**Java Collections FAQ**

|  |
| --- |
|  |

**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. |