InterviewBit

Java Collections Interview Questions

To view the live version of the page, [click here.](#)

Contents

Java Collections Interview Questions for Experienced

(.....Continued)

19. Differentiate between Comparable and Comparator in the context of Java.
20. What do you understand about BlockingQueue in Java?
21. Explain fail-fast and fail-safe iterators. Differentiate between them.
22. What is the purpose of RandomAccess Interface? Name a collection type which implements this interface.
23. Differentiate between Iterator and Enumeration.
24. What is the use of Properties class in Java? What is the advantage of the Properties file?
25. Differentiate between HashMap and HashTable.
26. Why does HashMap allow null whereas HashTable does not allow null?
27. How can you synchronize an ArrayList in Java?
28. Why do we need synchronized ArrayList when we have Vectors (which are synchronized) in Java?
29. Why does not the Map interface extend the Collection Interface or vice-versa?
30. Differentiate between HashMap and TreeMap in the context of Java.

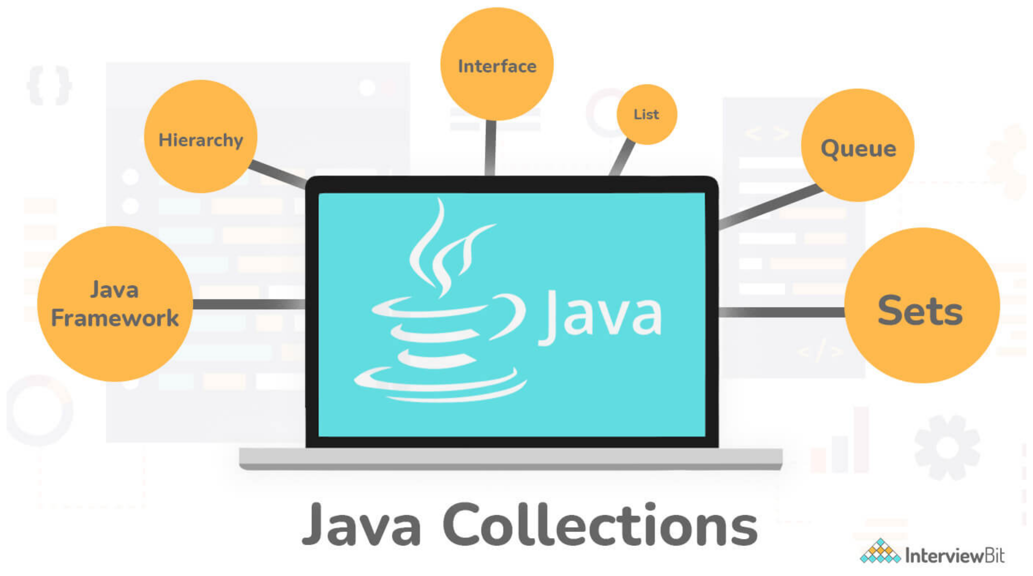
Java Collection Programs

31. Given an array in Java, convert it to a collection.
32. Write a program in Java to display the contents of a HashTable using enumeration.
33. Write a program to shuffle all the elements of a collection in Java.
34. Write a program in Java to clone a Treeset to another Treeset.
35. Write a program in java to get the collection view of the values present in a HashMap.
36. Write a program in java to join two arraylists into one arraylist.

Let's get Started

What is Collection in Java?

In Java, a collection is a framework that provides an architecture for storing and manipulating a collection of objects. In JDK 1.2, a new framework called "Collection Framework" was created, which contains all of the collection classes and interfaces. Collections in Java are capable of doing any data operations such as searching, sorting, insertion, manipulation, and deletion.
A single unit of objects in Java is referred to as a collection. The two basic "root" interfaces of Java collection classes are the Collection interface (java.util.Collection) and the Map interface(java.util.Map). Many interfaces (Set, List, Queue, Deque) and classes are available in the Java Collection framework (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).



Need for the Collection framework:-
Prior to the introduction of Collection Framework (or JDK 1.2), the standard techniques for aggregating Java objects (or collections) were Arrays, Vectors, or Hash Tables. There was no common interface for all of these collections. As a result, while the basic goal of all the collections is the same, their implementation was specified independently and there was no correlation between them. Furthermore, users find it challenging to remember all of the various methods, syntax, and constructors included in each collection class.

Java Collections Interview Questions for Freshers

1. What is the difference between Array and Collection in java?

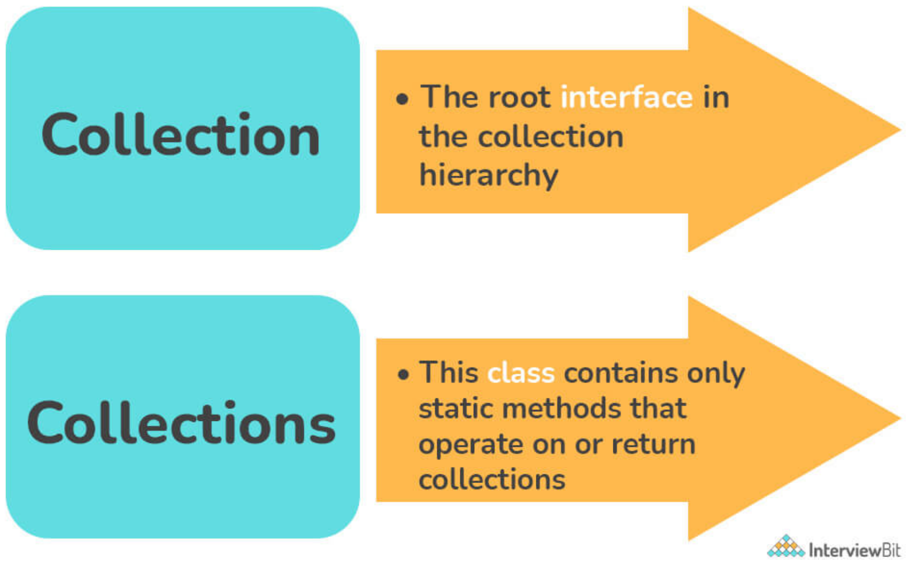
Array and Collection are equivalent in terms of storing object references and manipulating data, but they differ in a number of ways. The following are the primary distinctions between an array and a Collection:

Array	Collection
Arrays have a set size, which means that once we build one, we can't change it to meet our needs.	Collection are naturally grow-able and can be customized to meet our needs. We can change its size as per our requirement.
When it comes to performance, Arrays are the preferred to Collection.	Considering performance, Collection are not preferred to Arrays.
Only homogeneous data type elements can be stored in arrays.	Both homogeneous and heterogeneous components can be stored in a collection.
Because arrays have no underlying data structure, there is no ready-made method support.	Any collection class is built on a standard data structure, and so there is ready-made method support for every demand as a performance. These methods can be used directly, and we are not responsible for their implementation.
Objects and primitives can	Only object types can be stored in a

2. Differentiate between Collection and collections in the context of Java.

Collection : In the java.util.package, there is an interface called a collection. It's used to represent a collection of separate objects as a single entity. It's equivalent to the container in the C++ programming language. The collection framework's root interface is referred to as the collection. It has a number of classes and interfaces for representing a collection of individual objects as a single unit. The key sub-interfaces of the collection interface are List, Set, and Queue. Although the map interface is part of the Java collection framework, it does not inherit the interface's collection. The Collection interface's most significant functions are add(), remove(), clear(), size(), and contains().

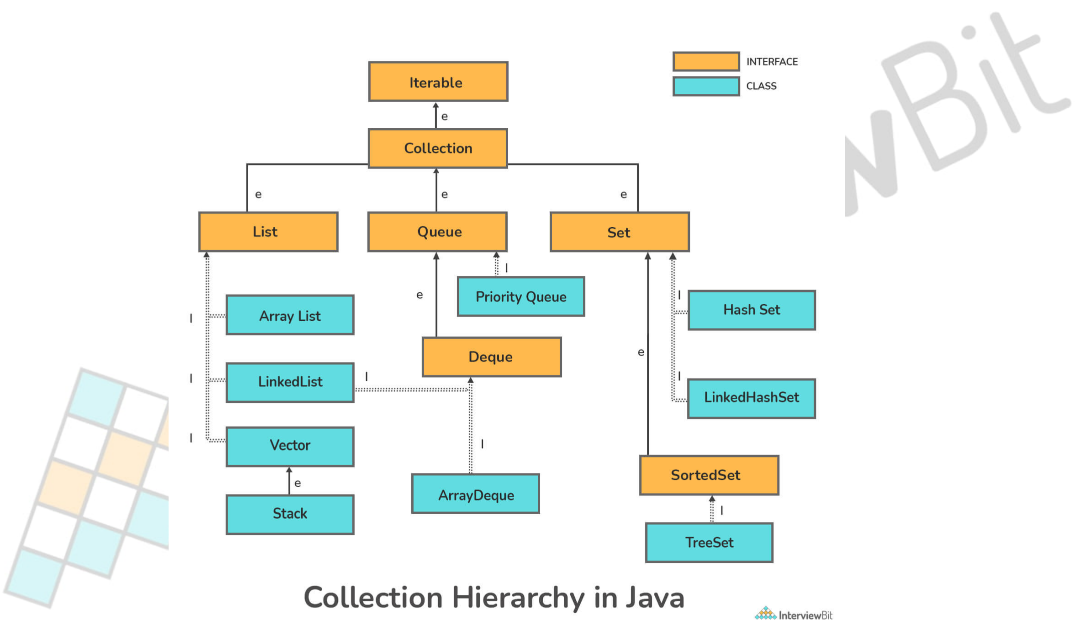
Collections: The java.util.package has a utility class called Collections. It defines various utility methods for working with collections, such as sorting and searching. All of the methods are static. These techniques give developers much-needed convenience, allowing them to interact with Collection Framework more successfully. It provides methods like sort() to sort the collection elements in the normal sorting order, and min() and max() to get the minimum and maximum value in the collection elements, respectively.


Collection
• The root interface in the collection hierarchy
Collections
• This class contains only static methods that operate on or return collections
InterviewBit

Collection	Collections
It's used to represent a collection of separate objects as a single entity.	It defines a number of useful methods for working with collections.
It is an interface.	It is a utility class.
Since Java 8, the Collection is an interface with a static function. Abstract and default methods can also be found in the Interface.	It only has static methods in it.

3. Explain the hierarchy of the Collection framework in Java.

The entire collection framework hierarchy is made up of four fundamental interfaces: Collection, List, Set, Map, and two specific interfaces for sorting called SortedSet and SortedMap. The java.util package contains all of the collection framework's interfaces and classes. The following diagram depicts the Java collection structure.



Collection Hierarchy in Java

Here, **e** denotes extends, **i** denotes implements

- **Extends**: The keyword extends is used to create inheritance between two classes and two interfaces.
- **Implements**: The keyword implements are used to create inheritance across classes and interfaces.

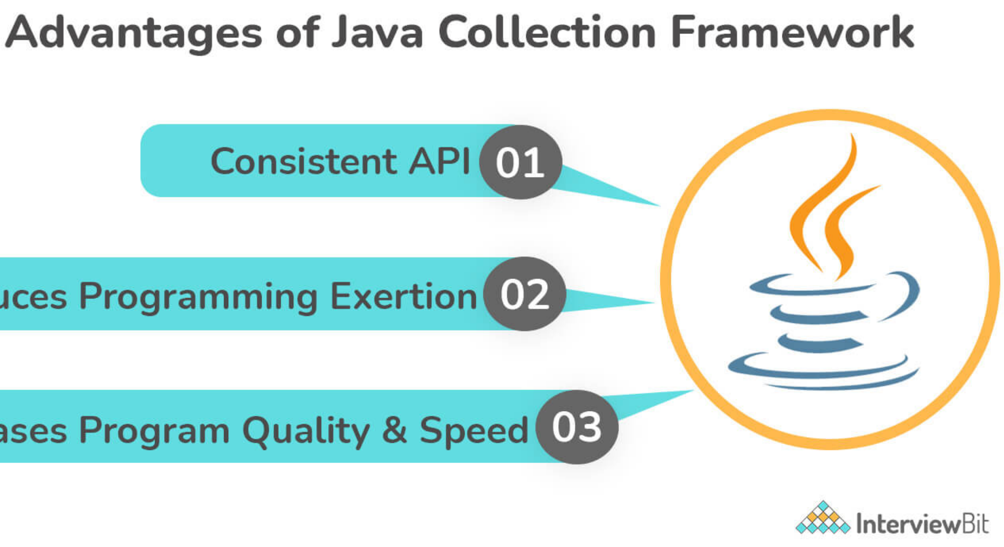
4. What are the advantages of the Collection framework?

Following are the advantages of the Collection framework:-

Consistent API: The API has a core set of interfaces like Collection, Set, List, or Map, and all the classes (ArrayList, LinkedList, Vector, and so on) that implement these interfaces have some common set of methods.

Cuts programming effort: Instead of worrying about the Collection's design, a programmer may concentrate on how best to use it in his program. As a result, the fundamental principle of Object-oriented programming (i.e. abstraction) has been applied successfully.

Improves program speed and quality by offering high-performance implementations of useful data structures and algorithms, as the programmer does not have to worry about the optimum implementation of a certain data structure in this scenario. They can simply use the best implementation to improve the performance of their program significantly.


Advantages of Java Collection Framework
Consistent API 01
Reduces Programming Exertion 02
Increases Program Quality & Speed 03
InterviewBit

5. Explain the various interfaces used in the Collection framework.

The collection framework has several interfaces, each of which is used to store a different sort of data. The interfaces included in the framework are listed below.

1. Iterable Interface: This is the collection framework's primary interface. The iterable interface is extended by the collection interface. As a result, all interfaces and classes implement this interface by default. This interface's main purpose is to provide an iterator for the collections. As a result, this interface only has one abstract method, the iterator.

2. Collection Interface: The collection framework's classes implement this interface, which extends the iterable interface. This interface covers all of the basic methods that every collection has, such as adding data to the collection, removing data from the collection, clearing data, and so on. All of these methods are incorporated in this interface because they are used by all classes, regardless of their implementation style. Furthermore, including these methods in this interface guarantees that the method names are consistent across all collections. In summary, we may conclude that this interface lays the groundwork for the implementation of collection classes.

3. List Interface: The collection interface has a child interface called the list interface. This interface is devoted to list data, in which we can store all of the objects in an ordered collection. This also allows for the presence of redundant data. Various classes, such as ArrayList, Vector, Stack, and others, implement this list interface. We can create a list object with any of these classes because they all implement the list.

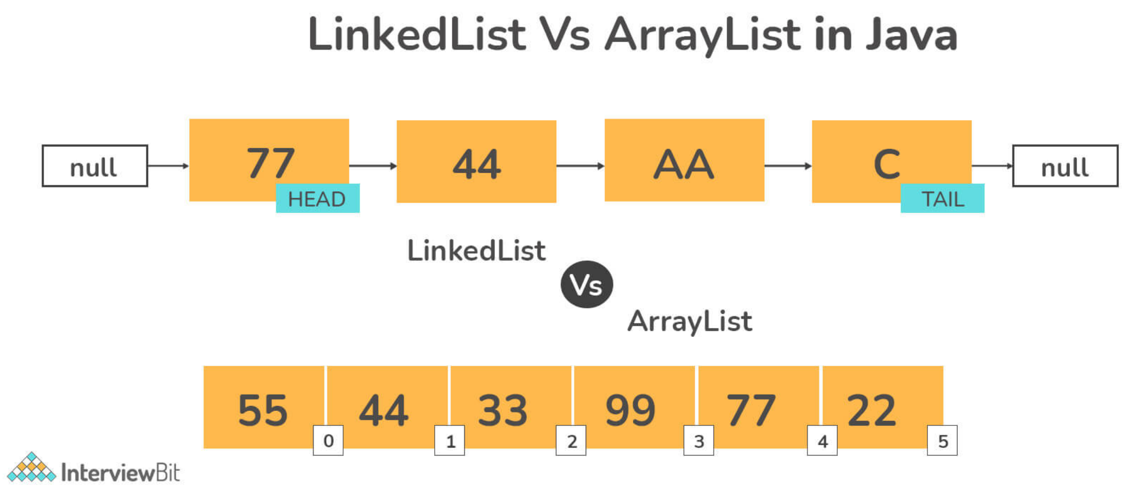
4. Queue Interface: A queue interface, as the name implies, follows the FIFO (First In First Out) order of a real-world queue line. This interface is for storing all elements in which the order of the elements is important. When we try to shop at a store, for example, the bills are issued on a first-come, first-served basis. As a result, the individual whose request is first in line receives the bill first. PriorityQueue, Deque, ArrayDeque, and other classes are available. Because all of these subclasses implement the queue, we can use any of them to create a queue object.

5. Deque Interface: It differs slightly from the queue data structure. Deque, also known as a double-ended queue, is a data structure in which elements can be added and removed from both ends. The queue interface is extended by this interface. ArrayDeque is the class that implements this interface. Because this class implements the deque, we can use it to create a deque object.

6. Set Interface: A set is an unordered group of objects in which duplicate values cannot be kept. This collection is utilised when we want to avoid duplication of things and only keep the ones that are unique. Various classes, such as HashSet, TreeSet, LinkedHashSet, and others, implement this set interface. We can create a set object with any of these classes because they all implement the set.

7. Sorted Set Interface: This interface resembles the set interface in appearance. The only difference is that this interface provides additional methods for maintaining element ordering. The sorted set interface is an extension of the set interface that is used to manage sorted data. TreeSet is the class that implements this interface. We can create a SortedSet object using this class because it implements the SortedSet interface.

6. Difference between ArrayList and LinkedList.

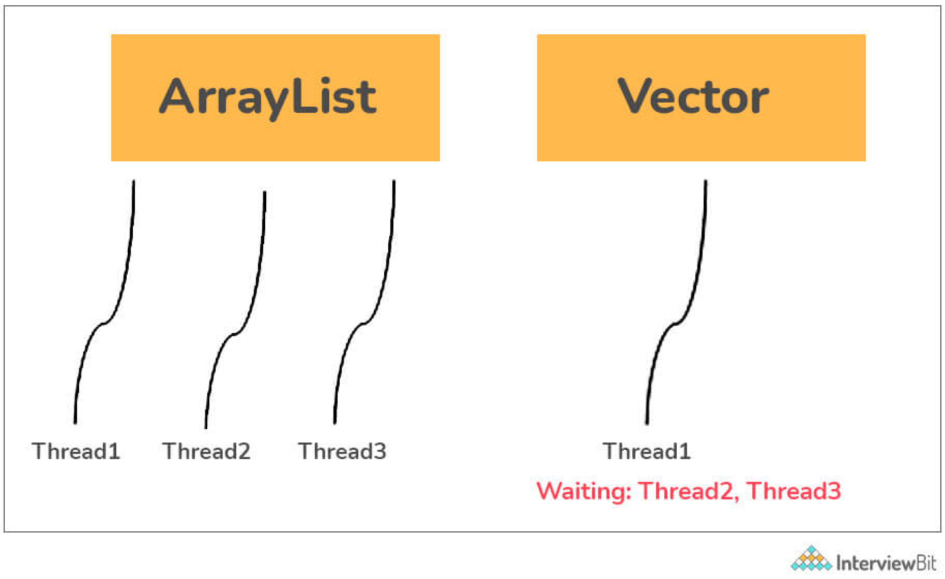


ArrayList	LinkedList
The elements of this class are stored in a dynamic array. This class now supports the storage of all types of objects thanks to the addition of generics.	The elements of this class are stored in a doubly-linked list. This class, like the ArrayList, allows for the storage of any type of object.
The List interface is implemented by this class. As a result, this serves as a list.	The List and Deque interfaces are both implemented by this class. As a result, it can be used as both a list and a deque.
Because of the internal implementation, manipulating an ArrayList takes longer. Internally, the array is scanned and the memory bits are shifted whenever we remove an element.	Because there is no concept of changing memory bits in a doubly-linked list, manipulating it takes less time than manipulating an ArrayList. The reference link is changed after traversing the list.
This class is more useful when the application requires data storage and access.	This class is more useful when the application requires data manipulation.

7. Differentiate between ArrayList and Vector in java.

Following are the differences between ArrayList and Vector in java :

- Vector is synchronized, which means that only one thread can access the code at a time, however, ArrayList is not synchronized, which means that multiple threads can operate on ArrayList at the same time. In a multithreading system, for example, if one thread is executing an add operation, another thread can be performing a removal action.
  If multiple threads access ArrayList at the same time, we must either synchronize the code that updates the list fundamentally or enable simple element alterations. The addition or deletion of element(s) from the list is referred to as structural change. It is not a structural change to change the value of an existing element.



- **Data Growth**: Both ArrayList and Vector dynamically expand and shrink to make the most use of storage space, but the manner they do it is different. If the number of elements in an array exceeds its limit, ArrayList increments 50% of the current array size, while vector increments 100%, thereby doubling the current array size.
- **Performance**: ArrayList is faster than vector operations because it is non-synchronized, but vector operations are slower since they are synchronized (thread-safe). When one thread works on a vector, it acquires a lock on it, requiring any other threads working on it to wait until the lock is released.
- Vector can traverse over its elements using both Enumeration and Iterator, whereas ArrayList can only traverse using Iterator.

8. Differentiate between List and Set in Java.

The List interface is used to keep track of an ordered collection. It is the Collection's child interface. It is an ordered collection of objects that allows for the storage of duplicate values. The insertion order is preserved in a list, which enables positional access and element insertion.

The set interface is part of java.util package and extends the Collection interface. It is an unordered collection of objects in which duplicate values cannot be stored. It's an interface for using the mathematical set. This interface inherits the Collection interface's methods and adds a feature that prevents duplicate elements from being inserted.

Set	List
It is an unordered sequence.	It is an ordered sequence.
Duplicate elements are not permitted in Set.	Duplicate elements are allowed in the list
Access to items from a certain position is not permitted.	Elements can be accessed based on their position.
A null element can only be stored once.	It is possible to store several null elements.

9. Differentiate between Iterator and ListIterator in Java.

In Java's Collection framework, iterators are used to obtain elements one by one. It can be used on any type of Collection object. We can execute both read and remove operations using Iterator. Iterator must be used whenever we want to iterate elements in all Collection framework implemented interfaces, such as Set, List, Queue, and Deque, as well as all Map interface implemented classes. The only cursor accessible for the entire collection framework is the iterator.

ListIterator is only useful for classes that implement List collections, such as array lists and linked lists. It can iterate in both directions. When we wish to enumerate List elements, we must use ListIterator. This cursor has additional methods and capabilities than the iterator.

Iterator
V/S
ListIterator
InterviewBit

Iterator	ListIterator
Only has the ability to traverse components in a Collection in a forward direction.	In both forward and backward orientations, can traverse components in a Collection.
Iterators cannot be used to obtain indexes.	It offers methods to get element indexes at any time while traversing List, such as next Index() and previous Index().
It aids in the traversal of Maps, Lists, and Sets.	Only List may be traversed, not the other two.
It throws a Concurrent Modification Exception since it can't add elements.	At any time, you can quickly add elements to a collection.
next(), remove(), and has Next are some of the Iterator's functions ().	next(), previous(), has Next(), has Previous(), and add() are some of the List Iterator's methods

10. Differentiate between HashSet and TreeSet. When would you prefer TreeSet to HashSet?

Following are the differences between HashSet and TreeSet:-

- **Internal implementation and speed**
  - **HashSet:** For search, insert, and remove operations, it takes constant time on average. TreeSet is slower than HashSet. A hash table is used to implement HashSet.
  - **TreeSet:** For search, insert, and delete, TreeSet takes O(Log n), which is higher than HashSet. TreeSet, on the other hand, preserves ordered data. Higher() (Returns the least higher element), floor(), ceiling(), and other operations are also supported. In TreeSet, these operations are likewise O(Log n), and HashSet does not implement them. A Self-Balancing Binary Search Tree is used to implement TreeSet (Red Black Tree). In Java, TreeSet is backed by TreeMap.
- **Way of storing elements**
  The elements of a HashSet are not ordered. In Java, the TreeSet class keeps objects in a Sorted order defined by the Comparable or Comparator methods. By default, TreeSet components are sorted in ascending order. It has a number of methods for dealing with ordered sets, including first(), last(), headSet(), tailSet(), and so on.
- **Allowing Null values**
  Null objects are allowed in HashSet. TreeSet does not allow null objects and throws a NullPointerException. This is because TreeSet compares keys using the compareTo() method, which throws java.lang. NullPointerException.
- **Comparison**
  HashSet compares two objects in a Set and detects duplicates using the equals() method. For the same purpose, TreeSet employs the compareTo() method. If equals() and compareTo() are not consistent, that is, if equals() returns true for two equal objects but compareTo() returns zero, the contract of the Set interface will be broken, allowing duplicates in Set implementations like TreeSet.

Following are the cases when TreeSet is preferred to HashSet :

1. Instead of unique elements, sorted unique elements are required. TreeSet returns a sorted list that is always in ascending order.
2. The locality of TreeSet is higher than that of HashSet. If two entries are close in order, TreeSet places them in the same data structure and hence in memory, but HashSet scatters the entries over memory regardless of the keys to which they are linked.
3. To sort the components, TreeSet employs the Red-Black tree method. TreeSet is a fantastic solution if you need to do read/write operations regularly.

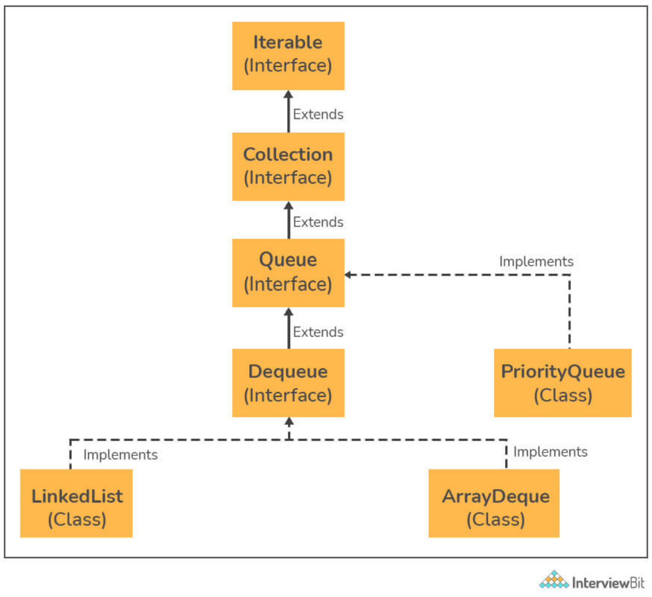
11. Can you add a null element into a TreeSet or HashSet?

We can add null elements in a HashSet but we cannot add null elements in a TreeSet. The reason is that TreeSet uses the compareTo() method for comparing and it throws a NullPointerException when it encounters a null element.

12. What is a priority queue in Java?

When the objects are meant to be processed in order of priority, a PriorityQueue is used. A Queue is known to follow the First-In-First-Out method, however, there are occasions when the components of the queue must be handled in order of priority, which is where the PriorityQueue comes into play. The priority heap is the foundation of the PriorityQueue. The members of the priority queue are ordered according to natural ordering or by a Comparator provided at queue construction time.

Serializable, Iterable<E>, Collection<E>, Queue<E> interfaces are implemented by the PriorityQueue class in Java.



13. What are some of the best practices while using Java Collections?

Following are some of the best practices while using Java Collections :

- **Selecting the appropriate Collection:**
  Before we use a collection, we must choose the most relevant collection for the problem we are seeking to solve. If we pick the wrong one, our program may still run, but it will be inefficient. On the other hand, if we pick the right one, our solution will be a lot simpler and our program will run much faster.
- **Specifying the initial capacity of the Collection :**
  Almost all collection classes contain an overloaded constructor that determines the collection's initial capacity. That is, if we know exactly how many pieces will be added to the collection, we can define the initial capacity when establishing a new instance.
- **Using isEmpty() instead of size():**
  To check if a collection is empty or not we should use the isEmpty() method rather than finding the size of the collection and comparing it with zero. This enhances the readability of the code.
- **Using Iterators for iterating over the collections:**
  We should use iterators for traversing over the collection elements instead of using a for loop for the same. The reason is that the iterator may throw ConcurrentModificationException if any other thread tries to modify the collection after the iterator has been created. This saves us from bugs.
- **Using Concurrent Collections over synchronized wrappers:**
  Instead of utilizing the synchronized collections generated by the Collections.synchronizedXXX() methods, we should consider using concurrent collections from the java.util.concurrent package in multi-threaded applications. Because concurrent collections employ various synchronization strategies such as copy-on-write, compare-and-swap, and specific locks, they are designed to give maximum performance in concurrent applications.
- **Eliminating Unchecked warnings:**
  We should not disregard unchecked warnings from the Java compiler. The ideal practice is to get rid of any warnings that aren't checked.
- **Favoring Generic types:**
  We should build new methods with generic parameters in mind, and convert existing methods to use type parameters, just as we should with generic types because generic methods are safer and easier to use than non-generic ones. Generic methods also aid in the creation of APIs that are both general and reusable.

14. Differentiate between Set and Map in Java.

The **Set** interface is provided by the Java.util package. The set interface is established by extending the collection interface. We can't add the same element to it since it won't let us. Because it contains elements in a sorted order, it does not keep the insertion order. The Set interface in Java is used to build the mathematical Set.

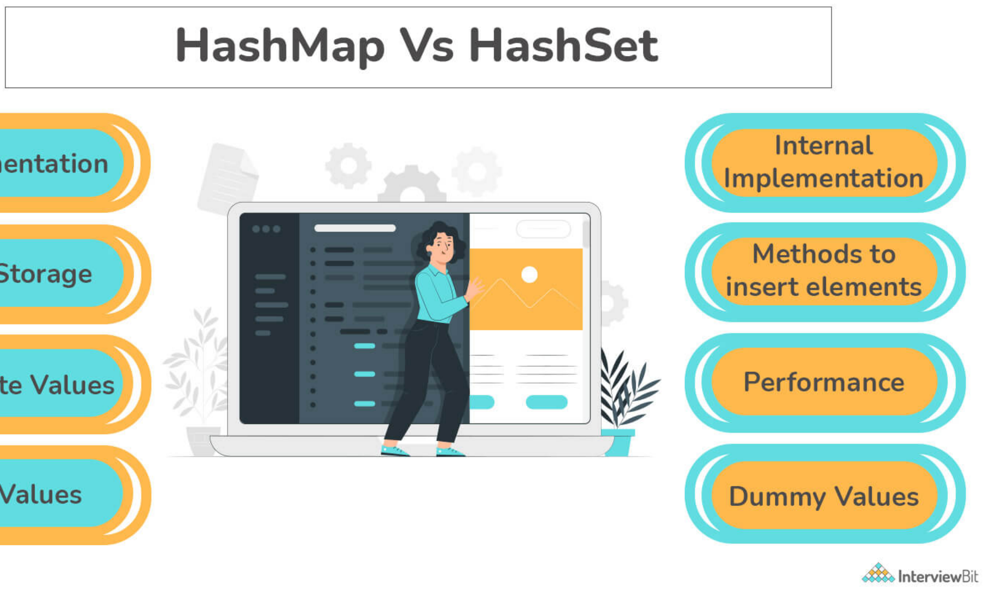
Map is similar to Set in that it is used to store a collection of objects as a single entity. A key-value pair is used to store each object. Because each value is associated with a unique key, we can quickly obtain the value using just the key.

Set	Map
It cannot have values that are repeated. It is not possible to add the same elements to a set. Only the unique value is stored in each class that implements the Set interface.	It is possible for different keys to have the same value. The map has a unique key and values that are repeated.
Using the keyset() and entryset() methods, we can quickly iterate the Set items.	It is not possible to iterate across map elements. To iterate the elements, we must convert Map to Set.
The Set interface does not keep track of insertion order. Some of its classes, such as LinkedHashSet, however, keep the insertion order.	The Map does not keep track of the insertion sequence. Some Map classes, such as TreeMap and LinkedHashMap, do the same thing.

15. Differentiate between HashSet and HashMap.

HashSet is a Set Interface implementation that does not allow duplicate values. The essential point is that objects stored in HashSet must override equals() and hashCode() methods to ensure that no duplicate values are stored in our set.

HashMap is a Map Interface implementation that maps a key to a value. In a map, duplicate keys are not permitted.



HashSet	HashMap
It implements the Set Interface.	It implements the Map Interface.
It does not allow duplicate values.	The key needs to be unique while two different keys can have the same value.
While adding an element it requires only one object as a parameter.	While adding an entry, it requires two object values, the **Key** and the **Value** as the parameter.
Internally, HashSet uses HashMap to add entries. The key K in a HashSet is the argument supplied in the add(Object) method. For each value supplied in the add(Object) method, Java assigns a dummy value.	There is no concept of duplicate values.
It is slower than HashMap.	It is faster than HashSet.
It uses the add() method for adding elements.	It uses the put() method for adding data elements.

16. What is the default size of the load factor in hashing based collection?

The default load factor size is **0.75**. The default capacity is calculated by multiplying the initial capacity by the load factor.

17. Differentiate between Array and ArrayList in Java.

Following are the differences between Arrays and ArrayLists in Java :

- Java provides arrays as a fundamental functionality. ArrayList is a component of Java's collection system. As a result, It is used to access array members, while ArrayList provides a set of methods for accessing and modifying components.
- ArrayList is not a fixed-size data structure, but Array is. When creating an ArrayList object, there is no need to provide its size. Even if we set a maximum capacity, we can add more parts afterward.
- Arrays can include both primitive data types and class objects, depending on the array's definition. ArrayList, on the other hand, only accepts object entries and not primitive data types. Note that when we use arraylist.add(1);, the primitive int data type is converted to an Integer object.
- Members of ArrayList are always referencing to objects at various memory locations since ArrayList can't be constructed for primitive data types As a result, the actual objects in an ArrayList are never kept in the same place. The references to the real items are maintained in close proximity. Whether an array is primitive or an object depends on the type of the array. Actual values for primitive kinds are continuous regions, whereas allocation for objects is equivalent to ArrayList.
- Many other operations, such as indexOf() and delete(), are supported by Java ArrayList. Arrays do not support these functions.

18. How can you make an ArrayList read-only in Java?

With the help of Collections.unmodifiableList() method, we can easily make an ArrayList read-only. This function takes a changeable ArrayList as an input and returns the ArrayList's read-only, unmodified view.

Example:

```
import java.util.*;

public class InterviewBit {
   public static void main(String[] args)
       throws Exception
   {
       try {

           // creating object of ArrayList<String>
           List<String> sample_list = new ArrayList<String>();

           sample_list.add(“practice”);
           sample_list.add(“solve”);
           sample_list.add(“interview”);

           // displaying the initial list
           System.out.println("The initial list is : "
                              + sample_list);

           // using unmodifiableList() method
           List<String>
               read_only_list = Collections
                                   .unmodifiableList(sample_list);

           // displaying the read-only list
           System.out.println("The ReadOnly ArrayList is : "
                              + read_only_list);

           // Trying to add an element to the read-only list
           System.out.println("Trying to modify the ReadOnly ArrayList.");
           read_only_list.add(“job”);
       }

       catch (UnsupportedOperationException e) {
           System.out.println("The exception thrown is : " + e);
       }
   }
}
```

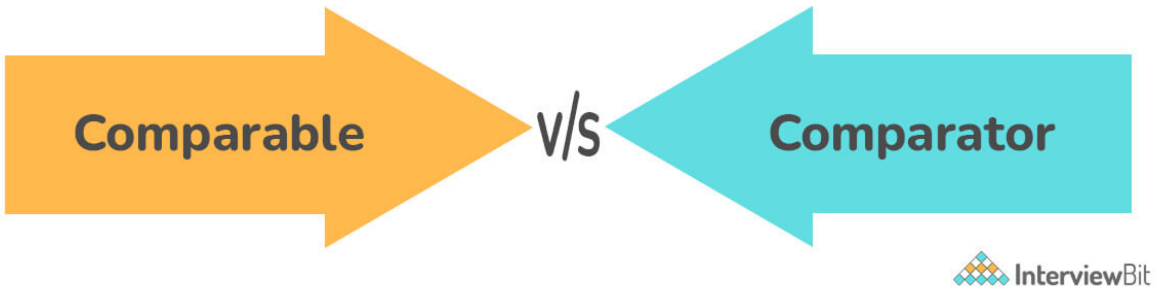
Output:

```
The initial list is : [practice, solve, interview]
The ReadOnly ArrayList is : [practice, solve, interview]
Trying to modify th eReadOnly ArrayList.
Exception thrown : java.lang.UnsupportedOperationException
```

We can see that as we try to add an element to a read-only ArrayList we get an exception thrown.

Java Collections Interview Questions for Experienced

19. Differentiate between Comparable and Comparator in the context of Java.

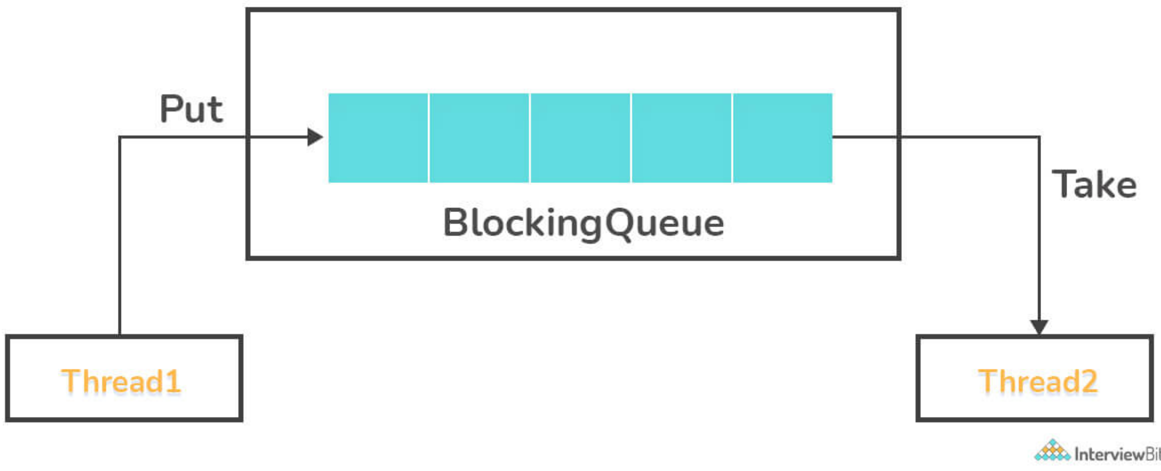


Comparable	Comparator
A single sorting sequence is provided by Comparable. To put it another way, we can sort the collection by a single attribute such as id, name, or price.	Multiple sorting sequences are available in the Comparator. To put it another way, we can sort the collection based on different criteria such as id, name, and price.
To sort elements, Comparable provides the compareTo() method.	To order elements, the Comparator provides the compare() method.
It is present in the java.lang package.	It is present in the java.util package.
The original class is affected by Comparable, i.e. the real class is changed.	The original class is unaffected by the comparator, i.e. the real class is unaffected.
The Collections.sort(List) method can be used to sort Comparable type list members.	The Collections.sort(List, Comparator) method can be used to sort the list components of the Comparator type.

20. What do you understand about BlockingQueue in Java?

BlockingQueue is an interface that has been included along with a number of other concurrent Utility classes such as ConcurrentHashMap, Counting Semaphore, CopyOnWriteArrrayList, and so on. In addition to queueing, the BlockingQueue interface enables flow control by adding blocking if either BlockingQueue is full or empty.

A thread attempting to enqueue an element in a full queue will be blocked until another thread clears the queue, either by dequeuing one or more elements or by clearing the queue entirely. It also prevents a thread from deleting from an empty queue until another thread inserts an item. A null value is not accepted by BlockingQueue. Implementations of the Java BlockingQueue interface are thread-safe. BlockingQueue's methods are all atomic and use internal locks or other forms of concurrency management.



There are two types of BlockingQueue in Java. They are as follows :

Unbounded Queue: The blocked queue's capacity will be set to Integer. MAX VALUE. An unbounded blocking queue will never block since it has the potential to grow to a very big size. As you add more pieces, the size of the queue grows.

Example :

```
BlockingQueue unbounded_queue = new LinkedBlockingDeque();
```

Bounded Queue: The bounded queue is the second type of queue. In the case of a bounded queue, the capacity of the queue can be passed to the constructor when the blocking queue is created.

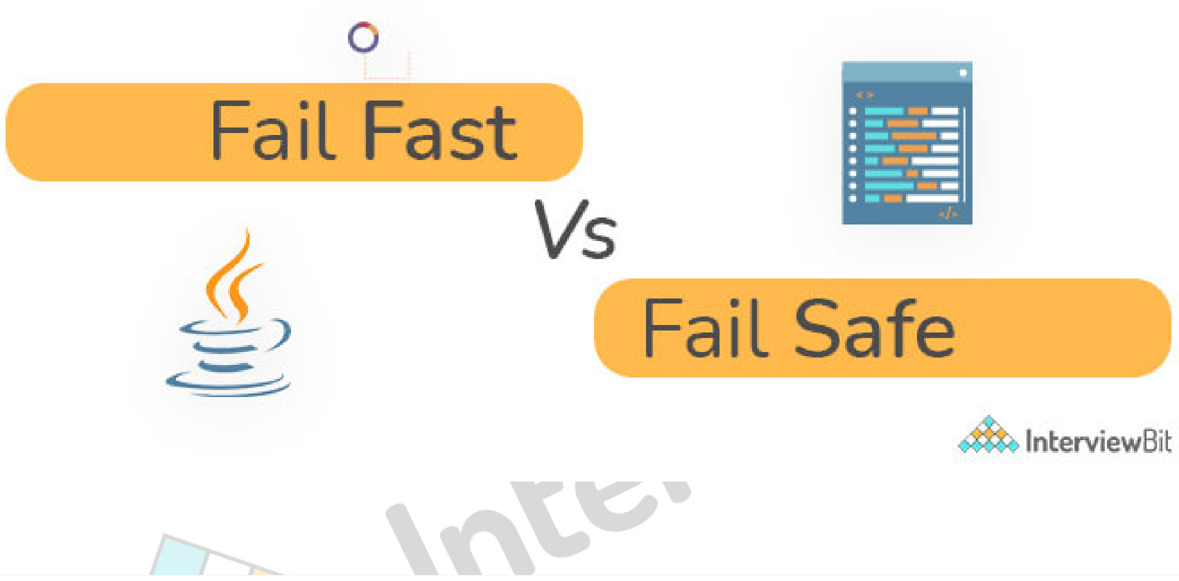
Example:

```
// Creates a Blocking Queue with capacity 10
BlockingQueue bounded_queue = new LinkedBlockingDeque(10);
```

21. Explain fail-fast and fail-safe iterators. Differentiate between them.

If the collection's structure is changed, **Fail-Fast** iterators immediately throw ConcurrentModificationException. While a thread is iterating over a collection, structural alteration includes adding or deleting any element. Fail-safe Iterator classes include ArrayList Iterator and HashMap Iterator. Fail-fast iterators use an internal indicator called modCount, which is updated each time a collection is modified, to determine if the collection has been structurally modified or not. When a fail-fast iterator gets the next item (through the next() method), it checks the modCount flag, and if it discovers that the modCount has been changed after the iterator was generated, it throws a ConcurrentModificationException.

If a collection is structurally updated while iterating over it, **fail-safe** iterators don't throw any exceptions. Because they operate on a clone of the collection rather than the original collection, they are referred to as fail-safe iterators. Fail-safe Iterators include the CopyOnWriteArrayList and ConcurrentHashMap classes.



Fail-Fast	Fail-Safe
These types of iterators do not allow modifying the collection while iterating over it.	These types of iterators allow modifying the collection while iterating over it.
It throws ConcurrentModificationException if the collection is modified while iterating over it.	No exception is thrown if the collection is modified while iterating over it.
It uses the original collection while traversing the elements.	It uses a copy of the original collection while traversing over it.
No extra memory is required in this case.	Extra memory is required in this case.

22. What is the purpose of RandomAccess Interface? Name a collection type which implements this interface.

RandomAccess, like the Serializable and Cloneable interfaces, is a marker interface. There are no methods defined in any of these marker interfaces. Rather, they designate a class as having a specific capability.

The Randrapid omAccess interface indicates whether or not a given java.util.List implementation supports random access. This interface seeks to define a vague concept: what does it mean to be fast? A simple guide is provided in the documentation: The List has fast random access if repeated access using the List.get( ) method is faster than repeated access using the Iterator.next( ) method.

Repeated access using List.get( ):

```
Object obj;
for (int i=0, n=list.size(  ); i < n; i++)
 obj = list.get(i);
```

Repeated access using Iterator.next( ):

```
Object obj;
for (Iterator itr=list.iterator(  ); itr.hasNext(  ); )
 obj = itr.next( );
```

23. Differentiate between Iterator and Enumeration.

Iterator: Because it can be applied to any Collection object, it is a universal iterator. We can execute both read and remove operations using Iterator. It's an enhanced version of Enumeration that adds the ability to remove an element from the list.

Enumeration: An enumeration (or enum) is a data type that is defined by the user. It's mostly used to give integral constants names, which make a program easier to comprehend and maintain. Enums are represented in Java (since 1.5) through the enum data type.

Iterator	Enumeration
Iterator is a universal cursor since it works with all collection classes.	Because it only applies to legacy classes, enumeration is not a universal cursor.
Iterators can make changes (for example, the delete() method removes an element from a Collection during traversal).	The Enumeration interface is a read-only interface, which means you can't make any changes to the Collection while traversing its elements.
The remove() method is available in the Iterator class.	The remove() method is not available in the enumeration.
Iterator is not a legacy interface. Iterator can traverse HashMaps, LinkedLists, ArrayLists, HashSets, TreeMaps, and TreeSets.	Enumeration is a legacy interface for traversing Hashtables and Vectors.

24. What is the use of Properties class in Java? What is the advantage of the Properties file?

The key and value pair are both strings in the properties object. The java.util.Properties class is a Hashtable subclass.

It can be used to calculate the value of a property based on its key. The Properties class has methods for reading and writing data to and from the properties file. It can also be used to obtain a system's attributes.

Advantage of the Properties file:

If the information in a properties file is modified, no recompilation is required: You don't need to recompile the java class if any information in the properties file changes. It is used to keep track of information that needs to be updated frequently.

Example:

Let us first create a properties file named "info.properties" having the following content :

user = success

password = determination

Let us now create a java class to read data from the properties file

```
import java.util.*;
import java.io.*;
public class Sample {
public static void main(String[] args)throws Exception{
   FileReader reader = new FileReader("info.properties");
   Properties obj_p = new Properties();
   obj_p.load(reader);
   System.out.println(obj_p.getProperty("user"));
   System.out.println(obj_p.getProperty("password"));
}
}
```

Output:

```
success
determination
```

25. Differentiate between HashMap and HashTable.

Following are the differences between HashMap and HashTable:

- HashMap is a non-synchronized data structure. It is not thread-safe and cannot be shared across many threads without the use of synchronization code, while Hashtable is synchronized. It's thread-safe and can be used by several threads.
- HashMap supports one null key and numerous null values, whereas Hashtable does not.
- If thread synchronization is not required, HashMap is often preferable over HashTable.

26. Why does HashMap allow null whereas HashTable does not allow null?

The objects used as keys must implement the hashCode and equals methods in order to successfully save and retrieve objects from a HashTable. These methods cannot be implemented by null because it is not an object. HashMap is a more advanced and improved variant of Hashtable.HashMap was invented after HashTable to overcome the shortcomings of HashTable.

27. How can you synchronize an ArrayList in Java?

An ArrayList can be synchronized using the following two ways :

- **Using Collections.synchronizedList() method:**
  All access to the backup list must be done through the returning list in order to perform serial access. When iterating over the returned list, it is critical that the user manually synchronizes.

```
import java.util.*;

class InterviewBit
{
   public static void main (String[] args)
   {
       List<String> synchronized_list =
          Collections.synchronizedList(new ArrayList<String>());

       synchronized_list.add("learn");
       synchronized_list.add("practice");
       synchronized_list.add("solve");
       synchronized_list.add("interview");

       synchronized(synchronized_list)// must be declared
       {

           Iterator it = synchronized_list.iterator();

           while (it.hasNext())
               System.out.println(it.next());
       }
   }
}
```

Output:

```
learn
practice
solve
interview
```

- **Using CopyOnWriteArrayList:**

Syntax:

```
CopyOnWriteArrayList<T> list_name = new CopyOnWriteArrayList<T>();
```

Here, a thread-safe variant of ArrayList is created. T represents generic.

All mutative actions (e.g. add, set, remove, etc.) are implemented by generating a separate copy of the underlying array in this thread-safe variation of ArrayList. It accomplishes thread safety by generating a second copy of List, which differs from how vectors and other collections achieve thread-safety.

Even if copyOnWriteArrayList is modified after an iterator is formed, the iterator does not raise ConcurrentModificationException because the iterator is iterating over a separate copy of ArrayList while a write operation is occurring on another copy of ArrayList.

Example:

```
import java.io.*;
import java.util.Iterator;
import java.util.concurrent.CopyOnWriteArrayList;

class InterviewBit
{
   public static void main (String[] args)
   {

       CopyOnWriteArrayList<String> synchronized_list
           = new CopyOnWriteArrayList<String>();// creating a thread-safe Arraylist.

       // Adding strings to the synchronized ArrayList
       synchronized_list.add("learn");
       synchronized_list.add("practice");
       synchronized_list.add("solve");
       synchronized_list.add("interview");

       System.out.println("The synchronized ArrayList has the following elements :");

       // Iterating on the synchronized ArrayList using an iterator.
       Iterator<String> it = synchronized_list.iterator();

       while (it.hasNext())
           System.out.println(it.next());
   }
}
```

Output:

```
The synchronized ArrayList has the following elements :
learn
practice
solve
interview
```

28. Why do we need synchronized ArrayList when we have Vectors (which are synchronized) in Java?

Following are the reasons why we need synchronized ArrayLists even though we have Vectors :

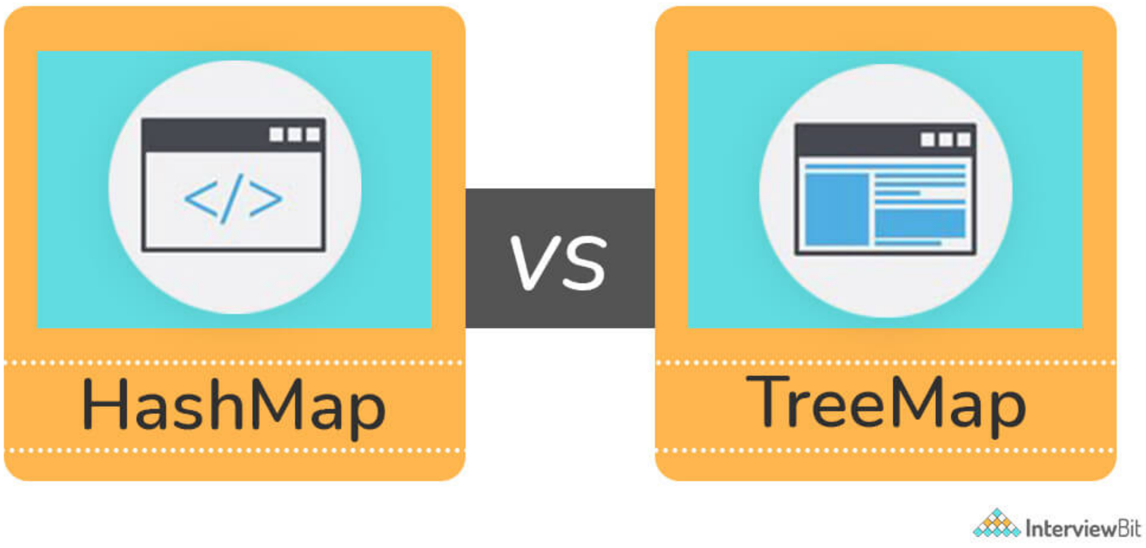
- Vector is slightly slower than ArrayList due to the fact that it is synchronized and thread-safe.
- Vector's functionality is that it synchronizes each individual action. A programmer's preference is to synchronize an entire sequence of actions. Individual operations are less safe and take longer to synchronize.
- Vectors are considered outdated in Java and have been unofficially deprecated. Vector also synchronizes on a per-operation basis, which is almost never done. Most java programmers prefer to use ArrayList since they will almost certainly synchronize the arrayList explicitly if they need to do so.

29. Why does not the Map interface extend the Collection Interface or vice-versa?

If Map extends the Collection Interface, "Key-value pairs" can be the only element type for this type of Collection, although this provides a very limited (and not really useful) Map abstraction. You can't inquire what value a specific key corresponds to, and you can't delete an entry without knowing what value it corresponds to.

The three "Collection view procedures" on Maps represent the fact that Maps can be viewed as Collections (of keys, values, or pairs) (keySet, entrySet, and values). While it is theoretically feasible to see a List as a Map mapping indices to items, this has the unfortunate side effect of changing the Key associated with every element in the List prior to the deleted member. This is the reason why Collection can not be made to extend the Map Interface either.

30. Differentiate between HashMap and TreeMap in the context of Java.



HashMap	TreeMap
The Java HashMap implementation of the Map interface is based on hash tables.	Java TreeMap is a Map interface implementation based on a Tree structure.
The Map, Cloneable, and Serializable interfaces are implemented by HashMap.	NavigableMap, Cloneable, and Serializable interfaces are implemented by TreeMap.
Because HashMap does not order on keys, it allows for heterogeneous elements.	Because of the sorting, TreeMap allows homogenous values to be used as a key.
HashMap is quicker than TreeMap because it offers O(1) constant-time performance for basic operations such as to get() and put().	TreeMap is slower than HashMap because it performs most operations with O(log(n)) performance, such as add(), remove(), and contains().
A single null key and numerous null values are allowed in HashMap.	TreeMap does not allow null keys, however multiple null values are allowed.
To compare keys, it uses the Object class's equals() method. It is overridden by the Map class's equals() function.	It compares keys using the compareTo() method.
HashMap does not keep track of any sort of order.	The elements are arranged in chronological sequence (ascending).

Java Collection Programs

31. Given an array in Java, convert it to a collection.

We can convert an array to a collection using the asList() method of the Arrays class in Java.

```
//including the required header files
import java.util.*;
public class Convert_Array_To_Collection {
   public static void main(String args[])
   {
       //creating a sample array
       String sample_array[]
           = { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sur

       int length_array = sample_array.length;
       System.out.println("The input elements are as follows : ");
       for(int i = 0; i < length_array; i ++)
       {
           System.out.print(sample_array[i] + " ");
       }
       System.out.println();// setting the print cursor to the next line

       List converted_list = Arrays.asList(sample_array);// converting the array to a l

       // print converted elements
       System.out.println("The converted list is as follows : "
                          + converted_list);
   }
}
```

Output:

```
The input elements are as follows :
Monday Tuesday Wednesday Thursday Friday Saturday Sunday
The converted list is as follows : [Monday, Tuesday, Wednesday, Thursday, Friday, Satur
```

32. Write a program in Java to display the contents of a HashTable using enumeration.

We use the hasMoreElements and the nextElement methods of the Enumeration class to iterate through the HashMap.

```
//including the necessary header files
import java.util.Enumeration;
import java.util.Hashtable;
public class Iterate_HashTable {
  public static void main(String[] args) {
     Hashtable hash_table = new Hashtable();//creating a hash table
     hash_table.put("1", "Monday");
     hash_table.put("2", "Tuesday");
     hash_table.put("3", "Wednesday");
     hash_table.put("4", "Thursday");
     hash_table.put("5", "Friday");
     hash_table.put("6", "Saturday");
     hash_table.put("7", "Sunday");
     Enumeration enumeration_hash_table = hash_table.elements();//creating an enumerati

     //while loop runs until the hashtable has more entries in it
     while(enumeration_hash_table.hasMoreElements()) {
        System.out.println(enumeration_hash_table.nextElement());
     }
  }
}
```

Output:

```
Saturday
Friday
Thursday
Wednesday
Tuesday
Monday
Sunday
```

We notice that the order of the values is not the same as that of the order in which we inserted the key-value pair in the hashtable. This is because the elements of a Hashtable are not guaranteed to be in any particular sequence. The hashtable's implementation divides values into multiple buckets based on their Hashcode and internal implementation, which means that the same values may appear in a different order on different machines, runs, or versions of the framework. This is because Hashtables are designed to retrieve data by key rather than by order.

33. Write a program to shuffle all the elements of a collection in Java.

We use the shuffle() method of the Collections class.

```java
//importing the required header files
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class Shuffle_collection {
  public static void main(String[] argv) throws Exception {

     ArrayList<String> array_list = new ArrayList<String>();//creating an arraylist of

     array_list.add("Monday");
     array_list.add("Tuesday");
     array_list.add("Wednesday");
     array_list.add("Thursday");
     array_list.add("Friday");
     array_list.add("Saturday");
     array_list.add("Sunday");

     Collections.shuffle(array_list);//shuffling the arraylist

     System.out.println("The shuffled array list is as follows : " + array_list);//prir

  }
}
```

Output:

```
The shuffled array list is as follows : [Thursday, Friday, Saturday, Wednesday, Tuesday
```

34. Write a program in Java to clone a Treeset to another Treeset.

We use the clone() method of the TreeSet class to clone one TreeSet into another.

```
//importing the required header files
import java.util.TreeSet;
import java.util.Iterator;
 public class Clone_Tree_Set {
 public static void main(String[] args) {

     TreeSet<String> tree_set = new TreeSet<String>();//creating an empty tree set
     //adding values in the tree set
     tree_set.add("Monday");
     tree_set.add("Tuesday");
     tree_set.add("Wednesday");
     tree_set.add("Thursday");
     tree_set.add("Friday");
     tree_set.add("Saturday");
     tree_set.add("Sunday");

     //printing the original tree set
     System.out.println("The original tree set is as follows : " + tree_set);
     //cloning the tree set
     TreeSet<String> cloned_tree_set = (TreeSet<String>)tree_set.clone();
     //printing the cloned tree set
     System.out.println("The cloned tree set is as follows : " + cloned_tree_set);
    }
}
```

Output:

```
The original tree set is as follows : [Friday, Monday, Saturday, Sunday, Thursday, Tues
The cloned tree set is as follows : [Friday, Monday, Saturday, Sunday, Thursday, Tuesda
```

35. Write a program in java to get the collection view of the values present in a HashMap.

We use the values() function of the HashMap to get the collection view.

```
//importing the required header files
import java.util.*;
public class Collection_View {
   public static void main(String args[]){

   HashMap<String,String> hash_map = new HashMap<String,String>();//creating an empty h
   //adding key values to the hash map
   hash_map.put("1","Monday");
   hash_map.put("2","Tuesday");
   hash_map.put("3","Wednesday");
   hash_map.put("4","Thursday");
   hash_map.put("5","Friday");
   hash_map.put("6","Saturday");
   hash_map.put("7","Sunday");
   //printing the original hash map
   System.out.println("The original hash map is as follows : " + hash_map);
   //printing the collection view of the hash map
   System.out.println("The collection view is as follows : " + hash_map.values());
 }
}
```

Output:

```
The original hash map is as follows: {1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=F
The collection view is as follows : [Monday, Tuesday, Wednesday, Thursday, Friday, Satu
```

36. Write a program in java to join two arraylists into one arraylist.

We use the addAll() method of the ArrayList class to add the contents of both the given arraylists into a new arraylist.

```
//importing the required header files
import java.util.ArrayList;
import java.util.Collections;
 public class Join_Lists {
 public static void main(String[] args) {
        //creating the first array list
        ArrayList<String> list_1 = new ArrayList<String>();
        list_1.add("Monday");
        list_1.add("Tuesday");
        list_1.add("Wednesday");
        list_1.add("Thursday");
        //printing the first array list
        System.out.println("The elements of the first array list is as follows : " + lis
        //creating the second array list
        ArrayList<String> list_2 = new ArrayList<String>();
        list_2.add("Friday");
        list_2.add("Saturday");
        list_2.add("Sunday");
        //printing the second array list
        System.out.println("The elements of the second array list is as follows : " + li

        //creating the third array list
        ArrayList<String> joined_list = new ArrayList<String>();
        joined_list.addAll(list_1);//adding the elements of the first array list
        joined_list.addAll(list_2);//adding the elements of the second array list

        System.out.println("The elements of the joined array list is as follows : " + jo

    }
}
```

Output:

```
The elements of the first array list is as follows : [Monday, Tuesday, Wednesday, Thurs
The elements of the second array list is as follows : [Friday, Saturday, Sunday]
The elements of the joined array list is as follows : [Monday, Tuesday, Wednesday, Thur
```

Useful Resources:

[Basics of Java](#)

[Java Developer Skills](#)

Links to More Interview Questions

C Interview Questions	Php Interview Questions	C Sharp Interview Questions
Web Api Interview Questions	Hibernate Interview Questions	Node Js Interview Questions
Cpp Interview Questions	Oops Interview Questions	Devops Interview Questions
Machine Learning Interview Questions	Docker Interview Questions	Mysql Interview Questions
Css Interview Questions	Laravel Interview Questions	Asp Net Interview Questions
Django Interview Questions	Dot Net Interview Questions	Kubernetes Interview Questions
Operating System Interview Questions	React Native Interview Questions	Aws Interview Questions
Git Interview Questions	Java 8 Interview Questions	Mongodb Interview Questions
Dbms Interview Questions	Spring Boot Interview Questions	Power Bi Interview Questions
Pl Sql Interview Questions	Tableau Interview Questions	Linux Interview Questions
Ansible Interview Questions	Java Interview Questions	Jenkins Interview Questions