**Java Collections Interview Questions**

**46.What is an Iterator ?**

* The Iterator interface is used to step through the elements of a Collection.
* Iterators let you process each element of a Collection.
* Iterators are a generic way to go through all the elements of a Collection no matter how it is organized.
* Iterator is an Interface implemented a different way for every Collection.

**47.How do you traverse through a collection using its Iterator?**

To use an iterator to traverse through the contents of a collection, follow these steps:

* Obtain an iterator to the start of the collection by calling the collectionâ€™s ***iterator()*** method.
* Set up a loop that makes a call to ***hasNext()***. Have the loop iterate as long as ***hasNext()*** returns**true**.
* Within the loop, obtain each element by calling **next()**.

**48.How do you remove elements during Iteration?**

Iterator also has a method ***remove()*** when remove is called, the current element in the iteration is deleted.

**49.What is the difference between Enumeration and Iterator?**

|  |  |
| --- | --- |
| **Enumeration** | **Iterator** |
| Enumeration doesn't have a remove() method | Iterator has a remove() method |
| Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects | Can be *abstract, final, native, static*, or *synchronized* |

**Note**: So Enumeration is used whenever we want to make Collection objects as Read-only.

|  |
| --- |
|  |

**50.How is ListIterator?**

**ListIterator** is just like Iterator, except it allows us to access the collection in either the forward or backward direction and lets us modify an element

**51.What is the List interface?**

* The List interface provides support for ordered collections of objects.
* Lists may contain duplicate elements.

**52.What are the main implementations of the List interface ?**

The main implementations of the List interface are as follows :

* **ArrayList** : Resizable-array implementation of the List interface. The best all-around implementation of the List interface.
* **Vector** : Synchronized resizable-array implementation of the List interface with additional "legacy methods."
* **LinkedList** : Doubly-linked list implementation of the List interface. May provide better performance than the ArrayList implementation if elements are frequently inserted or deleted within the list. Useful for queues and double-ended queues (deques).

**53.What are the advantages of ArrayList over arrays ?**

Some of the advantages ArrayList has over arrays are:

* It can grow dynamically
* It provides more powerful insertion and search mechanisms than arrays.

**54.Difference between ArrayList and Vector ?**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| ArrayList is **NOT** synchronized by default. | Vector List is synchronized by default. |
| ArrayList can use only Iterator to access the elements. | Vector list can use Iterator and Enumeration Interface to access the elements. |
| The ArrayList increases its array size by 50 percent if it runs out of room. | A Vector defaults to doubling the size of its array if it runs out of room |
| ArrayList has no default size. | While vector has a default size of 10. |

**55.How to obtain Array from an ArrayList ?**

Array can be obtained from an ArrayList using ***toArray()***method on ArrayList.

List arrayList = new ArrayList();  
 arrayList.add(â€¦

ObjectÂ  a[] = **arrayList.toArray()**;

**56.Why insertion and deletion in ArrayList is slow compared to LinkedList ?**

* **ArrayList**internally uses and array to store the elements, when that array gets filled by inserting elements a new array of roughly 1.5 times the size of the original array is created and all the data of old array is copied to new array.
* During deletion, all elements present in the array after the deleted elements have to be moved one step back to fill the space created by deletion. In linked list data is stored in nodes that have reference to the previous node and the next node so adding element is simple as creating the node an updating the next pointer on the last node and the previous pointer on the new node. Deletion in linked list is fast because it involves only updating the next pointer in the node before the deleted node and updating the previous pointer in the node after the deleted node.

|  |
| --- |
|  |

**57.Why are Iterators returned by ArrayList called Fail Fast ?**

Because, if list is structurally modified at any time after the iterator is created, in any way except through the iterator's own remove or add methods, the iterator will throw a ConcurrentModificationException. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

**58.How do you decide when to use ArrayList and When to use LinkedList?**

If you need to support random access, without inserting or removing elements from any place other than the end, then ArrayList offers the optimal collection. If, however, you need to frequently add and remove elements from the middle of the list and only access the list elements sequentially, then LinkedList offers the better implementation.

**59.What is the Set interface ?**

* The Set interface provides methods for accessing the elements of a finite mathematical set
* Sets do not allow duplicate elements
* Contains no methods other than those inherited from Collection
* It adds the restriction that duplicate elements are prohibited
* Two Set objects are equal if they contain the same elements

**60.What are the main Implementations of the Set interface ?**

The main implementations of the List interface are as follows:

* HashSet
* TreeSet
* LinkedHashSet
* EnumSet

**61.What is a HashSet ?**

* A HashSet is an unsorted, unordered Set.
* It uses the hashcode of the object being inserted (so the more efficient your hashcode() implementation the better access performance you’ll get).
* Use this class when you want a collection with no duplicates and you don’t care about order when you iterate through it.

**62.What is a TreeSet ?**

TreeSet is a Set implementation that keeps the elements in sorted order. The elements are sorted according to the natural order of elements or by the comparator provided at creation time.

**63.What is an EnumSet ?**

An EnumSet is a specialized set for use with enum types, all of the elements in the EnumSet type that is specified, explicitly or implicitly, when the set is created.

**64.Difference between HashSet and TreeSet ?**

|  |  |
| --- | --- |
| **HashSet** | **TreeSet** |
| HashSet is under set interface i.e. it  does not guarantee for either sorted order or sequence order. | TreeSet is under set i.e. it provides elements in a sorted  order (acceding order). |
| We can add any type of elements to hash set. | We can add only similar types  of elements to tree set. |

|  |
| --- |
|  |

**65.What is a Map ?**

* A map is an object that stores associations between keys and values (key/value pairs).
* Given a key, you can find its value. Both keys  and  values are objects.
* The keys must be unique, but the values may be duplicated.
* Some maps can accept a null key and null values, others cannot.

**66.What are the main Implementations of the Map interface ?**

The main implementations of the List interface are as follows:

* HashMap
* HashTable
* TreeMap
* EnumMap

**67.What is a TreeMap ?**

TreeMap actually implements the SortedMap interface which extends the Map interface. In a TreeMap the data will be sorted in ascending order of keys according to the natural order for the key's class, or by the comparator provided at creation time. TreeMap is based on the Red-Black tree data structure.

**68.How do you decide when to use HashMap and when to use 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.

**69.Difference between HashMap and Hashtable ?**

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| HashMap lets you have null values as well as one null key. | HashTable  does not allows null values as key and value. |
| The iterator in the HashMap is fail-safe (If you change the map while iterating, you’ll know). | The enumerator for the Hashtable is not fail-safe. |
| HashMap is unsynchronized. | Hashtable is synchronized. |

**Note**: Only one NULL is allowed as a key in HashMap. HashMap does not allow multiple keys to be NULL. Nevertheless, it can have multiple NULL values.

**70.How does a Hashtable internally maintain the key-value pairs?**

TreeMap actually implements the SortedMap interface which extends the Map interface. In a TreeMap the data will be sorted in ascending order of keys according to the natural order for the key's class, or by the comparator provided at creation time. TreeMap is based on the Red-Black tree data structure.

|  |
| --- |
|  |

**71.What Are the different Collection Views That Maps Provide?**

Maps Provide Three Collection Views.

* **Key Set**- allow a map's contents to be viewed as a set of keys.
* **Values Collection** - allow a map's contents to be viewed as a set of values.
* **Entry Set** - allow a map's contents to be viewed as a set of key-value mappings.

**72.What is a KeySet View ?**

KeySet is a set returned by the ***keySet()*** method of the Map Interface, It is a set that contains all the keys present in the Map.

**73.What is a Values Collection View ?**

Values Collection View is a collection returned by the ***values()*** method of the Map Interface, It contains all the objects present as values in the map.

**74.What is an EntrySet View ?**

Entry Set view is a set that is returned by the ***entrySet()*** method in the map and contains Objects of type Map. Entry each of which has both Key and Value.

**75.How do you sort an ArrayList (or any list) of user-defined objects ?**

Create an implementation of the *java.lang.Comparable* interface that knows how to order your objects and pass it to *java.util.Collections.sort*(List, Comparator).

**76.What is the Comparable interface ?**

The Comparable interface is used to sort collections and arrays of objects using the Collections.sort() and java.utils.Arrays.sort() methods respectively. The objects of the class implementing the Comparable interface can be ordered.

The Comparable interface in the generic form is written as follows:

interface Comparable<T>

*where T is the name of the type parameter.*  
  
All classes implementing the Comparable interface must implement the compareTo() method that has the return type as an integer. The signature of thecompareTo() method is as follows:

int i = object1.compareTo(object2)

* If object1 < object2: The value of i returned will be negative.
* If object1 > object2: The value of i returned will be positive.
* If object1 = object2: The value of i returned will be zero.

**77.What are the differences between the Comparable and Comparator interfaces ?**

|  |  |
| --- | --- |
| **Comparable** | **Comparato** |
| It uses the *compareTo()* method.  *int objectOne.compareTo(objectTwo).* | t uses the *compare()*method.  *int compare(ObjOne, ObjTwo)* |
| It is necessary to modify the class whose instance is going to be sorted. | A separate class can be created in order to sort the instances. |
| Only one sort sequence can be created. | Many sort sequences can be created. |
| It is frequently used by the API classes. | It used by third-party classes to sort instances. |