HashMap ?**  
  
  
**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.

**Q39 How ConcurrentHashMap works internally in Java?**

**Q41 What is hash-collision in Hashtable ? How it was handled in Java?**  
In Hashtble , 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.  
 **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-safe that is it will not 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** 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](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 ?**

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)  

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

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

#### **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().

**6. What is difference between Iterator ,ListIterator and Enumeration?**

* Enumeration interface implemented in java 1.2 version.So Enumeration is legacy interface.
* Enumeration uses elements() method.
* Iterator is implemented on all Java collection classes.
* Iterator uses iterator() method.
* Iterator can traverse in forward direction only.
* ListIterator is implemented only for List type classes
* ListIterator uses listIterator() method.

Read more :

#### **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

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

* ArrayList
* LinkedList
* Vector

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

* HashSet
* LinkedHashSet
* TreeSet

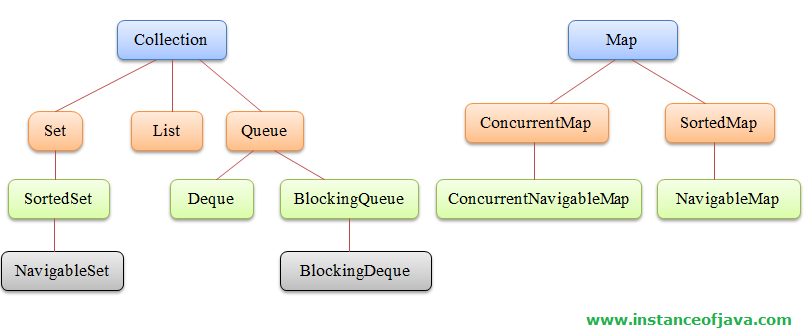
#### **11.How to make a collection thread safe?**

* 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?**



#### **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.

#### **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

1. What is Java Collections Framework? What are the advantages of the Java Collections Framework?

A **collection**, sometimes called a container, is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data. The Java collections framework (JCF) is a set of classes and interfaces that implement commonly reusable collection data structures.

The Java Collections Framework provides the following benefits:

* Reduces programming effort by providing useful data structures and algorithms.
* Increases program speed and quality by providing high-performance, high-quality implementations of useful data structures and algorithms.
* Allows inter-operability among unrelated APIs
* Reduces effort to design, learn and to use new APIs
* Encourages software reuse

3. What is the difference between ArrayList and Vector?

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| ArrayList is not synchronized. Hence ArrayList is not thread-safe. | Vector is thread-safe. In Vector class each method like add(), get(int i) is surrounded with a synchronized block and thus making Vector class thread-safe. |
| 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. ArrayList increases its size by 50% of the array size. | Vector increases its size by doubling the array size. |

6. What is the difference between ArrayList and LinkedList?

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| Resizable-array implementation of the List interface. ArrayList uses a dynamic array. | Doubly-linked list implementation of the List and Deque interfaces. |
| Frequent insertion/removal of elements can be slower based on the index. This is because the elements have to be shifted in the array. | Frequent insertion/removal of elements is faster. This is because removal only requires change in the pointer location in the two neighbor nodes (elements) of the node which is going to be removed. |
| Search operation is faster in ArrayList because it maintains index based system for its elements. | Search operation is slower because it requires traversing through all the elements for searching an element. |
| Memory consumption is comparatively low | Memory consumption is high because of the additional pointers required for each element to maintain the structure. |
| Supports only List operations | Supports both list and queue operations like poll(), peek() etc. |

Additionally they may ask which is preferred – Arraylist or Linkedlist? You can say that it depends on the requirement. If the requirement involves frequent insertion or removal then LinkedList is preferred. On the other hand if the requirement involves frequent accessing of the elements then ArrayList is preferred.

7. What is the difference between Iterator and Enumeration?

|  |  |
| --- | --- |
| **Iterator** | **Enumeration** |
| Iterators allow the caller to remove elements from the underlying collection. | With Enumeration, you cannot remove elements. |
| Iterator is fail-fast. 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. | Enumeration is not fail-fast. |
| Iterator is slower than Enumeration. | Enumeration is twice as fast as Iterator and uses very less memory. |

Also, Iterator method names have been improved to make it’s functionality clear. Iterator takes the place of Enumeration in the Java Collections Framework.

8. What is the difference between Iterator and ListIterator?

|  |  |
| --- | --- |
| **Iterator** | **ListIterator** |
| Iterator traverses the elements in forward direction only. | ListIterator traverses the elements in both directions(forward & backward). |
| Iterator can be used with List, Set and Queue. | ListIterator can be used with List only. |

Additionally, ListIterator comes with extra functionalities like adding an element, replacing an element, getting index position for previous and next elements.

9. What is the difference between HashMap and HashTable?

|  |  |
| --- | --- |
| **HashMap** | **HastTable** |
| HashMap is not thread-safe. Since HashMap is not synchronized, it performs better(faster) than Hashtable. | Mehods in HashTable are synchronized. So it is thread-safe |
| Allows one null key and any number of null values. | Does not permit null key and null values. |
| HashMap retrieval is not in order i.e. HashMap does not guarantee that the order of the map will remain constant over time. But LinkedHashMap provides ordered retrieval. | In case of HashTable, iteration order is unpredictable |
| Iterator in the HashMap is a fail-fast iterator | the enumerator for the Hashtable is not fail-fast. |

10. What do you mean by fail-fast and fail-safe property?

Iterator fail-fast property checks for any modification in the structure of the underlying collection every time we try to get the next element. If the collection is structurally modified at any time after the iterator is created, in any way except through the iterator’s own remove or add methods, the iterator will throw a **ConcurrentModificationException.** All the implementations of Iterator in Collection classes are fail-fast by design except the concurrent collection classes like ConcurrentHashMap and CopyOnWriteArrayList.

In the context of concurrent modification, fail-safe iterator does not throw ConcurrentModificationException

14. What are the differences between Comparable and Comparator interface?

Comparable and Comparator interfaces are used to sort collection or array of objects. Comparable interface is for self-comparing i.e. the class which implements comparable interface compares one of its objects with another. On the other hand, Comparator is more flexible. The class which implements Comparator can compare two objects of another class.

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| This interface imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class’s natural ordering, and the class’s compareTo method is referred to as its natural comparison method. | A comparison function, which imposes a total ordering on some collection of objects. Comparators can be passed to a sort method (such as Collections.sort or Arrays.sort) to allow precise control over the sort order. Comparators can also be used to control the order of certain data structures (such as sorted sets or sorted maps), or to provide an ordering for collections of objects that don’t have a natural ordering. |
| Virtually all Java core classes that implement Comparable have natural orderings that are consistent with equals. One exception is java.math.BigDecimal, whose natural ordering equates BigDecimal objects with equal values and different precisions (such as 4.0 and 4.00). | It is generally a good idea for comparators to also implement **java.io.Serializable**, as they may be used as ordering methods in serializable data structures (like TreeSet, TreeMap). In order for the data structure to serialize successfully, the comparator (if provided) must implement Serializable. |
| comparison of null arguments not allowed | Unlike Comparable, a comparator may optionally permit comparison of null arguments, while maintaining the requirements for an equivalence relation. |
| Present in **java.lang** package | Present in**java.util** package which very much says that Comparator should be used as an utility to sort objects which Comparable should be provided by default. |
| **compareTo(T ob)**  Compares this object with the specified object for order. | **compare(T o1, T o2)**  Compares its two arguments for order.  **equals(Object obj)**  Indicates whether some other object is “equal to” this comparator |

15. How can we sort a list of Objects?

If we need to sort a list of objects, we can use Collections.sort(). It provides overloaded sort() methods for natural sorting (using Comparable) or sorting based on criteria (using Comparator). Collections internally uses Arrays sorting method.

16. How can we create a synchronized collection from given collection?

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

17. What is the difference between HashSet and TreeSet

The HashSet is Implemented using a hash table and thus, its elements are not ordered. The add, remove, and contains methods of a HashSet have constant time complexity O(1). On the other hand, a [TreeSet](https://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html" \t "_blank) is implemented using a tree structure. The elements in a TreeSet are sorted, and thus, the add, remove, and contains methods have time complexity of O(logn).

18. What are the difference between Set and List?

|  |  |
| --- | --- |
| **Set** | **List** |
| Set is an unordered collection, it doesn’t maintain any order. There are few implementations of Set which maintains the order such as LinkedHashSet (It maintains the elements in insertion order). | List is an ordered collection i.e. it maintains the order in which elements are added |
| Set doesn’t allow duplicate elements. | List allows duplicates |
| Set can have only one null value at most. | List allows any number of null values. |
| ListIterator cannot be used to traverse a Set | ListIterator can be used to traverse a List in both the directions(forward and backward). |
| The List interface provides two methods to search for a specified object index i.e. indexOf() and lastIndexOf(). | No methods to search for index |
| The List interface provides two methods to efficiently insert and remove multiple elements at an arbitrary point in the list. Can add elements at a specified index. | Not possible with Set. |

19. What data structure to use for ordered map 🡺 [LinkedHashMap](https://docs.oracle.com/javase/7/docs/api/java/util/LinkedHashMap.html).

20. What data structure to use for sorted map 🡺 [TreeMap](https://docs.oracle.com/javase/7/docs/api/java/util/TreeMap.html).

21. What data structure to use for sorted set 🡺 [TreeSet](http://www.javatechblog.com/java/treeset-with-user-defined-objects-in-java/).

22. How to convert an array to list?

We can use the Arrays.asList() method to create list of objects from an array of objects.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import java.util.Arrays;  import java.util.List;    //String array  String[] array = {"cow", "lion", "horse", "dog", "cat"};  //Use utility class - Arrays  ArrayList<String> list = Arrays.asList(array); |

23. How to convert an ArrayList to array?

We can use the toArray() method defined in the List interface.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | List<String> list = new ArrayList<String>();  list.add("a");  list.add("b");  list.add("c");  String [] array = new String[list.size()];  list.toArray(array); |

24. How to reverse the list?

This question is to test your knowledge of [Collections](https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html) utility class. Use it’s reverse() method to reverse the list.

|  |  |
| --- | --- |
| 1 | Collections.reverse(list); |

25. How to make a collection thread safe?

Using the below methods:

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

26. How to iterate a HashMap?

It depends on what you want from the map. If you want to iterate over both keys and values, then you can use HashMap’s entrySet() method which will return a Set. You can then iterate the set using iterator. If you just want only keys or values, you can use keySet() or values() method of HashMap.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | import java.util.Collection;  import java.util.HashMap;  import java.util.Map;  import java.util.Map.Entry;  import java.util.Set;    Map<String, String> map = new HashMap<String, String>();  map.put("1", "one");  map.put("2", "two");  map.put("3", "three");  map.put("4", "four");  map.put("5", "five");    Set<Entry<String, String>> entrySet = map.entrySet();  for (Entry<String, String> entry : entrySet) {  System.out.println(entry.getKey());  System.out.println(entry.getValue());  }    Set<String> keys = map.keySet();  Collection<String> values = map.values(); |

27. What do you need to do to use a custom object as a key in Collection classes like Map or Set?

If you are using any custom object in Map as key, you need to override [equals() and hashCode()](http://www.javatechblog.com/java/equals-hashcode-java/)method, and make sure they follow their contract. On the other hand if you are storing a custom object in Sorted Collection e.g. SortedSet or SortedMap, you also need to make sure that your equals() method is consistent with compareTo() method. Not following this will break the underlying collection’s principle itself e.g. Set may allow duplicate values.

28. What are the different ways to iterate a list in Java?

There are two ways to iterate over the list in Java

* using Iterator

using for-each loop

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | List<String> list = new ArrayList<String>();  list.add("a");  list.add("b");  list.add("c");    // Using iterator  Iterator<String> itr = list.iterator();  while(itr.hasNext()){  System.out.println(itr.next());  }    // Using enhanced for loop  for(String str: list){  System.out.println(str);  } |