**1. What is the difference between Iterator and Enumeration? (answer)**

This is a beginner level collection interview questions and mostly asked during interviews of Junior Java developer up to experience of 2 to 3 years Iterator duplicate functionality of Enumeration with one addition of remove() method and both provide navigation functionally on objects of Collection.Another difference is that Iterator is more safe than Enumeration and doesn't allow another thread to modify collection object during iteration except remove() method and throws ConcurrentModificaitonException. See Iterator vs Enumeration in Java for more differences.

**2. How do you Sort objects on the collection? (solution)**

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

**3. 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 the 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;

Reduced development effort by using core collection classes rather than implementing our own collection classes.

Code quality is enhanced with the use of well tested collections framework classes.

Reduced effort for code maintenance by using collection classes shipped with JDK.

Reusability and Interoperability

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

Collection 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 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 is an ordered collection and can contain duplicate elements. You can access any element from its index. The list is more like an array with dynamic length.

A Map is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.

**5. What is an Iterator?**

The 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.

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

Enumeration is twice as fast as Iterator and uses very little memory. Enumeration is very basic and fits basic needs. But the 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 its functionality clear.

**7. What is different between Iterator and ListIterator?**

We can use Iterator to traverse Set and List collections whereas ListIterator can be used with Lists only.

Iterator can traverse in forward direction only whereas ListIterator can be used to traverse in both the directions.

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.

**8. What are different ways to iterate over a list?**

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

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

//using for-each loop

for(String obj : strList){

System.out.println(obj);

}

//using iterator

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

while(it.hasNext()){

String obj = it.next();

System.out.println(obj);

}

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

**9. 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 don’t provide a lot of features like ArrayList, such as addAll, removeAll, iterator, etc.

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

If the size of list is fixed and mostly used to store and traverse them.

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.

If you are working on fixed multi-dimensional situation, using [][] is far more easier than List<List<>>

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

Java provides a Comparable interface which should be implemented by any custom class if we want to use Arrays or Collections sorting methods. The comparable interface has a 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 an 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 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 choose 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 the first argument is less than the second one and returns zero if they are equal and positive int if the first argument is greater than the second one.

Check this post for use of Comparable and Comparator interface to sort objects.

**11. 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.

**12. 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 choose the comparator we want to use to sort the given collection of objects.

**13. 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.

**14. 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.

**15. What is framework in Java?**

A framework is a popular and readymade architecture that contains a set of classes and interfaces.

**16. What is the Collection framework in Java?**

Collection Framework is a grouping of classes and interfaces that is used to store and manage the objects. It provides various classes like Vector, ArrayList, HashSet, Stack, etc. Java Collection framework can also be used for interfaces like Queue, Set, List, etc.

**17. Distinguish between ArrayList and Vector in the Java collection framework.**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| ArrayList is cannot be synchronized. | Vector can be is synchronized. |
| It is not a legacy class. | It is a legacy class. |
| It can increase its size by 50% of the size of the array. | It can increase its size by doubling the size of the array. |
| ArrayList is not thread-safe. | Vector is a thread-safe. |

**18. What is ArrayList in Java?**

ArrayList is a data structure that can be stretched to accommodate additional elements within itself and shrink back to a smaller size when elements are removed. It is a very important data structure useful in handling the dynamic behavior of elements.

**19. Explain the basic interfaces of the Java collections framework**

Java collection framework is a root of the collection hierarchy. It represents a group of objects as its elements. The Java programming language does not provide a direct implementation of such interface.

Set: Set is a collection having no duplicate elements. It uses hashtable for storing elements.

List: List is an ordered collection that can contain duplicate elements. It enables developers to access any elements from its inbox. The list is like an array having a dynamic length.

MAP: It is an object which maps keys to values. It cannot contain duplicate keys. Each key can be mapped to at least one value.

**1. How does HashMap work in Java? (**[**answer**](http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html)**)**

This is *Classical Java Collection interview questions* which I have also discussed in my earlier article [how does HashMap works in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html). This collection interview questions is mostly asked during AVP Role interviews on Investment-Banks and has a lot of follow-up questions based on the response of interviewee e.g. Why HashMap keys need to be [immutable](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html), what is race conditions on HashMap and how HashMap resize in Java. For explanation and answers of these questions Please see earlier link.

**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. 

**3. What is the difference between fail-fast and fail-safe Iterators? (**[**answer**](http://java67.blogspot.com/2015/06/what-is-fail-safe-and-fail-fast-iterator-in-java.html)**)**

This is relatively *new collection interview questions* and can become trick if you hear the term fail-fast and fail-safe first time. Fail-fast Iterators throws ConcurrentModificationException when one [Thread](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) is iterating over collection object and other thread structurally modify Collection either by adding, removing or modifying objects on underlying collection. They are called fail-fast because they try to immediately throw Exception when they encounter failure. On the other hand [fail-safe Iterators](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html) works on copy of collection instead of original collection

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

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

Java 5 has added several new Concurrent Collection classes 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 using Collections.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.

**6. What is the difference between Iterator and Enumeration? (**[**answer**](http://javarevisited.blogspot.com/2010/10/what-is-difference-between-enumeration.html)**)**

This is a beginner level collection interview questions and mostly asked during interviews of Junior Java developer up to experience of 2 to 3 years Iterator duplicate functionality of Enumeration with one addition of remove() method and both provide navigation functionally on objects of Collection.Another difference is that Iterator is more safe than Enumeration and doesn't allow another thread to modify collection object during iteration except remove() method and throws ConcurrentModificaitonException. See Iterator vs Enumeration in Java for more differences.

**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 HashSet store 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.

**9. The difference between HashMap and Hashtable? (**[**answer**](http://javarevisited.blogspot.com/2015/08/difference-between-HashMap-vs-TreeMap-vs-LinkedHashMap-Java.html)**)**

This is another Classical Java Collection interview asked on beginner’s level and most of Java developer has a predefined answer for this interview questions e.g. HashMap is not synchronized while Hashtable is not or hashmap is faster than hash table etc. What could go wrong is that if he placed another follow-up question like how hashMap works in Java or can you replace Hashtable with ConcurrentHashMap etc. See [Hashtable vs HashMap in Java](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html) for detailed answer of this interview question.

**10. When do you use ConcurrentHashMap in Java? (**[**answer**](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html)**)**

This is another advanced level collection interview questions in Java which normally asked to check whether the interviewer is familiar with optimization done on ConcurrentHashMap 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.

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

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

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

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

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

One more beginner level collection interview questions, this is still very popular and mostly asked in the telephonic round. [ArrayList in Java](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html) is one of the most used Collection class and the most interviewers asked questions on ArrayList. See Difference between Vector and ArrayList for the answer to this interview question.

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

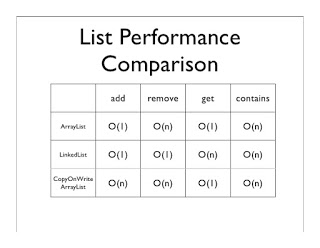
This collection interview questions is asked in conjunction with HashMap vs Hashtable. HashSet implements java.util.Set interface and that's why only contains unique elements, while HashMap allows duplicate values.  In fact, HashSet is actually implemented on top of java.util.HashMap. If you look internal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

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

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

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

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

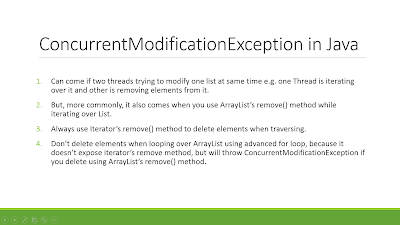
[](https://3.bp.blogspot.com/-C3omsD-5Dmk/VmQRsg6xYjI/AAAAAAAAEQk/Q2h35mRULLc/s1600/ArrayList+LinkedList+and+CopyOnWriteArrayList.jpg)

**19) Why ListIterator has added() method but Iterator doesn't or Why to add() method is declared in ListIterator and not on Iterator. (**[**answer**](http://java67.blogspot.com/2013/02/java-iterator-example-and-tutorial.html)**)**

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.

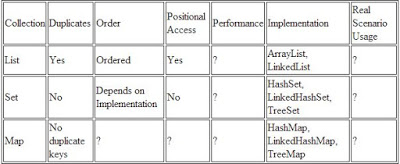
**20) When does ConcurrentModificationException occur on iteration? (**[**answer**](http://java67.blogspot.com/2015/10/how-to-solve-concurrentmodificationexception-in-java-arraylist.html)**)**

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.

[](https://4.bp.blogspot.com/-XAU2_20M9Yg/VmQRaG8-IBI/AAAAAAAAEQc/G9dmlrMXdU0/s1600/ConcurrentModificationException+while+Iterating+over+ArrayList.png)

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

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

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

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

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

Read more: <https://javarevisited.blogspot.com/2011/11/collection-interview-questions-answers.html#ixzz6u0luvwvs>

**1) What is framework in Java?**

A framework is a popular and readymade architecture that contains a set of classes and interfaces.

**2) What is the Collection framework in Java?**

Collection Framework is a grouping of classes and interfaces that is used to store and manage the objects. It provides various classes like Vector, ArrayList, HashSet, Stack, etc. Java Collection framework can also be used for interfaces like Queue, Set, List, etc.

**3) Explain Collections Class**

java.util.Collections is a class consists of static methods that operate on collections. It contains polymorphic algorithms to operate on collections, "wrappers". This class contains methods for algorithms, like binary sorting, search, shuffling, etc.

**4) What is the hashCode()?**

The hashCode() is a method that returns an integer hash code.

[](https://www.guru99.com/images/2/062520_0728_Top80JavaCo1.png)

**5) Distinguish between ArrayList and Vector in the Java collection framework.**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| ArrayList is cannot be synchronized. | Vector can be is synchronized. |
| It is not a legacy class. | It is a legacy class. |
| It can increase its size by 50% of the size of the array. | It can increase its size by doubling the size of the array. |
| ArrayList is not thread-safe. | Vector is a thread-safe. |

**6) What is ArrayList in Java?**

ArrayList is a data structure that can be stretched to accommodate additional elements within itself and shrink back to a smaller size when elements are removed. It is a very important data structure useful in handling the dynamic behavior of elements.

**7) Differentiate between Iterator and ListIterator**

The difference between Iterator and ListIterator is:

|  |  |
| --- | --- |
| **Iterator** | **ListIterator** |
| The Iterator can traverse the array elements in the forward direction. | ListIterator can traverse the array elements in backward as well as forward directions. |
| It can be used in Queue, List, and Set. | It can be used in List. |
| It can perform only remove operation. | It can perform add, remove, and set operation while traversing the collection. |

**8) What is the difference between Iterator and Enumeration?**

The difference between Iterator and Enumeration

|  |  |
| --- | --- |
| **Iterator** | **Enumeration** |
| The Iterator can traverse both legacies as well as non-legacy elements. | Enumeration can traverse only legacy elements. |
| The Iterator is fail-fast. | Enumeration is not fail-fast. |
| The Iterator is very slow compare to Enumeration. | Enumeration is fast compare to Iterator. |
| The Iterator can perform remove operation while traversing the collection. | The Enumeration can perform only traverse operation on the collection. |

**9) Define BlockingQueue**

BlockingQueue is an interface used in Java that can extend the Queue. It provides concurrency in various queue operations like retrieval, insertion, deletion, etc.

The Queue waits to become non-empty at the time of retrieving any elements. BlockingQueue should not contain null elements. The implementation of this Queue is thread-safe.

The syntax of BlockingQueue is:

public interface BlockingQueue<E> extends Queue <E>

**10) Explain override equals() method**

The equals method is used to check the similarity between two objects. In case if the programmer wants to check an object based on the property, then it needs to be overridden.

**11) What is the difference between Comparable and Comparator?**

The difference between Comparable and Comparator is:

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| Comparable provides compareTo() method to sort elements in Java. | Comparator provides compare() method to sort elements in Java. |
| Comparable interface is present in java.lang package. | Comparator interface is present in java. util package. |
| The logic of sorting must be in the same class whose object you are going to sort. | The logic of sorting should be in a separate class to write different sorting based on different attributes of objects. |
| The class whose objects you want to sort must implement the comparable interface. | Class, whose objects you want to sort, do not need to implement a comparator interface. |
| It provides single sorting sequences. | It provides multiple sorting sequences. |
| This method can sort the data according to the natural sorting order. | This method sorts the data according to the customized sorting order. |
| It affects the original class. i.e., the actual class is altered. | It doesn't affect the original class, i.e., the actual class is not altered. |
| Implemented frequently in the APIby Calendar, Wrapper classes, Date, and String. | It is implemented to sort instances of third-party classes. |
| All wrapper classes and String class implement the comparable interface. | The only implemented classes of Comparator are Collator and RuleBasedColator. |

**12) Explain equals() with example**

Equals() verifies whether the number object is equal to the object, which is passed as an argument or not.

The syntax of the equals() method is:

public boolean equals(Object o)

This method takes two parameters 1) any object, 2) return value. It returns true if the passed argument is not null and is an object of a similar type having the same numeric value.

**Example:**

import java.lang.Integer;

public class Test {

public static void main(String args[]) {

Integer p = 5;

Integer q = 20;

Integer r =5;

Short s = 5;

System.out.println(p.equals(q));

System.out.println(p.equals(r));

System.out.println(p.equals(s));

}

}

**13) List out benefits of generic collection**

The benefits of using the generic collection are:

* If the programmers are using generic class, they don't require typecasting.
* It is type-safe and can be checked at the time of compilation.
* It provides the stability of the code by detecting bug at the compilation time.

**14) Explain the method to convert ArrayList to Array and Array to ArrayList**

Programmers can convert an Array to ArrayList using asList() method of Arrays class. It is a static method of Arrays class that accept the List object. The syntax of asList() method is:

Arrays.asList(item)

Java programmers can convert ArrayList to the List object using syntax:

List\_object.toArray(new String[List\_object.size()])

**15) Give example of ArrayList**

The Example of reverse ArrayList is:

import java.util.ArrayList;

class Test\_ArrayList {

public static void main(String[] args) {

//Creating a generic ArrayList

ArrayList<String> arlTest = new ArrayList<String>();

//Size of arrayList

System.out.println("Size of ArrayList at creation: " + arlTest.size());

//Lets add some elements to it

arlTest.add("D");

arlTest.add("U");

arlTest.add("K");

arlTest.add("E");

//Recheck the size after adding elements

System.out.println("Size of ArrayList after adding elements: " + arlTest.size());

//Display all contents of ArrayList

System.out.println("List of all elements: " + arlTest);

//Remove some elements from the list

arlTest.remove("D");

System.out.println("See contents after removing one element: " + arlTest);

//Remove element by index

arlTest.remove(2);

System.out.println("See contents after removing element by index: " + arlTest);

//Check size after removing elements

System.out.println("Size of arrayList after removing elements: " + arlTest.size());

System.out.println("List of all elements after removing elements: " + arlTest);

//Check if the list contains "K"

System.out.println(arlTest.contains("K"));

}

}

**16) Give example to sort an array in dscending order**

The example of sort an array in decending order is:

package com.guru99;

public class SelectionSortAlgo {

public static void main(String a[])

{

int[] myArray = {860,8,200,9};

System.out.println("------Before Sort-----");

printArray(myArray);

selection(myArray);//sorting array using selection sort

System.out.println("-----After Sort-----");

printArray(myArray);

}

public static void selection(int[] array)

{

for (int i = 0; i < array.length - 1; i++)

{ System.out.println("Sort Pass Number "+(i+1));

int index = i;

for (int j = i + 1; j < array.length; j++)

{

System.out.println("Comparing "+ array[index] + " and " + array[j]);

if (array[j] < array[index]){

System.out.println(array[index] + " is greater than " + array[j] );

index = j;

}

}

int smallerNumber = array[index];

array[index] = array[i];

array[i] = smallerNumber;

System.out.println("Swapping Elements: New Array After Swap");

printArray(array);

}

}

static void printArray(int[] array){

for(int i=0; i < array.length; i++)

{

System.out.print(array[i] + " ");

}

System.out.println();

}

}

**17) Explain the basic interfaces of the Java collections framework**

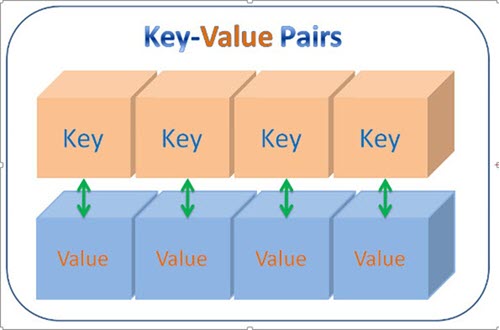
Java collection framework is a root of the collection hierarchy. It represents a group of objects as its elements. The Java programming language does not provide a direct implementation of such interface.

* **Set:** Set is a collection having no duplicate elements. It uses hashtable for storing elements.
* **List:**List is an ordered collection that can contain duplicate elements. It enables developers to access any elements from its inbox. The list is like an array having a dynamic length.
* **MAP:**It is an object which maps keys to values. It cannot contain duplicate keys. Each key can be mapped to at least one value.

**18) What are the features of Java Hashmap?**

Features of Java Hashmap are:

* The values can be stored in a map by forming a key-value pair. The value can be retrieved using the key by passing it to the correct method.
* If no element exists in the Map, it will throw a 'NoSuchElementException'.
* HashMap stores only object references. That is why it is impossible to use primitive data types like double or int. Use wrapper class (like Integer or Double) instead.

[](https://www.guru99.com/images/2/062520_0728_Top80JavaCo2.jpg)

**19) What is a Stack?**

A stack is a special area of computer's memory that stores temporary variables created by a function. In stack, variables are declared, stored, and initialized during runtime.

**20) What is linked list?**

A linked list is a data structure that can store a collection of items. In other words, linked lists can be utilized to store several objects of the same type. Each unit or element of the list is referred as a node. A node in the Linked list has its data and the address of the next node. It is like a chain. Linked Lists are used to create graphs and trees.

**21) Give example of ArrayList**

The example of ArrayList is:

import java.util.ArrayList;

class Test\_ArrayList {

public static void main(String[] args) {

//Creating a generic ArrayList

ArrayList<String> arlTest = new ArrayList<String>();

//Size of arrayList

System.out.println("Size of ArrayList at creation: " + arlTest.size());

//Lets add some elements to it

arlTest.add("D");

arlTest.add("U");

arlTest.add("K");

arlTest.add("E");

//Recheck the size after adding elements

System.out.println("Size of ArrayList after adding elements: " + arlTest.size());

//Display all contents of ArrayList

System.out.println("List of all elements: " + arlTest);

//Remove some elements from the list

arlTest.remove("D");

System.out.println("See contents after removing one element: " + arlTest);

//Remove element by index

arlTest.remove(2);

System.out.println("See contents after removing element by index: " + arlTest);

//Check size after removing elements

System.out.println("Size of arrayList after removing elements: " + arlTest.size());

System.out.println("List of all elements after removing elements: " + arlTest);

//Check if the list contains "K"

System.out.println(arlTest.contains("K"));

}

}

**22) Explain linked list supported by Java**

Two types of linked list supported by Java are:

* **Singly Linked list:** Singly Linked list is a type of data structure. In a singly linked list, each node in the list stores the contents of the node and a reference or pointer to the next node in the list. It does not store any reference or pointer to the previous node.
* **Doubly linked lists:**Doubly linked lists are a special type of linked list wherein traversal across the data elements can be done in both directions. This is made possible by having two links in every node, one that links to the next node and another one that connects to the previous node.

**23) Explain the methods provided by the Queue interface?**

Methods of Java Queue interface are:

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(object) | Inserts specified element into the Queue. It returns true in case it a success. |
| boolean offer(object) | This method is used to insert the element into the Queue. |
| Object remove() | It retrieves and removes the queue head. |
| Object poll() | **():** It retrieves and removes queue head or return null in case if it is empty. |
| Object poll() | It retrieves and removes queue head or return null in case if it is empty. |
| Object element() | Retrieves the data from the Queue, but does not remove its head. |
| Object peek() | Retrieves the data from the Queue but does not remove its head, or in case, if the Queue is the Queue is empty, it will retrieve null. |

**24) Mention the methods provided by Stack class**

Important methods provided by Stack class are:

* **push():** Push item into the stack.
* **empty ():** This method finds that whether the stack is empty or not.
* **pop ():** This Java collection framework method removes the object from the stack.
* **search ():** This method searches items in the stack.
* **peek ():** This Java method looks at the stack object without removing it.

**25) Define emptySet() in the Java collections framework**

Method emptySet() that returns the empty immutable set whenever programmers try to remove null elements. The set which is returned by emptySet() is serializable. The syntax of this method is:

public static final <T> Set<T> emptySet()

**26) Differentiate between Collection and Collections**

The difference between Collection and Collections are:

|  |  |
| --- | --- |
| Collection | Collections |
| The collection is an interface. | Collections is a class. |
| It represents a group of objects as a single entity. | It defines various utility methods for collection objects. |
| The collection is the root interface of the Java Collection framework. | Collections is a general utility class. |
| This interface is used to derive the collection data structures. | This class contains static methods to manipulate data structure. |

**27) Define LinkedHashSet in the Java Collection framework?**

LinkedHashSet is a subclass of the class called HashSet and implements the set interface. It is a well-ordered version of HashSet that maintains a doubly-linked List across its all elements.

**28) What is the difference between failfast and failsafe?**

|  |  |
| --- | --- |
| **Failfast** | **Failsafe** |
| It does not allow collection modification while iterating. | It allows collection modification while iterating. |
| It can throw ConcurrentModificationException | It can't throw any exception. |
| It uses the original collection to traverse the elements. | It uses an original collection copy to traverse the elements. |
| There is no requirement of extra memory. | There is a requirement of extra memory. |

**29) List collection views of a map interface**

Collection views of map interface are: 1) key set view, 2) value set view, and 3) entry set view.

**30) What are the benefits of the Collection Framework in Java?**

The benefits of Collection Framework in Java are:

* Java collection framework offers highly efficient and effective data structures that enhance the accuracy and speed of the program.
* The program developed with the Java collection framework is easy to maintain.
* A developer can mix classes with other types that result in increasing the reusability of code.
* The Java collection framework enables programmers to modify the primitive collection types the way they like.

**31) What is a good way to sort the Collection objects in Java?**

A good way to sort Java collection objects is using Comparable and Comparator interfaces. A developer can use Collections.sort(), the elements are sorted based on the order mention in compareTo().

When a developer uses Collections, sort (Comparator), it sorts the objects depend on compare() of the Comparator interface.

**32) Explain Vector in Java**

The vector is the same as an array. It has components that can be accessed using an index value. Vectors can contain a legacy method that is nor part of the collection framework.

**33) What is the difference between Set and Map?**

|  |  |
| --- | --- |
| **Set** | **Map** |
| Set belongs to package-java.util. | The map belongs package- java.util. |
| It can extend the collection interface. | It does not extend the collection interface. |
| It does not allow duplicate values. | It allows duplicate values. |
| Set can sort only one null value. | The map can sort multiple null values. |

**34) Define dictionary class**

The Dictionary class is a Java class that has a capability to store key-value pairs.

**35) Define EnumSet**

java.util.EnumSet is Set implementation that can be used with enum types. EnumSet having all elements must come from one enum type specified explicitly or implicitly. It is not synchronized, and also null keys are not allowed. EnumSet provides methods like EnumSetof(E first, E… rest), complementOf(EnumSet s), and copyOf(Collection c).

**36) What are the two ways to remove duplicates from ArrayList?**

Two ways to remove duplicates from ArrayList are:

* **HashSet:** Developer can use HashSet to remove the duplicate element from the ArrayList. The drawback is it cannot preserve the insertion order.
* **LinkedHashSet:** Developers can also maintain the order of insertion by using LinkedHashSet instead of HashSet.

**37) What is IdentityHashMap?**

IdentityHashMap is a class that implements Serializable, Clonable interfaces, Map, and extends AbstractMap class. It is designed for the case wherein there is a need of reference-equality semantics.

**38) What is WeakHashMap?**

WeakHashMap is an implementation of the Java Map. It is used to store weak references to its keys. Sorting using this Map allows a key-value pair is collected as garbage. Its key is not referenced outside WeakHashMap.

**39) What are the methods to make collection thread-safe?**

The methods to make collection thread safe are:

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

**40) Explain UnsupportedOperationException**

UnsupportedOperationException is an exception whch is thrown on methods that are not supported by actual collection type.

For example, Developer is making a read-only list using "Collections.unmodifiableList(list)" and calling call(), add() or remove() method. It should clearly throw UnsupportedOperationException.

**41) Name the collection classes that gives random element access to its elements**

Collection classes that give random element access to its elements are: 1) ArrayList, 2) HashMap, 3) TreeMap, and 4) Hashtable.

**42) Explain the difference between Queue and Deque.**

|  |  |
| --- | --- |
| **Queue** | **Deque** |
| It is called a single-ended Queue | It is called a double-ended Queue |
| Elements in the Queue are added or removed from one end | Elements in the Queue are added from either end can be added and removed from the both end |
| It is less versatile. | It is more versatile. |

**43) Mention the implementing List and Set interface**

**Class implementing List interface:** 1) ArrayList, 2) Vector, and 3) LinkedList.

**Class implementing Set interface:** 1) HashSet, and 2) TreeSet.

**44) Explain the design pattern followed by Iterator**

The iterator follows the detail of the iterator design pattern. It provides developer to navigate through the objects collections using a common interface without knowing its implementation.

**45) What is the peek() of the Queue interface?**

Peek () is a method of queue interface. It retrieves all the elements but does not remove the queue head. In case if the Queue is empty, then this method will return null.

**46) What is CopyOnWriteArrayList?**

CopyOnWriteArrayList is a variant of ArrayList in which operations like add and set are implemented by creating a copy of the array. It is a thread-safe, and thereby it does not throw ConcurrentModificationException. This ArrayLists permits all the elements, including null.

**47) Differentiate between ArrayList and LinkedList**

The difference between ArrayList and LinkedList is:

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| It uses a dynamic array. | It uses a doubly-linked list. |
| ArrayList is not preferable for manipulation. | LinkedList is preferable for manipulation. |
| ArrayList provides random access. | LinkedList does not provide random access. |
| ArrayList s stores only objects hence it takes less overhead of memory | LinkedList stores object as well as address object; hence, it takes more overhead of memory. |

**48) Explain the methods of iterator interface**

Methods of iterator interface are:

|  |  |
| --- | --- |
| **Method** | **Description** |
| public boolean hasNext() | It returns true in the iterator has elements; otherwise, it returns false. |
| public Object next() | This method returns the element and moves the pointer to the next value. |
| public void remove() | This Java method can remove the last elements returned by the iterator. Public void remove() is less used. |

**49) What are the methods of the HashSet class?**

Methods of HashSet class are:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| **boolean add(Object o)** | This method adds the mention element to this set if it is not already present. |
| **boolean contains(Object o):** | It returns true if the set contains the specified element. |
| **void clear():** | This method removes set elements. |
| **boolean isEmpty():** | It returns true in the case, the set has no elements. |
| **boolean remove(Object o):** | It remove the specified element from the set. |
| **object clone():** | This method returns a copy of the HashSet instance: the elements themselves are not cloned. |
| **iterator iterator()** | It returns an iterator over the elements in this set. |
| **int size():** | It returns the number of elements available in the set. |

**50) What are the methods of Java TreeSet class?**

The methods of Java TreeSet class are:

|  |  |
| --- | --- |
| **Methods** | **Descriptions** |
| boolean addAll(Collection c) | Add all the elements in the specified collection to this set. |
| boolean contains(Object o) | Returns true if the set contains the mention element. |
| boolean isEmpty() | This Java method returns true if this set contains no elements. |
| boolean remove(Object o) | Remove the specified element from the set. |
| void add(Object o) | It adds the specified element to the set. |
| void clear() | This Java method removes all the elements from the set. |

**51) Explain Linked HashSet**

Java LinkedHashSet class is a Linked list and Hash table implementation of the Set interface. It contains unique elements same as a HashSet. Linked HashSet in Java also provides optional set operations that can maintain the order of insertion.

**52) What are the important methods used in a linked list?**

The important methods used in the linked list are:

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add( Object o) | It is used to append the specified element to the end of the vector. |
| boolean contains(Object o) | It a method that returns true if this list contains the specified element. |
| void add (int index, Object element) | Inserts the element at the specified element in the vector. |
| void addFirst(Object o) | It is used to insert the given element at the beginning. |
| void addLast(Object o) | It is used to append the given element to the end. |
| Int size() | This method can be used to return the total number of elements in a list. |
| boolean remove(Object o) | It can remove the first occurrence of the specified element from this list. |
| int indexOf(Object element) | This Java method returns the index with the first occurrence of the mention element in this list, or -1. |
| int lastIndexOf(Object element) | It is a Java method that returns the index with the last occurrence of the specified element in this list, or -1. |

**53) List various classes available in sets**

Various classes available in sets are: HashSet, TreeSetand, and LinkedHashSet.

**54) List methods available in Java Queue interface**

* boolean add(object)
* boolean offer(object)
* object remove()
* object poll()
* object element()
* object peek()

**55) Differentiate between List and Set.**

|  |  |
| --- | --- |
| **List** | **Set** |
| An ordered collection of elements | An unordered collection of elements |
| Preserves the insertion order | Doesn't preserves the insertion order |
| Duplicate values are allowed | Duplicate values are not allowed |
| Any number of null values can be stored | Only one null values can be stored |
| ListIterator can be used to traverse the List in any direction | ListIterator cannot be used to traverse a Set |
| Contains a legacy class called vector | Doesn't contains any legacy class |

**56) Explain for each loop with example**

For-Each Loop is another form of for loop used to traverse the array. It reduces the code significantly, and there is no use of the index or rather the counter in the loop.

Exmple of for each loop:

class UsingForEach {

public static void main(String[] args) {

String[] arrData = {"Alpha", "Beta", "Gamma", "Delta", "Sigma"};

//The conventional approach of using the for loop

System.out.println("Using conventional For Loop:");

for(int i=0; i< arrData.length; i++){

System.out.println(arrData[i]);

}

System.out.println("\nUsing Foreach loop:");

//The optimized method of using the for loop - also called the foreach loop

for (String strTemp : arrData){

System.out.println(strTemp);

}

}

}

**57) Explain diamond operator**

Diamond operator enables the compiler to collect the type arguments of generic class. In Java SE, developer can substitute the parameterized constructor with an empty parameter sets (<>) known as diamond operator.

**58) Explain randomaccess interface**

RandomAccess interface is used by List implementations for the indication that they are supporting fast.

**59) Name the collection classes that implement random access interface**

Java.util package has classes that can implement random access interface are: CopyOnWriteArrayList, Stack, ArrayList, and Vector.

**60) How to join multiple ArrayLists?**

The list provides a addall() method multiple ArrayList in Java.

For example, consider two lists 1) areaList and 2) secondAreaList. A developer can join them using addall() like:

areaList.addAll(secondAreaList);

**61) Explain deque Interface**

Java.util.Deque is Java, an interface that extends Queue interface. It gives support for the insertion and deletion of elements at both the end. This Queue is also called a double-ended queue.

**62) Explain Linkedhashmap**

LinkedHashMap is the implementation of the Map interface. It can also extends the HashMap class. Therefore, like HashMap, LinkedHashMap enables Java developers to allow one null key and more than one null value.

**63) Explain methods to remove elements from ArrayList**

The methods to remove elements from ArrayList are:

|  |  |
| --- | --- |
| **Method** | **Description** |
| clear() | This method removes the elements from ArrayList. |
| remove(int index) | This method of ArrayList can remove the element at a particular position. |
| remove(Object o) | It can remove the first occurrence of the mention element from the ArrayList. |
| removeAll() | It can remove the list of elements that are in a particular collection. |
| removeIf(Predicate<? super E> filter) | This method removes elements that satisfy the mention of a predicate. |

**64) Explain map. entry In Map**

Map.entry is a Java interface of java.util. It has a nested interface in Map. This interface must be qualified by the name of class or interface, which it is a member. Therefore it is qualified as a Map. Entry. It represents a key and value pair that can forms element of a Map.

This method returns a view of the collection. For example, consider cityMap as a map. The developer can use entrySet() to get the set view of map having an element Map.Entry. Programmer can also use getKey() and getValue() of the Map.Entry to get the pair of key and value of the map.

**65) Which method is used to sort an array in ascending order?**

Java collection framework method, Collections.sort() is used to sort an array in ascending order.

**66) How to measure the performance of an ArrayList?**

The performance of ArrayList can be measure by:

* **Adding an element:** Developer can add an element at the end of ArrayList using add(E e) method. It is O(1). In the worst scenario, it might go to O(n). This can happen if the developer add more elements than the array capacity.
* **Retrieving an element**: - Developer can access the array index using get(int index). The performance, in this case, can be measure using ArrayList get() is O(1).
* **Removing an element:** In case, if the developers are removing element using the remove(int index), then the performance of ArrayList can be calculated using said remove(int index) operation is O(n - index) method.

**67) Explain LinkedList class**

LinkedList class in Java implements Deque and List using a doubly linked list. There is a private class node in a doubly-linked list which provides its structure. It also has an item variable for holding the value and reference to Node class. This can be used for connecting the next and previous nodes.

**68) Give an example of Hashmap**

The example of Hashmap is:

import java.util.HashMap;

import java.util.Map;

public class Sample\_TestMaps{

public static void main(String[] args){

Map<String, String> objMap = new HashMap<String, String>();

objMap.put("Name", "Suzuki");

objMap.put("Power", "220");

objMap.put("Type", "2-wheeler");

objMap.put("Price", "85000");

System.out.println("Elements of the Map:");

System.out.println(objMap);

}

}

**69) How to iterate map?**

The developer cannot directly iterate map, but, this interface has two methods that gives view set of map. These methods are:

* **Set<Map.Entry<K, V>>entrySet():** It is a method that returns a set having the entries mention in the map. These entries are generally objected, which has type Map. Entry.
* **Set<K>keySet():** This Java method returns a set that having the map key.

**70) Explain Treemap in Java**

TreeMap is a class that implements the Map interface LinkedHashMap and HashMap. It can also implements the NavigableMap interface and can extends the AbstractMap class.

**71) What is the difference between Hashmap and Hashtable?**

|  |  |
| --- | --- |
| **Hashmap** | **Hashtable** |
| It is not synchronized. | It is synchronized. |
| HashMap allows one key as a null value. | HashTable does not allow null values. |
| Iterator is used to traverse HashMap. | Either Iterator or Enumerator is used for traversing a HashTable. |
| It can be used for both HashTable, HashMap and is fail-fast. | It can be used with HashTable and is fail-safe. |
| HashMap perform faster than the HashTable. | Hashtable is not much faster as compared to HashMap. |

**72) Explain the internal working of HashSet in Java**

HashSet in Java internally uses HashMap to store elements. It can also store unique values with no duplicate values.

In Java, HashSet developer can have add(E e) method that takes just the element to add as a parameter. It does not accept the key and value pair.

**73) Explain Big-O notation with an example**

The Big-O notation depicts the performance of an algorithm as the number of elements in ArrayList. A developer can use Big-O notation to choose the collection implementation. It is based on performance, time, and memory.

For example, ArrayList get(index i) is a method to perform a constant-time operation. It does not depend on the total number of elements available in the list. Therefore, the performance in Big-O notation is O(1).

**74) Explain the best practices in Java Collection Framework**

The best practices in Java Collection Framework are:

* Choose the correct type of collection depends on the need.
* Avoid rehashing or resizing by estimating the total number of elements to be stored in collection classes.
* Write a Java program in terms of interfaces. This will help the developer to change it's implementation effortlessly in the future.
* A developer can use Generics for type-safety.
* Use immutable classes given by the Java Development Kit. Avoid implementation of equals() and hashCode() for custom classes.
* A programmer should use the Collections utility class for algorithms or to get read-only, synchronized, or empty collections. This will enhance code reusability with low maintainability.

**75) Explain various types of queues in Java**

There are three types of queues in Java:

* **Priority queue:** It is a special type of Queue wherein elements are sorted as per their natural ordering or custom comparator.
* **Circular Queue:** It is a type of Queue in which user operations are performed based on the FIFO method. The last element is connected to the first position in order to make a circle.
* **Double-ended Queue:**A double-ended queue is an abstract data type that generalizes a queue. The elements in this queue can be added or removed from either head or tail.

**76) What is the difference between stack and Queue?**

|  |  |
| --- | --- |
| **Stack** | **Queue** |
| The working principle of the stack is LIFO. | Working principale of queue is FIFO. |
| One end is used to perform the insertion or deletion of elements. | One end is used to perform insertion, and another end is used for the deletion of elements. |
| It uses one pointer. | It uses two pointers in a simple queue. |
| It does not have any kind of variant. | It has variants like priority queue, circular Queue, doubly ended Queue. |
| It is easy to use. | It is not easy to use. |

**77) What is the difference between array and stack?**

The difference between array and stack is:

|  |  |
| --- | --- |
| **Array** | **Stack** |
| It is a collection of elements that are identified by the index. | It is a collection operation that serve as operations push and pop. |
| It has a elements of data types which are same. | It has a elements of data types which are different. |
| Elements can be removed or added into the array using random access operation. | Elements can be removed or added into a stack using LIFO operation. |

**78) Define Iterator()**

The Iterator() is an interface that provides methods to iterate Collection. Iterator can take the place of Enumeration in Java. It allows the caller to remove elements from the collection. The method provides a generic way for traversal using elements of the collection and implementing iterator design pattern.

**79) What are the various ways to iterate over a list?**

Java collection Framework programmer can iterate over a list in two ways: 1) Using iterator, and 2) using it for each loop.

**80) What are the advantages of the stack?**

The advantages of the stack are:

* It helps you to manage the data in a Last In First Out (LIFO) method, which is not possible with the Linked list and array.
* When a function is called, the local variables are stored in a stack, and it is automatically destroyed once returned.
* A stack is used when a variable is not used outside that function.
* It allows you to control how memory is allocated and deallocated.
* Stack automatically cleans up the object.
* Not easily corrupted
* Variables cannot be resized.

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

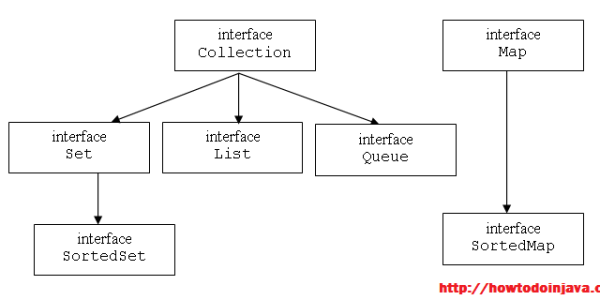
By definition, a collection is **an object that represents a group of objects**. Like in set theory, a set is group of elements. Easy enough !!

Prior to JDK 1.2, JDK has some utility classes such as Vector and HashTable, but there was no concept of Collection framework. Later from JDK 1.2 onwards, JDK felt the need of having a consistent support for reusable data structures. Finally, the collections framework was designed and developed primarily by Joshua Bloch, and was **introduced in JDK 1.2**.

Its most noticeable **benefits of java collections** can be listed as:

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

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

Java Collection Hierarchy

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

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

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

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

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

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

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

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

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

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

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

**Java collection interview – List interface questions**

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

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

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

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

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

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

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

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

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

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

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

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

**Java collection interview – Set interface questions**

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

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

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

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

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

**9) How HashSet store elements?**

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

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

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

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

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

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

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

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

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

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

**Java collection interview – Map interface questions**

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

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

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

**12) What are IdentityHashMap and WeakHashMap?**

**IdentityHashMap** is similar to HashMap except that**it uses reference equality when comparing elements**. IdentityHashMap class is not a widely used Map implementation. While this class implements the Map interface, it intentionally violates Map’s general contract, which mandates the use of the equals() method when comparing objects. IdentityHashMap is designed for use only in the rare cases wherein reference-equality semantics are required.

**WeakHashMap** is an implementation of the Map interface **that stores only weak references to its keys**. Storing only weak references allows a key-value pair to be garbage collected when its key is no longer referenced outside of the WeakHashMap. This class is intended primarily for use with key objects whose equals methods test for object identity using the == operator. Once such a key is discarded it can never be recreated, so it is impossible to do a look-up of that key in a WeakHashMap at some later time and be surprised that its entry has been removed.

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

*Taking from java docs:*

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

Read more about how [**ConcurrentHashMap interview questions**](https://howtodoinjava.com/interview-questions/hashmap-concurrenthashmap-interview-questions/).

**14) How hashmap works?**

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

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

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

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

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

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

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

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

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

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

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

**16) What are different Collection views provided by Map interface?**

Map interface provides 3 views of key-values pairs stored in it:

* key set view
* value set view
* entry set view

All the views can be navigated using iterators.

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

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

TreeMap is special form of HashMap. **It maintains the ordering of keys** which is missing in HashMap class. This ordering is **by default “natural ordering”**. The default ordering can be override by providing an instance of Comparator class, whose compare method will be used to maintain ordering of keys.

Please note that **all keys inserted into the map must implement the Comparable interface** (this is necessary to decide the ordering). Furthermore, all such keys must be mutually comparable: k1.compareTo(k2) must not throw a ClassCastException for any keys k1 and k2 in the map. If the user attempts to put a key into the map that violates this constraint (for example, the user attempts to put a string key into a map whose keys are integers), the put(Object key, Object value) call will throw a ClassCastException.

**Java collection interview – Tell the difference questions**

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

The most noticeable differences are :

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

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

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

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

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

There are several differences between HashMap and Hashtable in Java:

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

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

Lets note down the differences:

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

**22) Difference between Iterator and Enumeration?**

Iterators differ from enumerations in three ways:

* Iterators allow the caller to remove elements from the underlying collection during the iteration with its remove() method. You can not add/remove elements from a collection when using enumerator.
* Enumeration is available in legacy classes i.e Vector/Stack etc. whereas Iterator is available in all modern collection classes.
* Another minor difference is that Iterator has improved method names e.g. Enumeration.hasMoreElement() has become Iterator.hasNext(), Enumeration.nextElement() has become Iterator.next() etc.

**23) Difference between HashMap and HashSet?**

HashMap is collection of key-value pairs whereas HashSet is un-ordered collection of unique elements. That’s it. No need to describe further.

**24) Difference between Iterator and ListIterator?**

There are three Differences are there:

* We can use Iterator to traverse Set and List and also Map type of Objects. But List Iterator can be used to traverse for List type Objects, but not for Set type of Objects.
* By using Iterator we can retrieve the elements from Collection Object in forward direction only whereas List Iterator, which allows you to traverse in either directions using hasPrevious() and previous() methods.
* ListIterator allows you modify the list using add() remove() methods. Using Iterator you can not add, only remove the elements.

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

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

**26) Difference between ArrayList and LinkedList?**

* LinkedList store elements within a doubly-linked list data structure. ArrayList store elements within a dynamically resizing array.
* LinkedList allows for constant-time insertions or removals, but only sequential access of elements. In other words, you can walk the list forwards or backwards, but grabbing an element in the middle takes time proportional to the size of the list. ArrayLists, on the other hand, allow random access, so you can grab any element in constant time. But adding or removing from anywhere but the end requires shifting all the latter elements over, either to make an opening or fill the gap.
* LinkedList has more memory overhead than ArrayList because in ArrayList each index only holds actual object (data) but in case of LinkedList each node holds both data and address of next and previous node.

**More collection interview questions**

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

Use following methods:

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

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

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

Use below methods:

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

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

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

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

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

You can iterate over a list using following ways:

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

Read more : [http://www.mkyong.com/java/how-do-loop-iterate-a-list-in-java/](https://mkyong.com/java/how-do-loop-iterate-a-list-in-java/)

**31) What do you understand by iterator fail-fast property?**

**Fail-fast Iterators fail as soon as they realized that structure of Collection has been changed since iteration has begun**. Structural changes means adding, removing or updating any element from collection while one thread is Iterating over that collection.

Fail-fast behavior is implemented by keeping a modification count and if iteration thread realizes the change in modification count it throws ConcurrentModificationException.

**32) What is difference between fail-fast and fail-safe?**

You have understood fail-fast in previous question. **Fail-safe iterators** are just opposite to fail-fast. **They never fail if you modify the underlying collection on which they are iterating**, because they work on clone of Collection instead of original collection and that’s why they are called as fail-safe iterator.

Iterator of CopyOnWriteArrayList is an example of fail-safe Iterator also iterator written by ConcurrentHashMap keySet is also fail-safe iterator and never throw ConcurrentModificationException.

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

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

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

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

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

**34) What is UnsupportedOperationException?**

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

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

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

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

**36) What is BlockingQueue?**

**A Queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element.**

BlockingQueue methods come in four forms: one throws an exception, the second returns a special value (either null or false, depending on the operation), the third blocks the current thread indefinitely until the operation can succeed, and the fourth blocks for only a given maximum time limit before giving up.

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

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

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

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

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

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

**38) What is Comparable and Comparator interface?**

In java. all collection which have feature of automatic sorting, uses compare methods to ensure the correct sorting of elements. For example classes which use sorting are TreeSet, TreeMap etc.

**To sort the data elements a class needs to implement Comparator or Comparable interface**. That’s why all Wrapper classes like Integer,Double and String class implements Comparable interface.

**Comparable helps in preserving default natural sorting, whereas Comparator helps in sorting the elements in some special required sorting pattern.** The instance of comparator if passed usually as collection’s constructor argument in supporting collections.

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

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

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

**40) Recommended resources**

Well it is not interview question.. :-). This is only for fun. But you should really read my blog for more posts on collection framework knowledge.

I hope these java collection interview questions will help in in your next interview. Further, I will suggest you to read more on above questions apart from this post. A more knowledge will only help you.

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

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

* 1. [Java Stream API](https://www.journaldev.com/2774/java-8-stream) for collection classes for supporting sequential as well as parallel processing
  2. [Iterable interface is extended with forEach()](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 its argument Consumer is a [function interface](https://www.journaldev.com/2763/java-8-functional-interfaces).
  3. Miscellaneous Collection API improvements such as forEachRemaining(Consumer action) method in Iterator interface, Map replaceAll(), compute(), merge() methods.

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

Collections are used in every programming language and initial java release contained few classes for collections: **Vector**, **Stack**, **Hashtable**, **Array**. But looking at the larger scope and usage, Java 1.2 came up with Collections Framework that group all the collections interfaces, implementations and algorithms.  
Java Collections have come through a long way with the 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) 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](https://www.journaldev.com/1260/collections-in-java-tutorial#list-interface) is an ordered collection and can contain duplicate elements. You can access any element from its index. The list is more like an array with dynamic length.

A [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 doesn’t 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 less flexible and more restrictive. The specific implementation should decide as to whether it can be cloned or serialized.

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

Although Map interface and its implementations are part of the Collections Framework, Map is 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? The map contains key-value pairs and it provides methods to retrieve the list of Keys or values as Collection but it doesn’t fit into the “group of elements” paradigm.

1. **What is an Iterator?**

The 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**](https://www.journaldev.com/1716/iterator-design-pattern-java).

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

Enumeration is twice as fast as Iterator and uses very little memory. Enumeration is very basic and fits basic needs. But the 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 its functionality clear.

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

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

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

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

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

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

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

//using for-each loop

for(String obj : strList){

System.out.println(obj);

}

//using iterator

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

while(it.hasNext()){

String obj = it.next();

System.out.println(obj);

}

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

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

Iterator fail-fast property checks for any modification in the structure of the underlying collection everytime we try to get the next element. If there are any modifications found, it throws ConcurrentModificationException. All the implementations of Iterator in Collection classes are fail-fast by design except the concurrent collection classes like ConcurrentHashMap and CopyOnWriteArrayList.

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

Iterator fail-safe property work with the clone of underlying collection, hence it’s not affected by any modification in the collection. By design, all the collection classes in java.util package are fail-fast whereas collection classes in java.util.concurrent are fail-safe.  
Fail-fast iterators throw ConcurrentModificationException whereas fail-safe iterator never throws ConcurrentModificationException.  
Check this post for [CopyOnWriteArrayList Example](https://www.journaldev.com/1289/copyonwritearraylist-java).

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

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

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

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

1. **What is UnsupportedOperationException?**

UnsupportedOperationException is the exception used to indicate that the operation is not supported. It’s used extensively in [JDK](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 is an 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 stores 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 its value. The below image will explain these detail clearly.

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. The threshold is capacity multiplied by load factor and whenever we try to add an entry if map size is greater than the threshold, HashMap rehashes the contents of the map into a new array with a larger capacity. The capacity is always the 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 the database, it’s a good idea to initialize the HashMap with correct capacity and load factor.

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

HashMap uses the 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 the same hashCode() and equals() output and in that case, rather than storing it at a different location, HashMap will consider the same and overwrite them.

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

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

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

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

* 1. If the class overrides equals() method, it should also override hashCode() method.
  2. The class should follow the rules associated with equals() and hashCode() for all instances. Please refer earlier question for these rules.
  3. If a class field is not used in equals(), you should not use it in hashCode() method.
  4. Best practice for user defined key class is to make it immutable, so that hashCode() value can be cached for fast performance. Also immutable classes make sure that hashCode() and equals() will not change in future that will solve any issue with mutability.  
     For example, let’s say I have a class MyKey that I am using for the 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 its 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 the 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:

* 1. **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 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<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 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 remove operation, or the setValue operation on a map entry returned by the iterator) the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the Iterator remove, Set.remove, removeAll, retainAll, and clear operations. It does not support the add or addAll operations.

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

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

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

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

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

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

ArrayList and Vector are similar classes in many ways.

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

These are the differences between ArrayList and Vector.

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

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

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

Although ArrayList is the obvious choice when we work on the list, there are a few times when an array is 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 its elements with performance as O(1) but LinkedList stores data as list of nodes where every node is linked to its 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 its elements?**

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

1. **What is EnumSet?**

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

Check this post for [java enum tutorial](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.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](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/concurrenthashmap-in-java)

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 the java collections framework and it’s primarily used for implementing the 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 consumers 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.  
The 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 a Comparable interface which should be implemented by any custom class if we want to use Arrays or Collections sorting methods. The comparable interface has a 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 an 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 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 choose 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 the first argument is less than the second one and returns zero if they are equal and positive int if the first argument is greater than the 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 choose 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 the number of elements in a data structure. Since Collection classes are 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 its 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 LinkedHashMap. 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 its 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 the last question. So if we would have been allowed to create generic arrays, because of type erasure we would not get an array store exception even though both types are not related.

1) What is the Collection framework in Java?

Collection Framework is a combination of classes and interface, which is used to store and manipulate the data in the form of objects. It provides various classes such as ArrayList, Vector, Stack, and HashSet, etc. and interfaces such as List, Queue, Set, etc. for this purpose.

2) What are the main differences between array and collection?

Array and Collection are somewhat similar regarding storing the references of objects and manipulating the data, but they differ in many ways. The main differences between the array and Collection are defined below:

* Arrays are always of fixed size, i.e., a user can not increase or decrease the length of the array according to their requirement or at runtime, but In Collection, size can be changed dynamically as per need.
* Arrays can only store homogeneous or similar type objects, but in Collection, heterogeneous objects can be stored.
* Arrays cannot provide the ?ready-made? methods for user requirements as sorting, searching, etc. but Collection includes readymade methods to use.

3) Explain various interfaces used in Collection framework?

Collection framework implements various interfaces, Collection interface and Map interface (java.util.Map) are the mainly used interfaces of Java Collection Framework. List of interfaces of Collection Framework is given below:

**1. Collection interface:** Collection (java.util.Collection) is the primary interface, and every collection must implement this interface.

**Syntax:**

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

Where <E> represents that this interface is of Generic type

**2. List interface:**List interface extends the Collection interface, and it is an ordered collection of objects. It contains duplicate elements. It also allows random access of elements.

**Syntax:**

1. **public** **interface** List<E> **extends** Collection<E>

**3. Set interface:** Set (java.util.Set) interface is a collection which cannot contain duplicate elements. It can only include inherited methods of Collection interface

**Syntax:**

1. **public** **interface** Set<E> **extends** Collection<E>

**Queue interface:**Queue (java.util.Queue) interface defines queue data structure, which stores the elements in the form FIFO (first in first out).

**Syntax:**

1. **public** **interface** Queue<E> **extends** Collection<E>

**4. Dequeue interface:** it is a double-ended-queue. It allows the insertion and removal of elements from both ends. It implants the properties of both Stack and queue so it can perform LIFO (Last in first out) stack and FIFO (first in first out) queue, operations.

**Syntax:**

1. **public** **interface** Dequeue<E> **extends** Queue<E>

**5. Map interface:**A Map (java.util.Map) represents a key, value pair storage of elements. Map interface does not implement the Collection interface. It can only contain a unique key but can have duplicate elements. There are two interfaces which implement Map in java that are Map interface and Sorted Map.

4) What is the difference between ArrayList and Vector?

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **Vector** |
| 1) | ArrayList is not synchronized. | Vector is synchronized. |
| 2) | ArrayList is not a legacy class. | Vector is a legacy class. |
| 3) | ArrayList increases its size by 50% of the array size. | Vector increases its size by doubling the array size. |
| 4) | ArrayList is not ?thread-safe? as it is not synchronized. | Vector list is ?thread-safe? as it?s every method is synchronized. |

5) What is the difference between ArrayList and LinkedList?

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **LinkedList** |
| 1) | ArrayList uses a dynamic array. | LinkedList uses a doubly linked list. |
| 2) | ArrayList is not efficient for manipulation because too much is required. | LinkedList is efficient for manipulation. |
| 3) | ArrayList is better to store and fetch data. | LinkedList is better to manipulate data. |
| 4) | ArrayList provides random access. | LinkedList does not provide random access. |
| 5) | ArrayList takes less memory overhead as it stores only object | LinkedList takes more memory overhead, as it stores the object as well as the address of that object. |

6) What is the difference between Iterator and ListIterator?

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

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **ListIterator** |
| 1) | The Iterator traverses the elements in the forward direction only. | ListIterator traverses the elements in backward and forward directions both. |
| 2) | The Iterator can be used in List, Set, and Queue. | ListIterator can be used in List only. |
| 3) | The Iterator can only perform remove operation while traversing the collection. | ListIterator can perform ?add,? ?remove,? and ?set? operation while traversing the collection. |

7) What is the difference between Iterator and Enumeration?

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **Enumeration** |
| 1) | The Iterator can traverse legacy and non-legacy elements. | Enumeration can traverse only legacy elements. |
| 2) | The Iterator is fail-fast. | Enumeration is not fail-fast. |
| 3) | The Iterator is slower than Enumeration. | Enumeration is faster than Iterator. |
| 4) | The Iterator can perform remove operation while traversing the collection. | The Enumeration can perform only traverse operation on the collection. |

8) What is the difference between List and Set?

The List and Set both extend the collection interface. However, there are some differences between the both which are listed below.

* The List can contain duplicate elements whereas Set includes unique items.
* The List is an ordered collection which maintains the insertion order whereas Set is an unordered collection which does not preserve the insertion order.
* The List interface contains a single legacy class which is Vector class whereas Set interface does not have any legacy class.
* The List interface can allow n number of null values whereas Set interface only allows a single null value.

9) What is the difference between HashSet and TreeSet?

The HashSet and TreeSet, both classes, implement Set interface. The differences between the both are listed below.

* HashSet maintains no order whereas TreeSet maintains ascending order.
* HashSet impended by hash table whereas TreeSet implemented by a Tree structure.
* HashSet performs faster than TreeSet.
* HashSet is backed by HashMap whereas TreeSet is backed by TreeMap.

10) What is the difference between Set and Map?

The differences between the Set and Map are given below.

* Set contains values only whereas Map contains key and values both.
* Set contains unique values whereas Map can contain unique Keys with duplicate values.
* Set holds a single number of null value whereas Map can include a single null key with n number of null values.

11) What is the difference between HashSet and HashMap?

The differences between the HashSet and HashMap are listed below.

* HashSet contains only values whereas HashMap includes the entry (key, value). HashSet can be iterated, but HashMap needs to convert into Set to be iterated.
* HashSet implements Set interface whereas HashMap implements the Map interface
* HashSet cannot have any duplicate value whereas HashMap can contain duplicate values with unique keys.
* HashSet contains the only single number of null value whereas HashMap can hold a single null key with n number of null values.

12) What is the difference between HashMap and TreeMap?

The differences between the HashMap and TreeMap are given below.

* HashMap maintains no order, but TreeMap maintains ascending order.
* HashMap is implemented by hash table whereas TreeMap is implemented by a Tree structure.
* HashMap can be sorted by Key or value whereas TreeMap can be sorted by Key.
* HashMap may contain a null key with multiple null values whereas TreeMap cannot hold a null key but can have multiple null values.

13) What is the difference between HashMap and Hashtable?

|  |  |  |
| --- | --- | --- |
| **No.** | **HashMap** | **Hashtable** |
| 1) | HashMap is not synchronized. | Hashtable is synchronized. |
| 2) | HashMap can contain one null key and multiple null values. | Hashtable cannot contain any null key or null value. |
| 3) | HashMap is not ?thread-safe,? so it is useful for non-threaded applications. | Hashtable is thread-safe, and it can be shared between various threads. |
| 4) | 4) HashMap inherits the AbstractMap class | Hashtable inherits the Dictionary class. |

14) What is the difference between Collection and Collections?

The differences between the Collection and Collections are given below.

* The Collection is an interface whereas Collections is a class.
* The Collection interface provides the standard functionality of data structure to List, Set, and Queue. However, Collections class is to sort and synchronize the collection elements.
* The Collection interface provides the methods that can be used for data structure whereas Collections class provides the static methods which can be used for various operation on a collection.

15) What is the difference between Comparable and Comparator?

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

16) What do you understand by BlockingQueue?

BlockingQueue is an interface which extends the Queue interface. It provides concurrency in the operations like retrieval, insertion, deletion. While retrieval of any element, it waits for the queue to be non-empty. While storing the elements, it waits for the available space. BlockingQueue cannot contain null elements, and implementation of BlockingQueue is thread-safe.

**Syntax:**

1. **public** **interface** BlockingQueue<E> **extends** Queue <E>

17) What is the advantage of Properties file?

If you change the value in the properties file, you don't need to recompile the java class. So, it makes the application easy to manage. It is used to store information which is to be changed frequently. Consider the following example.

1. **import** java.util.\*;
2. **import** java.io.\*;
3. **public** **class** Test {
4. **public** **static** **void** main(String[] args)**throws** Exception{
5. FileReader reader=**new** FileReader("db.properties");
7. Properties p=**new** Properties();
8. p.load(reader);
10. System.out.println(p.getProperty("user"));
11. System.out.println(p.getProperty("password"));
12. }
13. }

**Output**

system

oracle

18) What does the hashCode() method?

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

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

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

If two objects do not produce an equal result by using the equals() method, then the hashcode() method will provide the different integer result for both the objects.

19) Why we override equals() method?

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

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

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

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

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

21) What is the advantage of the generic collection?

There are three main advantages of using the generic collection.

* If we use the generic class, we don't need typecasting.
* It is type-safe and checked at compile time.
* Generic confirms the stability of the code by making it bug detectable at compile time.

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

Two different keys with the same hash value are known as hash-collision. Two separate entries will be kept in a single hash bucket to avoid the collision. There are two ways to avoid hash-collision.

* Separate Chaining
* Open Addressing

23) What is the Dictionary class?

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

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

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

25) What do you understand by fail-fast?

The Iterator in java which immediately throws ConcurrentmodificationException, if any structural modification occurs in, is called as a Fail-fast iterator. Fail-fats iterator does not require any extra space in memory.

26) [What is the difference between Array and ArrayList?](https://www.javatpoint.com/array-vs-arraylist-in-java)

The main differences between the Array and ArrayList are given below.

|  |  |  |
| --- | --- | --- |
| **SN** | **Array** | **ArrayList** |
| 1 | The Array is of fixed size, means we cannot resize the array as per need. | ArrayList is not of the fixed size we can change the size dynamically. |
| 2 | Arrays are of the static type. | ArrayList is of dynamic size. |
| 3 | Arrays can store primitive data types as well as objects. | ArrayList cannot store the primitive data types it can only store the objects. |

27) [What is the difference between the length of an Array and size of ArrayList?](https://www.javatpoint.com/difference-between-length-of-array-and-size-of-arraylist-in-java)

The length of an array can be obtained using the property of length whereas ArrayList does not support length property, but we can use size() method to get the number of objects in the list.

**Finding the length of the array**

1. Int [] array = **new** **int**[4];
2. System.out.println("The size of the array is " + array.length);

**Finding the size of the ArrayList**

1. ArrayList<String> list=**new** ArrayList<String>();
2. list.add("ankit");
3. list.add("nippun");
4. System.out.println(list.size());

28) [How to convert ArrayList to Array and Array to ArrayList?](https://www.javatpoint.com/how-to-convert-arraylist-to-array-and-array-to-arraylist-in-java)

We can convert an Array to ArrayList by using the asList() method of Arrays class. asList() method is the static method of Arrays class and accepts the List object. Consider the following syntax:

1. Arrays.asList(item)

We can convert an ArrayList to Array using toArray() method of the ArrayList class. Consider the following syntax to convert the ArrayList to the List object.

1. List\_object.toArray(**new** String[List\_object.size()])

29) [How to make Java ArrayList Read-Only?](https://www.javatpoint.com/how-to-make-java-arraylist-read-only)

We can obtain java ArrayList Read-only by calling the Collections.unmodifiableCollection() method. When we define an ArrayList as Read-only then we cannot perform any modification in the collection through  add(), remove() or set() method.

30) [How to remove duplicates from ArrayList?](https://www.javatpoint.com/how-to-remove-duplicates-from-arraylist-in-java)

There are two ways to remove duplicates from the ArrayList.

* **Using HashSet:** By using HashSet we can remove the duplicate element from the ArrayList, but it will not then preserve the insertion order.
* **Using LinkedHashSet:** We can also maintain the insertion order by using LinkedHashSet instead of HashSet.

The Process to remove duplicate elements from ArrayList using the LinkedHashSet:

* Copy all the elements of ArrayList to LinkedHashSet.
* Empty the ArrayList using clear() method, which will remove all the elements from the list.
* Now copy all the elements of LinkedHashset to ArrayList.

31) [How to reverse ArrayList?](https://www.javatpoint.com/how-to-reverse-arraylist-in-java)

To reverse an ArrayList, we can use reverse() method of Collections class. Consider the following example.

1. **import** java.util.ArrayList;
2. **import** java.util.Collection;
3. **import** java.util.Collections;
4. **import** java.util.Iterator;
5. **import** java.util.List;
6. **public** **class** ReverseArrayList {
7. **public** **static** **void** main(String[] args) {
8. List list = **new** ArrayList<>();
9. list.add(10);
10. list.add(50);
11. list.add(30);
12. Iterator i = list.iterator();
13. System.out.println("printing the list....");
14. **while**(i.hasNext())
15. {
16. System.out.println(i.next());
17. }
18. Iterator i2 = list.iterator();
19. Collections.reverse(list);
20. System.out.println("printing list in reverse order....");
21. **while**(i2.hasNext())
22. {
23. System.out.println(i2.next());
24. }
25. }
26. }

**Output**

printing the list....

10

50

30

printing list in reverse order....

30

50

10

32) [How to sort ArrayList in descending order?](https://www.javatpoint.com/how-to-sort-java-arraylist-in-descending-order)

To sort the ArrayList in descending order, we can use the reverseOrder method of Collections class. Consider the following example.

1. **import** java.util.ArrayList;
2. **import** java.util.Collection;
3. **import** java.util.Collections;
4. **import** java.util.Comparator;
5. **import** java.util.Iterator;
6. **import** java.util.List;
8. **public** **class** ReverseArrayList {
9. **public** **static** **void** main(String[] args) {
10. List list = **new** ArrayList<>();
11. list.add(10);
12. list.add(50);
13. list.add(30);
14. list.add(60);
15. list.add(20);
16. list.add(90);
18. Iterator i = list.iterator();
19. System.out.println("printing the list....");
20. **while**(i.hasNext())
21. {
22. System.out.println(i.next());
23. }
25. Comparator cmp = Collections.reverseOrder();
26. Collections.sort(list,cmp);
27. System.out.println("printing list in descending order....");
28. Iterator i2 = list.iterator();
29. **while**(i2.hasNext())
30. {
31. System.out.println(i2.next());
32. }
34. }
35. }

**Output**

printing the list....

10

50

30

60

20

90

printing list in descending order....

90

60

50

30

20

10

33) [How to synchronize ArrayList?](https://www.javatpoint.com/how-to-synchronize-arraylist-in-java)

We can synchronize ArrayList in two ways.

* Using Collections.synchronizedList() method
* Using CopyOnWriteArrayList<T>

34) [When to use ArrayList and LinkedList?](https://www.javatpoint.com/when-to-use-arraylist-and-linkedlist-in-java)

LinkedLists are better to use for the update operations whereas ArrayLists are better to use for the search operations.

**1. What are the advantages of the Collection Framework in Java?**

Below table contains the major advantages of the Java Collection Framework:

|  |  |
| --- | --- |
| **Feature** | **Description** |
| *Performance* | The collection framework provides highly effective and efficient data structures that result in enhancing the speed and accuracy of a program. |
| *Maintainability* | The code developed with the collection framework is easy to maintain as it supports data consistency and interoperability within the implementation. |
| *Reusability* | The classes in Collection Framework can effortlessly mix with other types which results in increasing the code reusability. |
| *Extensibility* | The Collection Framework in Java allows the developers to customize the primitive collection types as per their requirements. |

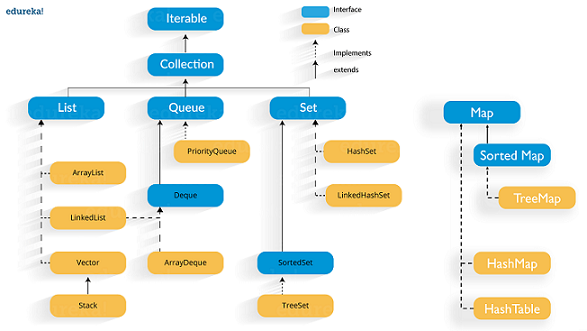
**2. What do you understand by Collection Framework in Java?**

The Java Collection framework provides an architecture to store and manage a group of objects. It permits the developers to access prepackaged data structures as well as algorithms to manipulate data. The collection framework includes the following:

* Interfaces
* [Classes](https://www.edureka.co/blog/java-objects-and-classes/)
* Algorithm

All these classes and interfaces support various operations such as Searching, Sorting, Insertion, Manipulation, and Deletion which makes the data manipulation really easy and quick.

**3. Describe the Collection hierarchy in Java.**



**4. List down the primary interfaces provided by Java Collections Framework?**

Below are the major interfaces provided by the Collection Framework:

* *Collection Interface*: java.util.Collection is the root of the Java Collection framework and most of the collections in Java are inherited from this [interface](https://www.edureka.co/blog/java-interface/).

|  |  |
| --- | --- |
| 1 | public interface Collection<E>extends Iterable |

* *List Interface*: java.util.List is an extended form of an array that contains ordered elements and may include duplicates. It supports the index-based search, but elements can be easily inserted irrespective of the position. The List interface is implemented by various classes such as ArrayList, LinkedList, Vector, etc.

|  |  |
| --- | --- |
| 1 | public interface List<E> extends Collection<E> |

* *Set Interface*: java.util.Set refers to a collection class that cannot contain duplicate elements. Since it doesn’t define an order for the elements, the index-based search is not supported. It is majorly used as a mathematical set abstraction model. The Set interface is implemented by various classes such as HashSet, TreeSetand LinkedHashSet.

|  |  |
| --- | --- |
| 1 | public interface Set<E> extends Collection<E> |

* *Queue Interface*: java.util.Queue in Java follows a FIFO approach i.e. it orders the elements in First In First Out manner. Elements in Queue will be added from the rear end while removed from the front.

|  |  |
| --- | --- |
| 1 | public interface Queue<E> extends Collection<E> |

* *Map Interface*: java.util.Map is a two-dimensional data structure in Java that is used to store the data in the form of a Key-Value pair. The key here is the unique hashcode and value represent the element. Map in Java is another form of the Java Set but can’t contain duplicate elements.

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

The Collection interface in Java specifies a group of objects called elements. The maintainability and ordering of elements is completely dependent on the concrete implementations provided by each of the Collection. Thus, there is no use of extending the Cloneable and Serializable interfaces.

**6. List down the major advantages of the Generic Collection.**

Below are the main advantages of using the [generic collection](https://www.edureka.co/blog/generics-in-java/) in Java:

* Provides stronger type checks at the time of compilation
* Eliminates the need for typecasting
* Enables the implementation of generic algorithms which makes the code customizable, type-safe and easier to read

**7. What is the main benefit of using the Properties file?**

The main advantage of using the properties file in Java is that in case the values in the properties file is changed it will be automatically reflected without having to recompile the java class.  Thus it is mainly used to store information which is liable to change such as username and passwords. This makes the management of the application easy and efficient. Below is an example of the same:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | import java.util.\*;  import java.io.\*;  public class PropertiesDemo{  public static void main(String[] args)throws Exception{  FileReader fr=new FileReader("db.properties");  Properties pr=new Properties();  pr.load(fr);  System.out.println(pr.getProperty("user"));  System.out.println(pr.getProperty("password"));  }  } |

**8. What do you understand by Iterator in the Java Collection Framework?**

Iterator in Java is an interface of the Collection framework present in java.util package. It is a Cursor in Java which is used to iterate a collection of objects. Below are a few other major functionalities provided by the Iterator interface:

* Traverse a collection object elements one by one
* Known as Universal Java Cursor as it is applicable for all the classes of the Collection framework
* Supports READ and REMOVE Operations.
* Iterator method names are easy to implement

**9. What is the need for overriding equals() method in Java?**

The initial implementation of the equals method helps in checking whether two objects are the same or not. But in case you want to compare the objects based on the property you will have to override this method.

**10. How the Collection objects are sorted in Java?**

Sorting in Java Collections is implemented via [Comparable](https://www.edureka.co/blog/comparable-in-java/) and [Comparator](https://www.edureka.co/blog/comparator-interface-java/) interfaces. When Collections.sort()  method is used the elements get sorted based on the natural order that is specified in the compareTo() method. On the other hand when Collections.sort(Comparator) method is used it sorts the objects based on compare() method of the Comparator interface. 

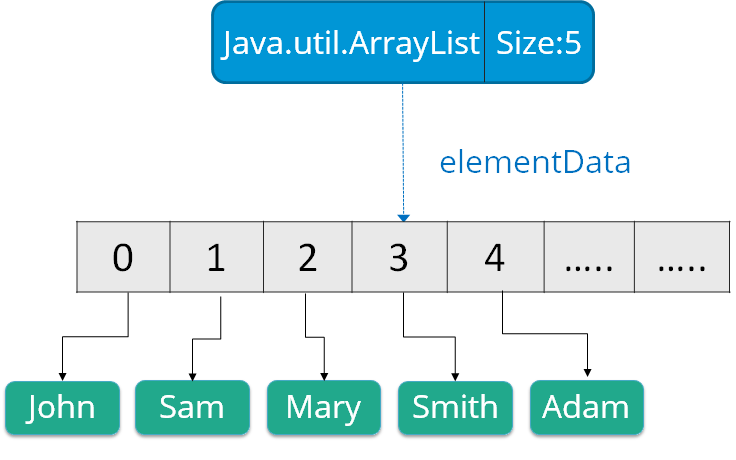
**List – Java Collections Interview Questions**

**11. What is the use of the List interface?**

The List interface in Java is an **ordered collection** of elements. It maintains the insertion order and allows duplicate values to be stored within. This interface contains various methods which enables smooth manipulation of elements based on the element index. The main classes implementing the List interface of the Collection framework are**ArrayList**, [**LinkedList**](https://www.edureka.co/blog/arraylist-vs-linkedlist/), **Stack, and Vector**.

**12. What is ArrayList in Java?**

ArrayList is the implementation of List Interface where the elements can be dynamically added or removed from the list. ArrayList in the Collection framework provides positional access and insertion of elements. It is an ordered collection that permits duplicate values. The size of an ArrayList can be increased dynamically if the number of elements is more than the initial size.

**Syntax**:

|  |  |
| --- | --- |
| 1 | ArrayList object = new ArrayList (); |

**13. How would you convert an ArrayList to Array and an Array to ArrayList?**

An Array can be converted into an ArrayList by making use of the asList() method provided by the Array class. It is a static method that accepts List objects as a parameter.

[[](https://www.edureka.co/java-j2ee-training-course)](https://www.edureka.co/java-j2ee-training-course" \t "_blank)

**[Java Certification Training Course](https://www.edureka.co/java-j2ee-training-course" \t "_blank)**

* *[Instructor-led Sessions](https://www.edureka.co/java-j2ee-training-course" \t "_blank)*
* *[Real-life Case Studies](https://www.edureka.co/java-j2ee-training-course" \t "_blank)*
* *[Assignments](https://www.edureka.co/java-j2ee-training-course" \t "_blank)*
* *[Lifetime Access](https://www.edureka.co/java-j2ee-training-course" \t "_blank)*

[Explore Curriculum](https://www.edureka.co/java-j2ee-training-course" \t "_blank)

**Syntax:**

|  |  |
| --- | --- |
| 1 | Arrays.asList(item) |

Whereas an ArrayList can be converted into an Array using the toArray() method of the ArrayList class.

**Syntax:**

|  |  |
| --- | --- |
| 1 | List\_object.toArray(new String[List\_object.size()]) |

**14. How will you reverse an List?**

ArrayList can be reversed using the reverse() method of the Collections class.

***Syntax:***

|  |  |
| --- | --- |
| 1 | public static void reverse(Collection c) |

***For Example:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | public class ReversingArrayList {  public static void main(String[] args) {  List<String> myList = new ArrayList<String>();  myList.add("AWS");  myList.add("Java");  myList.add("Python");  myList .add("Blockchain");  System.out.println("Before Reversing");  System.out.println(myList.toString());  Collections.reverse(myList);  System.out.println("After Reversing");  System.out.println(myList);  }  } |

**15. What do you understand by LinkedList in Java? How many types of LinkedList does Java support?**

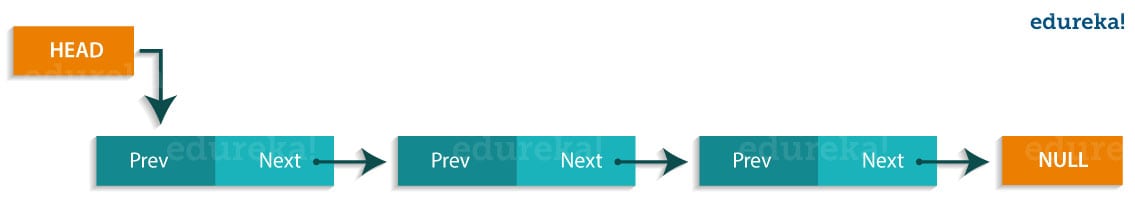
LinkedList in Java is a data structure that contains a sequence of links. Here each link contains a connection to the next link.

**Syntax:**

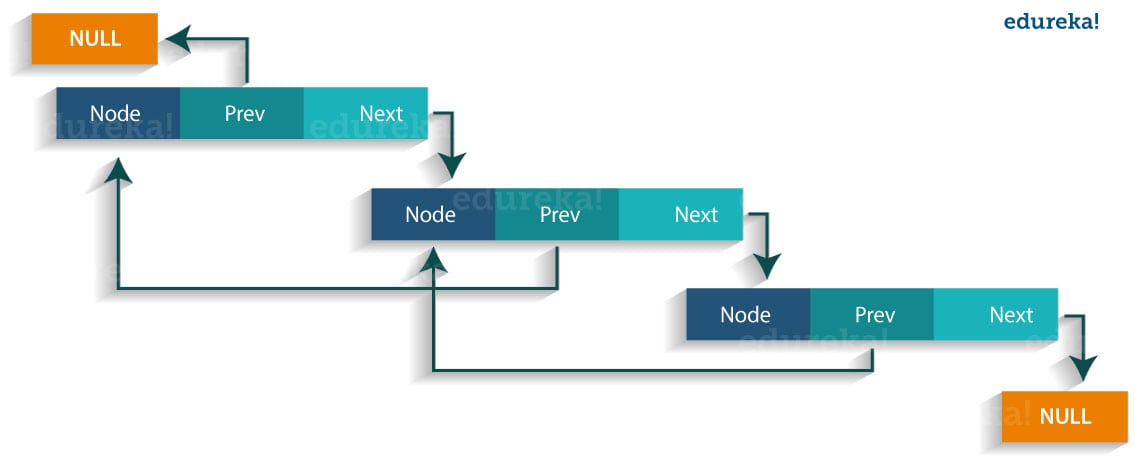
|  |  |
| --- | --- |
| 1 | Linkedlist object = new Linkedlist(); |

Java LinkedList class uses two types of LinkedList to store the elements:

* ***Singly Linked List:***In a singly LinkedList, each node in this list stores the data of the node and a pointer or reference to the next node in the list.



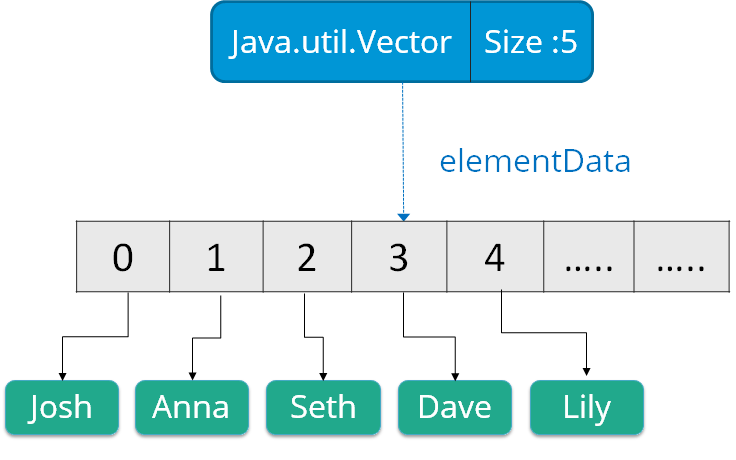
* ***Doubly Linked List:***In a doubly LinkedList, it has two references, one to the next node and another to the previous node.



**16. What is a Vector in Java?**

[Vectors](https://www.edureka.co/blog/vector-in-java/) are similar to arrays, where the elements of the vector object can be accessed via an index into the vector. Vector implements a dynamic array. Also, the vector is not limited to a specific size, it can shrink or grow automatically whenever required. It is similar to ArrayList, but with two differences :

* Vector is synchronized.
* Vector contains many legacy methods that are not part of the collections framework.

**Syntax**:

|  |  |
| --- | --- |
| 1 | Vector object = new Vector(size,increment); |

**Queue – Java Collections Interview Questions**

**17. What are the various methods provided by the Queue interface?**

Below are some of the methods of Java Queue interface:

|  |  |
| --- | --- |
| **Method** | **Description** |
| *boolean add(object)* | Inserts the specified element into the queue and returns true if it is a success. |
| *boolean offer(object)* | Inserts the specified element into this queue. |
| *Object remove()* | Retrieves and removes the head of the queue. |
| *Object poll()* | Retrieves and removes the head of the queue, or returns null if the queue is empty. |
| *Object element()* | Retrieves, but does not remove the head of the queue. |
| *Object peek()* | Retrieves, but does not remove the head of this queue, or returns null if the queue is empty. |

**18. What do you understand by BlockingQueue?**

BlockingQueue interface belongs to the **java.util.concurrent**[package](https://www.edureka.co/blog/packages-in-java/). This interface enhances flow control by activating blocking, in case a thread is trying to dequeue an empty queue or enqueue an already full queue. While working with the BlockingQueue interface in Java, you must remember that it does not accept a null value. In case you try to do that it will instantly throw a NullPointerException. The below figure represents the working of the BlockingQueue interface in Java.

**Programming & Frameworks Training**

**[JAVA CERTIFICATION TRAINING COURSE](https://www.edureka.co/java-j2ee-training-course" \o "Java Certification Training Course" \t "_blank)**

**[Java Certification Training Course](https://www.edureka.co/java-j2ee-training-course" \o "Java Certification Training Course" \t "_blank)**

*[Reviews](https://www.edureka.co/java-j2ee-training-course" \o "Java Certification Training Course" \t "_blank)*

**[4](https://www.edureka.co/java-j2ee-training-course" \o "Java Certification Training Course" \t "_blank)**[(46775)](https://www.edureka.co/java-j2ee-training-course" \o "Java Certification Training Course" \t "_blank)

**[PYTHON SCRIPTING CERTIFICATION TRAINING](https://www.edureka.co/python-scripting" \o "Python Scripting Certification Training" \t "_blank)**

**[Python Scripting Certification Training](https://www.edureka.co/python-scripting" \o "Python Scripting Certification Training" \t "_blank)**

*[Reviews](https://www.edureka.co/python-scripting" \o "Python Scripting Certification Training" \t "_blank)*

**[5](https://www.edureka.co/python-scripting" \o "Python Scripting Certification Training" \t "_blank)**[(10008)](https://www.edureka.co/python-scripting" \o "Python Scripting Certification Training" \t "_blank)

**[PYTHON DJANGO TRAINING AND CERTIFICATION](https://www.edureka.co/python-django" \o "Python Django Training and Certification" \t "_blank)**

**[Python Django Training and Certification](https://www.edureka.co/python-django" \o "Python Django Training and Certification" \t "_blank)**

*[Reviews](https://www.edureka.co/python-django" \o "Python Django Training and Certification" \t "_blank)*

**[5](https://www.edureka.co/python-django" \o "Python Django Training and Certification" \t "_blank)**[(5595)](https://www.edureka.co/python-django" \o "Python Django Training and Certification" \t "_blank)

**[NODE.JS CERTIFICATION TRAINING](https://www.edureka.co/nodejs-certification-training" \o "Node.js Certification Training" \t "_blank)**

**[Node.js Certification Training](https://www.edureka.co/nodejs-certification-training" \o "Node.js Certification Training" \t "_blank)**

*[Reviews](https://www.edureka.co/nodejs-certification-training" \o "Node.js Certification Training" \t "_blank)*

**[5](https://www.edureka.co/nodejs-certification-training" \o "Node.js Certification Training" \t "_blank)**[(6513)](https://www.edureka.co/nodejs-certification-training" \o "Node.js Certification Training" \t "_blank)

**[PHP & MYSQL WITH MVC FRAMEWORKS CERTIFICATION TRAINING](https://www.edureka.co/php-mysql-self-paced" \o "PHP & MySQL with MVC Frameworks Certification Training" \t "_blank)**

**[PHP & MySQL with MVC Frameworks Certification Training](https://www.edureka.co/php-mysql-self-paced" \o "PHP & MySQL with MVC Frameworks Certification Training" \t "_blank)**

*[Reviews](https://www.edureka.co/php-mysql-self-paced" \o "PHP & MySQL with MVC Frameworks Certification Training" \t "_blank)*

**[5](https://www.edureka.co/php-mysql-self-paced" \o "PHP & MySQL with MVC Frameworks Certification Training" \t "_blank)**[(3571)](https://www.edureka.co/php-mysql-self-paced" \o "PHP & MySQL with MVC Frameworks Certification Training" \t "_blank)

**[SPRING FRAMEWORK CERTIFICATION TRAINING](https://www.edureka.co/spring-certification-course" \o "Spring Framework Certification Training " \t "_blank)**

**[Spring Framework Certification Training](https://www.edureka.co/spring-certification-course" \o "Spring Framework Certification Training " \t "_blank)**

*[Reviews](https://www.edureka.co/spring-certification-course" \o "Spring Framework Certification Training " \t "_blank)*

**[5](https://www.edureka.co/spring-certification-course" \o "Spring Framework Certification Training " \t "_blank)**[(8621)](https://www.edureka.co/spring-certification-course" \o "Spring Framework Certification Training " \t "_blank)

**[MICROSOFT .NET FRAMEWORK CERTIFICATION TRAINING](https://www.edureka.co/microsoft-dotnet-framework-self-paced" \o "Microsoft .NET Framework Certification Training" \t "_blank)**

**[Microsoft .NET Framework Certification Training](https://www.edureka.co/microsoft-dotnet-framework-self-paced" \o "Microsoft .NET Framework Certification Training" \t "_blank)**

*[Reviews](https://www.edureka.co/microsoft-dotnet-framework-self-paced" \o "Microsoft .NET Framework Certification Training" \t "_blank)*

**[5](https://www.edureka.co/microsoft-dotnet-framework-self-paced" \o "Microsoft .NET Framework Certification Training" \t "_blank)**[(3818)](https://www.edureka.co/microsoft-dotnet-framework-self-paced" \o "Microsoft .NET Framework Certification Training" \t "_blank)

**[MASTERING PERL SCRIPTING CERTIFICATION TRAINING](https://www.edureka.co/mastering-perl-scripting-self-paced" \o "Mastering Perl Scripting Certification Training" \t "_blank)**

**[Mastering Perl Scripting Certification Training](https://www.edureka.co/mastering-perl-scripting-self-paced" \o "Mastering Perl Scripting Certification Training" \t "_blank)**

*[Reviews](https://www.edureka.co/mastering-perl-scripting-self-paced" \o "Mastering Perl Scripting Certification Training" \t "_blank)*

**[5](https://www.edureka.co/mastering-perl-scripting-self-paced" \o "Mastering Perl Scripting Certification Training" \t "_blank)**[(4358)](https://www.edureka.co/mastering-perl-scripting-self-paced" \o "Mastering Perl Scripting Certification Training" \t "_blank)

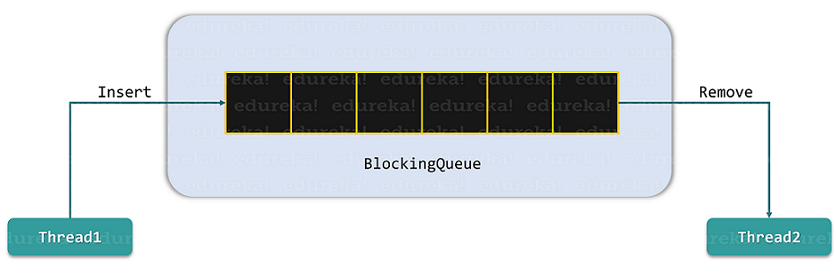
**[ADVANCED JAVA CERTIFICATION TRAINING](https://www.edureka.co/advanced-java-sp" \o "Advanced Java Certification Training" \t "_blank)**

**[Advanced Java Certification Training](https://www.edureka.co/advanced-java-sp" \o "Advanced Java Certification Training" \t "_blank)**

*[Reviews](https://www.edureka.co/advanced-java-sp" \o "Advanced Java Certification Training" \t "_blank)*

**[5](https://www.edureka.co/advanced-java-sp" \o "Advanced Java Certification Training" \t "_blank)**[(3823)](https://www.edureka.co/advanced-java-sp" \o "Advanced Java Certification Training" \t "_blank)

Next



**19. What is a priority queue in Java?**

A priority queue in Java is an abstract data type similar to a regular queue or stack data structure but has a special feature called priority associated with each element. In this queue, a high priority element is served before a low priority element irrespective of their insertion order. The PriorityQueue is based on the priority heap. The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.

**20. What is the Stack class in Java and what are the various methods provided by it?**

Java Stack class is an important part of  the Java Collection framework and is based on the basic principle of last-in-first-out. In other words, the elements are added as well as removed from the rear end. The action of adding an element to a stack is called push while removing an element is referred to as pop. Below are the various methods provided by this class:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| *empty()* | Checks if the stack is empty |
| *push()* | Pust an item to the top of the stack |
| *pop()* | Remove the object from the stack |
| *peek()* | Looks at the object of a stack without removing it |
| *search()* | Searches item in the stack to get its index |

**Set  – Java Collections Interview Questions**

**21. What is Set in Java Collections framework and list down its various implementations?**

A Set refers to a collection that cannot contain duplicate elements. It is mainly used to model the mathematical set abstraction. The Java platform provides three general-purpose Set implementations which are:

1. HashSet
2. TreeSet
3. LinkedHashSet

**22. What is the HashSet class in Java and how does it store elements?**

java.util.HashSet class is a member of the Java collections framework which inherits the AbstractSet class and implements the Set interface. It implicitly implements a hashtable for creating and storing a collection of unique elements. Hashtable is an instance of the HashMap class that uses a hashing mechanism for storing the information within a HashSet. Hashing is the process of converting the informational content into a unique value that is more popularly known as hash code. This hashcode is then used for indexing the data associated with the key. The entire process of transforming the informational key into the hashcode is performed internally.

**23. Can you add a null element into a TreeSet or HashSet?**

In HashSet, only one null element can be added but in TreeSet it can’t be added as it makes use of NavigableMap for storing the elements. This is because the NavigableMap is a subtype of SortedMap that doesn’t allow null keys. So, in case you try to add null elements to a TreeSet, it will throw a NullPointerException.

**24. Explain the emptySet() method in the Collections framework?**

The Collections.emptySet() is used to return the empty immutable Set while removing the null elements. The set returned by this method is serializable. Below is the method declaration of emptySet().

**Syntax:**

|  |  |
| --- | --- |
| 1 | public static final <T> Set<T> emptySet() |

**25. What is LinkedHashSet in Java Collections Framework?**

A java.util.LinkedHashSet is a subclass of the [HashSet](https://www.edureka.co/blog/hashset-in-java/)class and implements the Set interface. It is an ordered version of HashSet which maintains a doubly-linked List across all elements contained within. It preserves the insertion order and contains only unique elements like its parent class.

**Syntax:**

|  |  |
| --- | --- |
| 1 | LinkedHashSet<String> hs = new LinkedHashSet<String>(); |

**Map  – Java Collections Interview Questions**

**26. What is Map interface in Java?**

The java.util.Map interface in Java stores the elements in the form of keys-values pairs which is designed for faster lookups. Here every key is unique and maps to a single value. These key-value pairs are known as the map entries. This interface includes method signatures for insertion, removal, and retrieval of elements based on a key. With such methods, it’s a perfect tool to use for key-value association mapping such as dictionaries.

**27. Why Map doesn’t extend the Collection Interface?**

The Map interface in Java follows a key/value pair structure whereas the Collection interface is a collection of objects which are stored in a structured manner with a specified access mechanism. The main reason Map doesn’t extend the Collection interface is that the add(E e) method of the Collection interface doesn’t support the key-value pair like Map interface’s put(K, V) method. It might not extend the Collection interface but still is an integral part of the Java Collections framework.

**28. List down the different Collection views provided by the Map interface in the Java Collection framework?**

The Map interface provides 3 views of key-value pairs which are:

* key set view
* value set view
* entry set view

All these views can be easily navigated through using the iterators.

**29. What is the ConcurrentHashMap in Java and do you implement it?**

**ConcurrentHashMap** is a Java class that implements ConcurrentMap as well as Serializable interfaces. This class is the enhanced version of HashMap as it doesn’t perform well in the multithreaded environment. It has a higher performance rate compared to the HashMap.

Below is a small example demonstrating the implementation of ConcurrentHashMap:

[[](https://www.edureka.co/java-j2ee-training-course)](https://www.edureka.co/java-j2ee-training-course" \t "_blank)

**[Java Certification Training Course](https://www.edureka.co/java-j2ee-training-course" \t "_blank)**

[Weekday / Weekend BatchesSee Batch Details](https://www.edureka.co/java-j2ee-training-course" \t "_blank)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | package edureka;  import java.util.concurrent.\*;    public class ConcurrentHashMapDemo {      public static void main(String[] args)      {          ConcurrentHashMap m = new ConcurrentHashMap();          m.put(1, "Welcome");          m.put(2, "to");          m.put(3, "Edureka's");          m.put(4, "Demo");            System.out.println(m);            // Here we cant add Hello because 101 key          // is already present in ConcurrentHashMap object          m.putIfAbsent(3, "Online");          System.out.println("Checking if key 3 is already present in the ConcurrentHashMap object: "+ m);            // We can remove entry because 101 key          // is associated with For value          m.remove(1, "Welcome");          System.out.println("Removing the value of key 1: "+m);            // Now we can add Hello          m.putIfAbsent(1, "Hello");          System.out.println("Adding new value to the key 1: "+m);            // We cant replace Hello with For          m.replace(1, "Hello", "Welcome");          System.out.println("Replacing value of key 1 with Welcome: "+ m);      }  } |

**30. Can you use any class as a Map key?**

Yes, any class can be used as Map Key as long as the following points are considered:

* The class overriding the equals() method must also override the hashCode() method
* The class should adhere to the rules associated with equals() and hashCode() for all instances
* The class field which is not used in the equals() method should not be used in hashCode() method as well
* The best way to use a user-defined key class is by making it immutable. It helps in caching the hashCode() value for better performance. Also if the class is made immutable it will ensure that the hashCode() and equals() are not changing in the future.

**Differences – Java Collections Interview Questions**

**31. Differentiate between Collection and Collections.**

|  |  |
| --- | --- |
| **Collection** | **Collections** |
| java.util.Collection is an interface | java.util.Collections is a class |
| Is used to represent a group of objects as a single entity | It is used to define various utility method for collection objects |
| It is the root interface of the Collection framework | It is a utility class |
| It is used to derive the data structures of the Collection framework | It contains various static methods which help in data structure manipulation |

**32. Differentiate between an Array and an ArrayList.**

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
| java.util.Array is a class | java.util.ArrayList is a class |
| It is strongly typed | It is loosely types |
| Cannot be dynamically resized | Can be dynamically resized |
| No need to box and unbox the elements | Needs to box and unbox the elements |

**33. Differentiate between Iterable and Iterator.**

|  |  |
| --- | --- |
| **Iterable** | **Iterator** |
| Iterable is an interface | Iterator is an interface |
| Belongs to java.lang package | Belongs to java.util package |
| Provides one single abstract method called iterator() | Provides two abstract methods called hasNext() and next() |
| It is a representation of a series of elements that can be traversed | It represents the object with iteration state |

**34. Differentiate between ArrayList and LinkedList.**

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| Implements dynamic array internally to store elements | Implements doubly linked list internally to store elements |
| Manipulation of elements is slower | Manipulation of elements is faster |
| Can act only as a List | Can act as a List and a Queue |
| Effective for data storage and access | Effective for data manipulation |

**35. Differentiate between Comparable and Comparator.**

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| Present in java.lang package | Present in java.util package |
| Elements are sorted based on natural ordering | Elements are sorted based on user-customized ordering |
| Provides a single method called compareTo() | Provides to methods equals() and compare() |
| Modifies the actual class | Doesn’t modifies the actual class |

**36. Differentiate between List and Set.**

|  |  |
| --- | --- |
| **List** | **Set** |
| An ordered collection of elements | An unordered collection of elements |
| Preserves the insertion order | Doesn’t preserves the insertion order |
| Duplicate values are allowed | Duplicate values are not allowed |
| Any number of null values can be stored | Only one null values can be stored |
| ListIterator can be used to traverse the List in any direction | ListIterator cannot be used to traverse a Set |
| Contains a legacy class called vector | Doesn’t contains any legacy class |

**37. Differentiate between Set and Map.**

|  |  |
| --- | --- |
| **Set** | **Map** |
| Belongs to java.util package | Belongs to java.util package |
| Extends the Collection interface | Doesn’t extend the Collection interface |
| Duplicate values are not allowed | Duplicate keys are not allowed but duplicate values are |
| Only one null values can be stored | Only one null key can be stored but multiple null values are allowed |
| Doesn’t maintain any insertion order | Doesn’t maintain any insertion order |

**38. Differentiate between List and Map.**

|  |  |
| --- | --- |
| **List** | **Map** |
| Belongs to java.util package | Belongs to java.util package |
| Extends the Collection interface | Doesn’t extend the Collection interface |
| Duplicate elements are allowed | Duplicate keys are not allowed but duplicate values are |
| Multiple null values can be stored | Only one null key can be stored but multiple null values are allowed |
| Preserves the insertion order | Doesn’t maintain any insertion order |
| Stores elements based on Array Data Structure | Stores data in key-value pairs using various hashing techniques |

**39. Differentiate between Queue and Stack.**

|  |  |
| --- | --- |
| **Queue** | **Stack** |
| Based on FIFO (First-In-First-Out) principle | Based on LIFO (Last-In-First-Out) principle |
| Insertion and deletion takes place from two opposite ends | Insertion and deletion takes place the same end |
| Element insertion is called enqueue | Element insertion is called push |
| Element deletion is called dequeue | Element deletion is called pop |
| Two pointers are maintained one point to the first element and the other one points the last element on the list | Only one pointer is maintained which points to the top element on the stack |

**40. Differentiate between PriorityQueue and TreeSet.**

|  |  |
| --- | --- |
| **PriorityQueue** | **TreeSet** |
| It is a type of Queue | It is based on a Set data structure |
| Allows duplicate elements | Doesn’t allows duplicate elements |
| Stores the elements based on an additional factor called priority | Stores the elements in a sorted order |

**41. Differentiate between the Singly Linked List and Doubly Linked List.**

|  |  |
| --- | --- |
| **Singly Linked List(SLL)** | **Doubly Linked List(DLL)** |
| Contains nodes with a data field and a next node-link field | Contains nodes with a data field, a previous link field, and a next link field |
| Can be traversed using the next node-link field only | Can be traversed using the previous node-link or the next node-link |
| Occupies less memory space | Occupies more memory space |
| Less efficient in providing access to the elements | More efficient in providing access to the elements |

**42. Differentiate between Iterator and Enumeration.**

|  |  |
| --- | --- |
| **Iterator** | **Enumeration** |
| Collection element can be removed while traversing it | Can only traverse through the Collection |
| Used to traverse most of the classes of the Java Collection framework | Used to traverse the legacy classes such as Vector, HashTable, etc |
| Is fail-fast in nature | Is fail-safe in nature |
| Is safe and secure | Is not safe and secure |
| Provides methods like hasNext(), next() and remove() | Provides methods like hasMoreElements() and nextElement() |

**43. Differentiate between HashMap and HashTable.**

|  |  |
| --- | --- |
| **HashMap** | **HashTable** |
| It is non-synchronized in nature | It is synchronized in nature |
| Allows only one null key but multiple null values | Doesn’t allow any null key or value |
| Has faster processing | has slower processing |
| Can be traversed by Iterator | Can be traversed by Iterator and Enumeration |
| Inherits AbstractMap class | Inherits Dictionary class |

**44. Differentiate between HashSet and HashMap.**

|  |  |
| --- | --- |
| **HashSet** | **HasMap** |
| Based on Set implementation | Based on Map implementation |
| Doesn’t allow any duplicate elements | Doesn’t allow any duplicate keys but duplicate values are allowed |
| Allows only a single null value | Allows only one null key but any number of null values |
| Has slower processing time | Has faster processing time |
| Uses HashMap as an underlying data structure | Uses various hashing techniques for data manipulation |

**45. Differentiate between Iterator and ListIterator.**

|  |  |
| --- | --- |
| **Iterator** | **ListIterator** |
| Can only perform remove operations on the Collection elements | Can perform add, remove and replace operations the Collection elements |
| Can traverse List, Sets and maps | Can traverse only Lists |
| Can traverse the Collection in forward direction | Can traverse the collection in any direction |
| Provides no method to retrieve the index of the element | Provides methods to retrieve the index of the elements |
| iterator() method is available for the entire Collection Framework | listIterator() is only available for the collections implementing the List interface |

**46. Differentiate between HashSet and TreeSet.**

|  |  |
| --- | --- |
| **HashSet** | **TreeSet** |
| Uses HasMap to store elements | Uses Treemap to store elements |
| It is unordered in nature | By default, it stores elements in their natural ordering |
| Has faster processing time | Has slower processing time |
| Uses hasCode() and equals() for comparing | Uses compare() and compareTo() for comparing |
| Allows only one null element | Doesn’t allow any null element |
| Takes up less memory space | Takes up more memory space |

**47. Differentiate between Queue and Deque.**

|  |  |
| --- | --- |
| **Queue** | **Deque** |
| Refers to single-ended queue | Refers to double-ended queue |
| Elements can be added or removed from only one end | Elements can be added and removed from either end |
| Less versatile | More versatile |

**48. Differentiate between HashMap and TreeMap.**

|  |  |
| --- | --- |
| **HashMap** | **TreeMap** |
| Doesn’t preserves any ordering | Preserves the natural ordering |
| Implicitly implements the hashing principle | Implicitly implements the Red-Black Tree Implementation |
| Can store only one null key | Cannot store any null key |
| More memory usage | Less memory usage |
| Not synchronized | Not synchronized |

**49. Differentiate between ArrayList and Vector.**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| Non-synchronized in nature | Synchronized in nature |
| It is not a legacy class | Is a legacy class |
| Increases size by 1/2 of the ArrayList | Increases size by double of the ArrayList |
| It is not thread-safe | It is thread-safe |

**50. Differentiate between failfast and failsafe.**

|  |  |
| --- | --- |
| **failfast** | **failsafe** |
| Doesn’t allow modifications of a collection while iterating | Allows modifications of a collection while iterating |
| Throws ConcurrentModificationException | Don’t throw any exceptions |
| Uses the original collection to traverse over the elements | Uses a copy of the original collection to traverse over the elements |
| Don’t require extra memory | Require extra memory |

So this brings us to the end of the Java Collections interview questions. The topics that you learned in this Java Collections Interview Questions are the most sought-after skill sets that recruiters look for in a Java Professional. These sets of Java Collection Interview Questions will definitely help you ace your job interview. **Good luck with your interview!**

**Q #1) Explain the Java Collections Framework.**

**Answer:** The Java Collections Framework is an architecture that helps in managing and storing a group of objects. With it, the developers can access prepackaged data structures and manipulate data with the use of algorithms as well.

Java collection includes the interface, and classes, that support operations like searching, deleting, insertion, sorting, etc. Along with interface and classes, Java Collections include algorithms as well that helps in manipulations.

**Q #2) What are the benefits of Java Collections?**

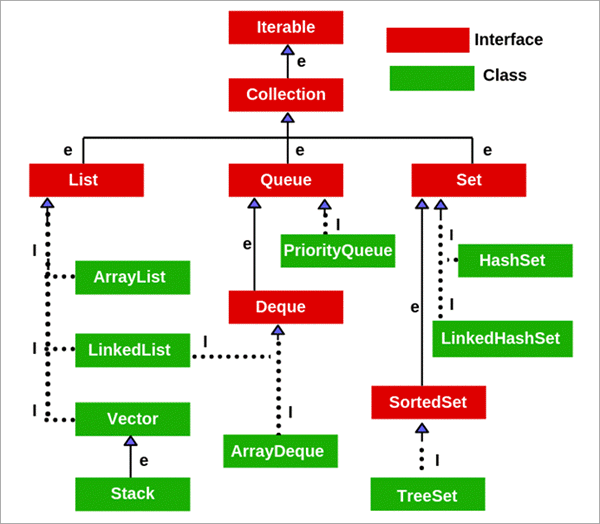
**Answer:**

**The benefits of Java Collections are:**

* Instead of implementing our collection classes, it uses the core collection classes, thereby reducing the effort required for its development.
* It uses the collection framework classes that are well tested. Hence, its code quality is enhanced.
* It reduces the effort in code maintenance.
* Java Collection Framework is interoperable and reusable.

**Q #3) What do you know about the Hierarchy of Collections in Java?**

**Answer:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/05/java-collection-hierarchy.png)

*[image*[*source*](https://1.bp.blogspot.com/)*]*

**Q #4) Tell us about the Java Collection’s basic interfaces.**

**Answer:**

**Enlisted below are the basic interfaces of the Java Collection Framework.**

**Collection:** It is the base of the Collection hierarchy and represents its elements. However, Java doesn’t provide any direct application of Collection. Also, most of the Java collections come from this interface.

**Set:** It cannot hold duplicate elements as it models the abstraction of the mathematical set. As the name suggests, it represents sets, **for example,** a deck of cards.

**List:** It can hold duplicate elements and is an ordered collection. You can use its index to access any element in it. The list is like an arrangement, an array, with a dynamic length.

These are a few interfaces of Java Collection. However, there are a few more interfaces as well as shown below.

* Queue
* Dequeue
* Iterator
* Iterable
* SortedSet
* ListIterator.

**Q #5) Why Serializable and Cloneable Interfaces are not extended by the Collection Interface?**

**Answer:**The task of the Collection Interface is to specify a group of objects called elements. The implementation of the Collections decides how the elements will be maintained. **For example,** the List implementations allow duplicate elements but Set implementations don’t.

Many implementations have a method for public cloning. But it isn’t practical to include it in all Collection implementations as the Collection is abstract and implementation is all that matters.

The meaning and the ramifications of both serialization and cloning makes sense while working with the substantive implementations. Hence, it is up to the actual implementation to decide if it can be serialized or cloned and how.

***Learn more =>>  [Marker Interface In Java: Serializable And Cloneable](https://www.softwaretestinghelp.com/marker-interfaces-java/)***

That’s why inducting serialization and cloning in every implementation isn’t very flexible and is restrictive.

**Q #6) What do you understand by Iterator in the Java Collection Framework?**

**Answer:** In simples arrays, we can use loops to access each element. When a similar approach is needed to access elements in a collection, we go for iterators. Iterator is a construct used to access elements of Collection objects.

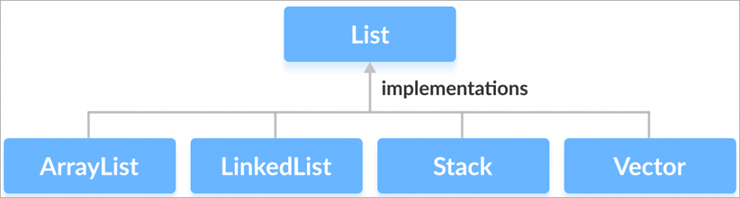
In Java, Iterators are the objects that implement the “Iterator” interface of Collection Framework. This interface is a part of the java.util package.

**Some of the characteristics of Iterators are:**

* Iterators are used to traverse the Collection objects.
* Iterators are known as “Universal Java Cursor” as we can use the same Iterator for all the collections.
* Iterators provide “Read” and “Remove” operations apart from traversing the collections.
* As they are universal and work with all the collections, Iterators are easier to implement.

List Java Collection Questions

**Q #7) Are you aware of the uses of the List Interface?**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/05/Java-list-interface.png)

*[image*[*source*](https://cdn.programiz.com/)*]*

**Answer:**The purpose of the List Interface is to maintain the order of the insertion. It also allows the storage of duplicate values.

It helps in the smooth manipulation of the elements depending on its index with the various methods it contains. ArrayList, Vector, Stack, and LinkedList are the various classes that implement the List Interface.

**Q #8) What do you understand about ArrayList in Java?**

**Answer:**The implementation of the List Interface is ArrayList. It dynamically adds or removes elements from the list and it also provides insertion of elements along with the positional access. ArrayList permits duplicate values and its size can dynamically increase if the number of the elements exceeds the initial size.

**Q #9) How will you convert a string array to an ArrayList?**

**Answer:**This is a beginner’s level programming question that an interviewer asks to check your grasp of Collection utility.classes. Collection and Arrays are the two utility classes of the Collection Framework that interviewers are often interested in.

Collections offer certain static functions for performing specific tasks on collection types. While Array has utility functions that it performs on array types.

|  |
| --- |
| //String array  String[] num\_words = {"one", "two", "three", "four", "five"};  //Use java.util.Arrays class to convert to list  List wordList = Arrays.asList(num\_words); |

**Note that apart from String type, you can also use other type Arrays to convert to ArrayList.**

**For example,**

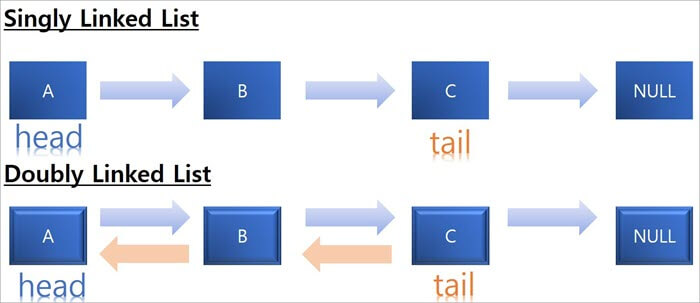
|  |
| --- |
| //Integer array  Integer[] numArray = {10,20,30,40};  //Convert to list using Arrays class asList method  List num\_List = Arrays.asList(numArray); |

**Q #10) Convert Array to ArrayList and ArrayList to Array.**

**Answer:**To convert ArrayList to Array, the toArray() method is used- *List\_object.toArray(new String[List\_object.size()])*

While the asList() method is used for converting Array to ArrayList- *Arrays.asList(item).*The asList() is a static method where List objects are the parameters.

**Q #11) What is a LinkedList and how many types of it are supported in Java?**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/05/singly-linked-list-vs-doubly-linked-list-1.jpg)

*[image*[*source*](https://personalzone-hulgokm2zfcmm9u.netdna-ssl.com/wp-content/uploads/2017/07/singly-linked-list-vs-doubly-linked-list.jpg)*]*

**Answer:**LinkedList is a data structure with a sequence of links where every link is connected to the next link.

**Two types of LinkedList are used in Java for storing the elements:**

1. **Singly LinkedList:** Here, each node stores the data of the node along with a reference or the pointer to the next node.
2. **Doubly LinkedList:**A doubly LinkedList comes with dual references, one reference to the next node, and another one for the previous node.

**Q #12) What do you understand by BlockingQueue?**

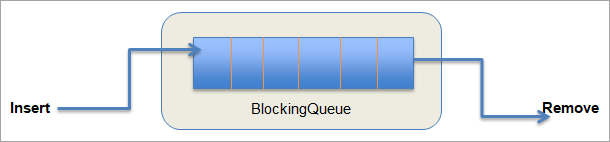
**Answer:**In a simple queue, we know that whenever the queue is full, we cannot insert any more items. In this case, the queue simply provides a message that the queue is full and exits. A similar case happens when the queue is empty and there is no element to be removed in the queue.

Instead of just exiting when insert/remove cannot be done, how about we wait till we can insert or remove the item?

This is answered by a variation of queue called **“Blocking queue”**. In blocking queue, blocking is activated during enqueue and dequeue operations whenever the queue is trying to enqueue full queue or dequeue an empty queue.

**The blocking is shown in the following figure.**

**BlockingQueue**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/05/2020-05-26_13-46-13.png)

Thus, during enqueue operation, the blocking queue will wait till a space becomes available so that an item can be successfully inserted. Similarly, in the dequeue operation blocking queue will wait until an item become available for the operation.

Blocking queue implements ‘BlockingQueue’ interface that belongs to ‘java.util.concurrent’ package. We should remember that the BlockingQueue interface does not allow null value. If it encounters null, then it throws NullPointerException.

**Q #13) What is a Priority Queue in Java?**

**Answer:**A priority queue in Java is similar to stack or queue data structures. It is an abstract data type in Java and is implemented as a PriorityQueue class in java.util package. The priority queue has a special feature that each item in the Priority queue has a priority.

In a priority queue, an item with higher priority is the server before the item with lower priority.

All the items in the priority queue are ordered as per natural ordering. We can also order the elements according to custom order by providing a comparator at the time of creating a priority queue object.

Set Interface Interview Questions

**Q #14) What is the use of Set Interface? Tell us about the classes implementing this Interface.**

**Answer:**Set Interface is used in the set theory to shape the mathematical set. It is similar to the List interface and yet is a little different from it. Set Interface isn’t an ordered collection hence, there is no preserved ordering when you are removing or adding the elements.

Mainly, it doesn’t support duplicate elements thus each element in the Set Interface is unique.

It also allows meaningful comparisons of Set instances even when there are different implementations. Also, it puts in a more substantial contract on the actions of the operations of equals and hashCode. If two examples have the same elements, then they are equal.

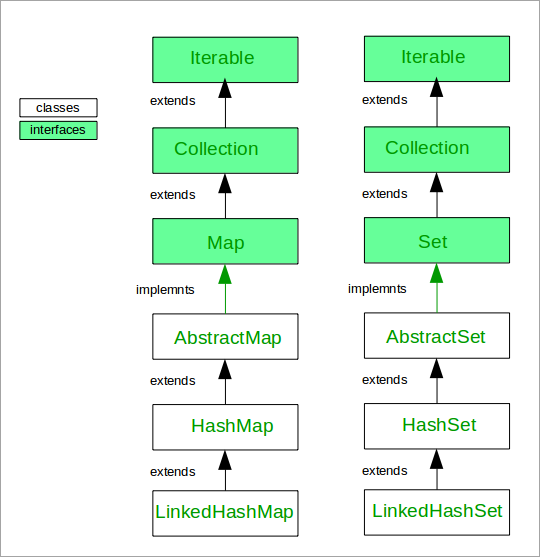
For all these reasons, Set Interface doesn’t have element index-based operations like List. It only uses Collection Interface inherited methods. TreeSet, EnumSet, LinkedHashSet, and HashSet implements Set Interface.

**Q #15) I want to add a null element to HashSet and TreeSet. Can I?**

**Answer:**You can’t add any null element in TreeSet as it uses NavigableMap for element storage. But you can add just one to HashSet. SortedMap doesn’t allow null keys and NavigableMap is its subset.

That’s why you can’t add a null element to TreeSet, it will come up with the NullPointerException every time you try to do that.

**Q #16) What do you know about LinkedHashSet?**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/05/LinkedHashSet.png)

**Answer:**LinkedHashSet is the subclass of HashSet and it enforces the Set Interface. As an ordered form of HashSet, it manages a doubly-linked List throughout all the elements it contains. It retains the order of insertion and just like its parent class, it only carries unique elements.

**Q #17) Talk about the way HashSet stores elements.**

**Answer:**HashMap stores the pairs of key-values but the keys should be unique. This feature of Map is used by HashSet to make sure every element is unique.

**The Map declaration in HashSet appears as shown below:**

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

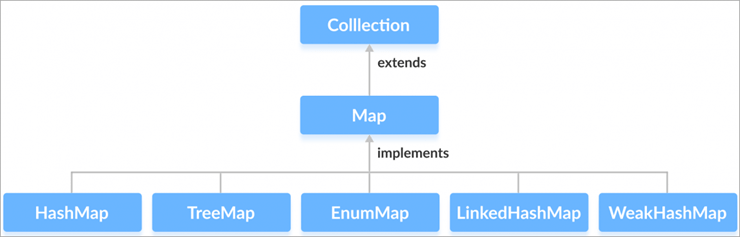
The stored elements in HashSet are stored as a key in the Map and the object is presented as a value.

**Q #18) Explain the EmptySet() method.**

**Answer:**The Emptyset() method removes the null elements and returns the empty unchangeable set. This immutable set is serializable. The method declaration of the Emptyset() is- *public static final <T> Set<T> emptySet().*

Map Interface Interview Questions

**Q #19) Tell us about the Map Interface.**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/05/java-map-implementation.png)

**Answer:**Map Interface is designed for faster lookups and it stores the elements in the form of pairs of key-values. As every key is unique here, it connects or maps to a single value only. These pairs of key-values are called map entries.

In this interface, there are method signatures for retrieval, insertion, and removal of elements depending on the unique key. This makes it a perfect tool for mapping key-value associations, like a dictionary.

**Q #20) The map doesn’t extend the Collection Interface. Why?**

**Answer:**The collection Interface is the accumulation of objects and these objects are stored structurally with the mechanism of specified access. While the Map interface follows the structure of key-value pairs. The add method of the Collection Interface doesn’t support the put method of Map Interface.

That’s why Map doesn’t extend the Collection Interface but still, it is an important part of the Java Collection Framework.

**Q #21) How does HashMap work in Java?**

**Answer:**HashMap is a collection based on Map and its items consist of key-value pairs. A HashMap is typically denoted by <Key, Value>, or <K, V>. Each hashmap element can be accessed using its key.

A HashMap works on the principle of “Hashing”. In the hashing technique, a longer string is transformed into a smaller string by a ‘hash function’ which is nothing but an algorithm. The smaller string aids in faster searching and efficient indexing.

**Q #22) Explain IdentityHashMap, WeakHashMap, And ConcurrentHashMap.**

**Answer:**

**IdentityHashMap** is much like HashMap. The difference is that while comparing elements, IdentityHashMap uses reference equality. It is not a preferred Map Implementation and although it executes the Map Interface, it fails to comply with the general contract of the Map intentionally.

So, when comparing objects, this authorizes the use of the equals method. It is designed for usage in rare cases where one needs reference-equality semantics.

**WeakHashMap** Implementation stores only weak references to its keys. This allows the garbage collection of a key-value pair when there is no more reference of its keys outside the WeakHashMap.

It is primarily used with those key objects where the test for object identity is carried out by its equals methods using the == operator.

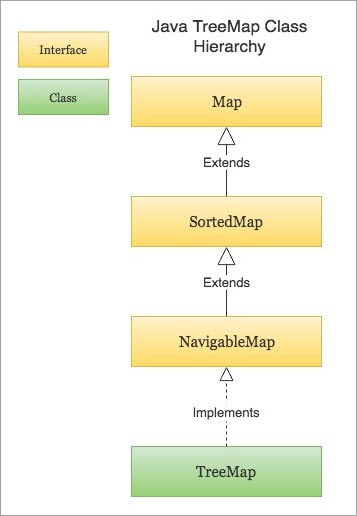
**ConcurrentHashMap** implements both ConcurrentMap and Serializable interfaces. It is the upgraded, enhanced version of HashMap as it doesn’t work well with the multithreaded environment. When compared to the HashMap, it has a higher performance rate.

**Q #23) What is the quality of a good key for HashMap?**

**Answer:**Understanding how HashMap works, it is easy to know that they depend mainly on equals and hashCode methods of key objects. So, a good key must provide the same hashCode over and over again irrespective of the times it is fetched.

In the same way, when compared with the equals method, the same keys must return true and different keys must return false. That’s why the best candidate for HashMap keys is said to be immutable classes.

**Q #24) When can you use TreeMap?**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/05/java-treemap-hierarchy-1.jpg)

*[image*[*source*](https://www.callicoder.com/)*]*

**Answer:**TreeMap, as a special form of HashMap, maintains the ordering of the keys by default ‘natural ordering’, as something that is missing in HashMap. You can use it for sorting objects with some key.

**For example,** if you want to implement and print a dictionary in alphabetical order, you can use TreeMap together with TreeSet. It will sort automatically. Of course, you could have done that manually as well but the work will be done more efficiently with the use of TreeMap. You can also use it if random access is vital for you.

Difference Between Questions

**Q #25) What is the difference between Collection and collections?**

**Answer:**

| **Collection** | **Collections** |
| --- | --- |
| It is an interface. | It is class. |
| The collection represents a group of objects as a single entity. | Collections define different methods of utility for collection objects. |
| It is Collection Framework’s root interface. | Collections are a utility class. |
| It derives Collection Framework’s data structures. | Collections contain many different static methods for aiding in manipulating the data structure. |

**Q #26) How is Array different from an ArrayList?**

**Answer:**

**Differences between Array and ArrayList are given below:**

| **Array** | **ArrayList** |
| --- | --- |
| The array is a strongly typed class. | ArrayList is a loosely typed class. |
| Array can’t be resized dynamically, its dimension is static. | ArrayList can be resized dynamically. |
| An array doesn’t need boxing and unboxing of elements. | ArrayList needs boxing and unboxing of elements. |

**Q #27) Differentiate between ArrayList and LinkedList.**

**Answer:**

| **ArrayList** | **LinkedList** |
| --- | --- |
| ArrayList uses the dynamic array internally for storing elements. | LinkedList implements the doubly linked list. |
| ArrayList manipulation of elements is rather slow. | LinkedList manipulates its elements much faster. |
| ArrayList can act solely as a List. | LinkedList can act as both List and a Queue. |
| Useful for storing and accessing data. | Useful for manipulating data. |

**Q #28) How is Iterable different from Iterator?**

**Answer:**

| **Iterable** | **Iterator** |
| --- | --- |
| It is Java.lang package interface. | It is Java.util package interface. |
| Yields only one abstract method known as the Iterator. | It comes with two abstract methods- hasNext and next. |
| Represents a series of elements that can be traversed. | Stands for objects with iteration state. |

**Q #29) State the differences between Set and List.**

**Answer:**

| **Set** | **List** |
| --- | --- |
| Set implements Set interface. | The list implements the List interface. |
| Set is an unordered set of elements. | The list is an ordered set of elements. |
| The set doesn’t maintain the order of elements during insertion. | List retains the order of elements during insertion. |
| The set doesn’t allow duplicate values. | The list allows duplicate values. |
| The set does not contain any legacy class. | List contains Vector, a legacy class. |
| The set allows only one null value. | No restriction on the number of null values in List. |
| We cannot use ListIterator to traverse a set. | ListIterator can traverse List in any direction. |

**Q #30) What is the difference between Queue and Stack?**

**Answer:**

| **Queue** | **Stack** |
| --- | --- |
| Queue works on the principle of the First-In-First-Out (FIFO) approach. | Stack works on a Last-In-First-Out (LIFO) basis. |
| Insertion and deletion in the queue take place at different ends. | Insertion and deletion are performed from the same end called the top of the stack. |
| Enqueue is the name of Insertion and dequeue is the deletion of elements. | Push is insertion and Pop is the deletion of elements in Stack. |
| It has two pointers- one to the first element of the list (front) and one to the last (rear). | It only has one pointer pointing to the top element. |

**Q #31) How are SinglyLinkedList and DoublyLinkedList different from each other?**

**Answer:**

| **Singly Linked List** | **Doubly Linked List** |
| --- | --- |
| Each node of the singly linked list consists of a data and a pointer to the next node. | A doubly linked list consists of data, a pointer to the next node, and a pointer to the previous node. |
| The singly-linked list can be traversed using the next pointer. | A doubly linked list can be traversed using both previous and next pointer. |
| The singly-linked list takes less space compared to a doubly-linked list. | The doubly linked list takes up a lot of memory space. |
| Element access is not very efficient. | Element access is efficient. |

**Q #32) How is HashMap different from HashTable?**

**Answer:**

| **HashMap** | **HashTable** |
| --- | --- |
| HashMap inherits AbstractMap class | HashTable inherits Dictionary class. |
| HashMap is not synchronized. | HashTable is synchronized. |
| HashMap allows multiple null values but only one null key. | HashTable does not allow a null value or key. |
| HashMap is faster. | HashTable is slower than HashMap. |
| HashMap can be traversed by Iterator. | HashTable cannot be traversed using iterator or enumerator. |

**Q #33) List down the difference between ArrayList and Vector.**

**Answer:**

| **ArrayList** | **Vector** |
| --- | --- |
| ArrayList is non-synchronized. | Vector is synchronized. |
| ArrayList is not a legacy class. | Vector is a legacy class. |
| ArrayList increases size by half of ArrayList when an element is inserted beyond its size. | Vector increases its size by double when an element is inserted beyond its size. |
| ArrayList is not thread-safe | Vector is a thread-safe. |

**Q #34) How is FailFast different from Failsafe?**

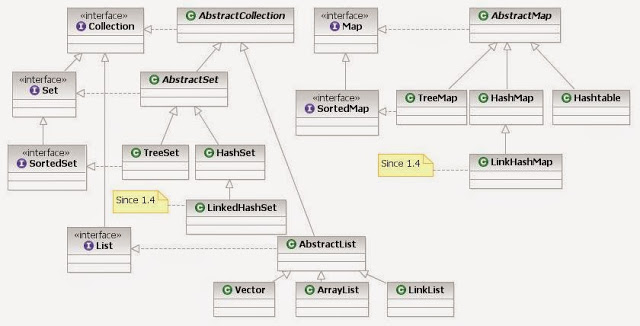
**Answer:**

| **FailFast** | **FailSafe** |
| --- | --- |
| While iterating, no modification of a collection is allowed. | Allows modification while iterating. |
| Uses original collection for traversing. | Uses a copy of the original collection. |
| No extra memory required. | Needs extra memory. |
| Throws ConcurrentModificationException. | No exception is thrown. |

**Q1  What is Collection? What is a Collections Framework? What are the benefits of the Java Collections Framework?**  
  
**Collection :** A collection (also called a container) is an object that groups multiple elements into a single unit.  
  
**Collections Framework :**Collections framework provides a 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 the collection hierarchy?**  
  
The root interface in the collection hierarchy is the**Collection interface.**Few interviewers may argue that  
the Collection interface extends the**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](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html), that Collection interface is a member of the Java Collections framework.  For the [Iterable interface Oracle doc,](https://docs.oracle.com/javase/7/docs/api/java/lang/Iterable.html) the 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?**  
  
The Collection is an interface while Collections is a java class, both are present in java.util package and part of the java collections framework. ([answer](https://javahungry.blogspot.com/2019/03/difference-between-collection-and-collections-in-java.html))

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

**Q5 Name the core Collection interfaces?**

[](https://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 interfaces also on the list :  SortedSet, SortedMap, Deque, ListIterator, etc.  
 **Q6 What is the difference between List and Set?**  
  
Set contains only unique elements while List can contain duplicate elements.  
Set is unordered while the 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 contains 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?**  
  
The 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 the iterator design pattern. An 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 an Iterator design pattern.  
 **Q12 Which methods you need to override to use any object as a key in HashMap?**  
  
To use any object as a key in HashMap, it needs to implement equals() and hashCode() method.  
  
**Q13  What is the difference between Queue and Stack?**  
  
The Queue is a data structure that is based on FIFO ( first in first out ) property. An example of a Queue in the real-world is buying movie tickets in the multiplex or cinema theaters.  
  
The Stack is a data structure that is based on LIFO (last in first out) property. An example of Stack in the real-world is the 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 the Collections class. reverse(List list) accepts the list as a 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 a 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 questions, 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 Array size by 50%. find detailed explanation   [ArrayList vs Vector.](https://javahungry.blogspot.com/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 questions for java developers. 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 a detailed explanation here [Hashtable vs HashMap in Java](https://javahungry.blogspot.com/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 are used to remove the head object of the Queue. The main difference lies when the Queue is empty().  
If the Queue is empty then the 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 the queue is empty then the peek() method also returns null.

**Q19 What is the difference between Iterator and ListIterator.**  
  
Using Iterator we can traverse the list of objects in the forward direction. But ListIterator can traverse the collection in both directions that are forward as well as backward.  
  
**Q20 What is the difference between Array and ArrayList in Java?**  
  
This question checks whether the student understands the concept of the 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](https://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 the null object while TreeSet can not store the null object.  
find a detailed explanation here [TreeSet vs HashSet in Java](https://javahungry.blogspot.com/2014/03/difference-between-hashset-and-treeset-similarities-and-example.html)