****

**Q) How is ArrayList Implemented?**

🡪 Internally an ArrayList uses an Object[].

As you add items to an ArrayList, the list checks to see if the backing array has room left. If there is room, the new item is just added at the next empty space. If there is not room, a new, larger, array is created, and the old array is copied into the new one.

Now, there is more room left, and the new element is added in the next empty space.

/\*\*

\* The array buffer into which the elements of the ArrayList are stored.

\* The capacity of the ArrayList is the length of this array buffer. Any

\* empty ArrayList with elementData == DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA

\* will be expanded to DEFAULT\_CAPACITY when the first element is added.

\*/

**transient** Object[] elementData; // non-private to simplify nested class access

**Q) What is RandomAccess interface?**

🡪 The RandomAccess interface **contains no members**. However, by implementing this interface, a collection **signals** that it supports efficient random access to its elements. Although a collection might support random access, it might not do so efficiently. By checking for the RandomAccess interface, client code can determine at run time whether a collection is suitable for certain types of random access operations—especially as they apply to large collections. (You can use instanceof to determine if a class implements an interface.) RandomAccess is implemented by ArrayList and by the legacy Vector class, among others.

**What is the diff between araylist and vector?**

Major Differences between ArrayList and Vector:

1. **Synchronization** : Vector is synchronized, which means only one thread at a time can access the code, while arrayList is not synchronized, which means multiple threads can work on arrayList at the same time. For example, if one thread is performing an add operation, then there can be another thread performing a remove operation in a multithreading environment.  
   If multiple threads access arrayList concurrently, then we must synchronize the block of the code which modifies the list structurally, or alternatively allow simple element modifications. Structural modification means addition or deletion of element(s) from the list. Setting the value of an existing element is not a structural modification.
2. **Performance**: ArrayList is faster, since it is non-synchronized, while vector operations give slower performance since they are synchronized (thread-safe). If one thread works on a vector, it has acquired a lock on it, which forces any other thread wanting to work on it to have to wait until the lock is released.
3. **Data Growth**: ArrayList and Vector both grow and shrink dynamically to maintain optimal use of storage – but the way they resize is different. **ArrayList** increments **50%** of the current array size if the number of elements exceeds its capacity, while **vector** increments **100%** – essentially doubling the current array size.
4. **Traversal**: **Vector** can use both [**Enumeration and Iterator**](https://www.geeksforgeeks.org/iterators-in-java/) for traversing over elements of vector while **ArrayList** can only use **Iterator** for traversing.

**What is the diff between set and list?**

1) List is an **ordered** collection it maintains the **insertion order**, which means upon displaying the list content it will display the elements in the same order in which they got inserted into the list.

Set is an **unordered** collection, it doesn’t maintain any order. There are few implementations of Set which maintains the order such as LinkedHashSet (It maintains the elements in insertion order).

2) List allows duplicates while Set doesn’t allow duplicate elements. All the elements of a Set should be unique if you try to insert the duplicate element in Set it would replace the existing value.

3) List implementations: ArrayList, LinkedList etc.

Set implementations: HashSet, LinkedHashSet, TreeSet etc.

4) List allows any number of **null** values. Set can have only a single **null** value at most.

5) ListIterator can be used to traverse a List in both the directions (forward and backward) however it cannot be used to traverse a Set. We can use Iterator (It works with List too) to traverse a Set.

6) List interface has one legacy class called Vector whereas Set interface does not have any legacy class.

**What is the diff between Hashmap and Hashtable?**

There are several differences between HashMap and Hashtable in Java:

Hashtable is **synchronized**, whereas HashMap is not. This makes HashMap better for non-threaded applications, as unsynchronized Objects typically perform better than synchronized ones.

Hashtable **does not allow null keys** or values. HashMap allows one null key and any number of null values.

One of HashMap's subclasses is LinkedHashMap, so in the event that you'd want predictable iteration order (which is insertion order by default), you could easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable.

Since synchronization is not an issue for you, I'd recommend HashMap. If synchronization becomes an issue, you may also look at ConcurrentHashMap.

**What is the diff between Collection & Collections?**

**Collection Interface :**

Collection is a root level interface of the Java Collection Framework. Most of the classes in Java Collection Framework inherit from this interface. List, Set and Queue are main sub interfaces of this interface. JDK doesn’t provide any direct implementations of this interface. But, JDK provides direct implementations of its sub interfaces. ArrayList, Vector, HashSet, LinkedHashSet, PriorityQueue are some indirect implementations of Collection interface. Map interface, which is also a part of java collection framework, doesn’t inherit from Collection interface. Collection interface is a member of java.util package.

**Collections Class:**

Collections are an utility class in java.util package. It consists of only static methods which are used to operate on objects of type Collection. For example, it has the method to find the maximum element in a collection, it has the method to sort the collection, it has the method to search for a particular element in a collection

**What is the diff between Treeset and Hashset ?**

HashSet is Implemented using a hash table. Elements are not ordered. The add, remove, and contains methods have constant time complexity O(1).

TreeSet is implemented using a tree structure(red-black tree in algorithm book). The elements in a set are sorted, but the add, remove, and contains methods has time complexity O(log (n)). It offers several methods to deal with the ordered set like first(), last(), headSet(), tailSet(), etc.

1) First major difference between HashSet and TreeSet is performance. HashSet is faster than TreeSet and should be preferred choice if sorting of element is not required.

2) Second difference between HashSet and TreeSet is that HashSet allows null object but TreeSet doesn't allow null Object and throw NullPointerException, Why, because TreeSet uses compareTo() method to compare keys and compareTo() will throw java.lang.NullPointerException.

3) Another significant difference between HashSet and TreeSet is that , HashSet is backed by HashMap while TreeSet is backed by TreeMap in Java.

4) One more difference between HashSet and TreeSet which is worth remembering is that HashSet uses equals() method to compare two object in Set and for detecting duplicates while TreeSet uses compareTo() method for same purpose. if equals() and compareTo() are not consistent, i.e. for two equal object equals should return true while compareTo() should return zero, than it will break contract of Set interface and will allow duplicates in Set implementations like TreeSet

5) Now most important difference between HashSet and TreeSet is ordering. HashSet doesn't guaranteed any order while TreeSet maintains objects in Sorted order defined by either Comparable or Comparator method in Java.

6) TreeSet does not allow to insert Heterogeneous objects. It will throw classCastException at Runtime if trying to add hetrogeneous objects, whereas HashSet allows hetrogeneous objects.

**What is difference between Arraylist and Linked list?**

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

However, there are many differences between ArrayList and LinkedList classes that are given below.



**`What is difference between hashmap, hashtable and concurrent hashmap?**

|  |  |  |  |
| --- | --- | --- | --- |
| Property | HashMap | Hashtable | ConcurrentHashMap |
| Null values/keys | allowed | not allowed | not allowed |
| Is thread-safe | no | yes | yes |
| Lock mechanism | No locking | locks the whole object | locks the bucket |
| Iterator | fail fast | fail fast | weakly consistent |

**Differences Between HashSet, LinkedHashSet and TreeSet In Java :**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **HashSet** | **LinkedHashSet** | **TreeSet** |
| How they work internally? | HashSet uses HashMap internally to store it’s elements. | LinkedHashSet uses  LinkedHashMap internally to store it’s elements. | TreeSet uses TreeMap internally to store it’s elements. |
| Order Of Elements | HashSet doesn’t maintain any order of elements. | LinkedHashSet maintains insertion order of elements. i.e elements are placed as they are inserted. | TreeSet orders the elements according to supplied Comparator. If no comparator is supplied, elements will be placed in their natural ascending order. |
| Performance | HashSet gives better performance than the LinkedHashSet and TreeSet. | The performance of LinkedHashSet is between HashSet and TreeSet. It’s performance is almost similar to HashSet. But slightly in the slower side as it also maintains LinkedList internally to maintain the insertion order of elements. | TreeSet gives less performance than the HashSet and LinkedHashSet as it has to sort the elements after each insertion and removal operations. |
| Insertion, Removal And Retrieval Operations | HashSet gives performance of order O(1) for insertion, removal and retrieval operations. | LinkedHashSet also gives performance of order O(1) for insertion, removal and retrieval operations. | TreeSet gives performance of order O(log(n)) for insertion, removal and retrieval operations. |
| How they compare the elements? | HashSet uses equals() and hashCode() methods to compare the elements and thus removing the possible duplicate elements. | LinkedHashSet also uses equals() and hashCode() methods to compare the elements. | TreeSet uses compare() or compareTo() methods to compare the elements and thus removing the possible duplicate elements. It doesn’t use equals() and hashCode() methods for comparision of elements. |
| Null elements | HashSet allows maximum one null element. | LinkedHashSet also allows maximum one null element. | TreeSet doesn’t allow even a single null element. If you try to insert null element into TreeSet, it throws NullPointerException. |
| Memory Occupation | HashSet requires less memory than LinkedHashSet and TreeSet as it uses only HashMap internally to store its elements. | LinkedHashSet requires more memory than HashSet as it also maintains LinkedList along with HashMap to store its elements. | TreeSet also requires more memory than HashSet as it also maintains Comparator to sort the elements along with the TreeMap. |
| When To Use? | Use HashSet if you don’t want to maintain any order of elements. | Use LinkedHashSet if you want to maintain insertion order of elements. | Use TreeSet if you want to sort the elements according to some Comparator. |