**LIST**

A List is an ordered Collection (sometimes called a sequence). Lists may contain duplicate elements. Elements can be inserted or accessed by their position in the list, using a zero-based index.

* [ArrayList](http://beginnersbook.com/2014/08/arraylist-in-java/)
* [LinkedList](http://beginnersbook.com/2014/08/java-linkedlist-class/)
* [Vector](http://beginnersbook.com/2014/08/java-vector-class/)

**Array List**

ArrayList is a resizable-array implementation of the List interface. It implements all optional list operations, and permits all elements, including null.

**Linked List**

LinkedList is a doubly-linked list implementation of the List and Deque interfaces.

**Vector**

The **Vector class** implements a growable array of objects. Vector implements List Interface. Similar to array, elements of Vector can be accessed using an integer index.

Vector is synchronized which means it is suitable for thread-safe operations but it gives poor performance when used in [**multi-thread environment**](http://beginnersbook.com/2013/03/multithreading-in-java/). It is recommended to use ArrayList (it is non-synchronized, gives good performance) in place of Vector when there is no need of thread-safe operations.

**SET**

A **Set** is a Collection that cannot contain duplicate elements.

* [HashSet](http://beginnersbook.com/2013/12/hashset-class-in-java-with-example/)
* [LinkedHashSet](http://beginnersbook.com/2013/12/linkedhashset-class-in-java-with-example/)
* [TreeSet](http://beginnersbook.com/2013/12/treeset-class-in-java-with-example/)

**HashSet**

**HashSet**, which stores its elements in a hash table, is the best-performing implementation; however it makes no guarantees concerning the order of iteration. This class is not synchronized. However it can be synchronized explicitly like this: Set s = Collections.synchronizedSet(new HashSet (...));

**Points to Note about HashSet:**

1. HashSet doesn’t maintain any order, the elements would be returned in any random order.
2. HashSet doesn’t allow duplicates. If you try to add a duplicate element in HashSet, the old value would be overwritten.
3. HashSet allows null values however if you insert more than one nulls it would still return only one null value.
4. HashSet is non-synchronized.
5. The iterator returned by this class is fail-fast which means iterator would throw ConcurrentModificationException if HashSet has been modified after creation of iterator, by any means except iterator’s own remove method.

**LinkedHashSet**

**LinkedHashSet**, which is implemented as a hash table with a linked list running through it, orders its elements based on the order in which they were inserted into the set (insertion-order).

[LinkedHashSet](http://docs.oracle.com/javase/6/docs/api/java/util/LinkedHashSet.html) is also an implementation of Set interface, it is similar to the HashSet and TreeSet except the below mentioned differences:

1. HashSet doesn’t maintain any kind of order of its elements.
2. TreeSet sorts the elements in ascending order.
3. LinkedHashSet maintains the insertion order. Elements get sorted in the same sequence in which they have been added to the Set.

**TreeSet**

**TreeSet**, which stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashSet.

TreeSet is similar to [**HashSet**](http://beginnersbook.com/2013/12/hashset-class-in-java-with-example/) except that it sorts the elements in the ascending order while HashSet doesn’t maintain any order. TreeSet allows null element but like HashSet it doesn’t allow. Like most of the other collection classes this class is also not synchronized, however it can be synchronized explicitly like this: SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));

**Map**

A Map is an object that maps keys to values. A map cannot contain duplicate keys.

* [HashMap](http://beginnersbook.com/2014/08/java-hashmap-class/)
* [TreeMap](http://beginnersbook.com/2013/12/treemap-in-java-with-example/)
* [LinkedHashMap](http://beginnersbook.com/2013/12/linkedhashmap-in-java/)

**HashMap**

**HashMap**: it makes no guarantees concerning the order of iteration

**HashMap** maintains key and value pairs and often denoted as HashMap<Key, Value> or HashMap<K, V>. HashMap implements Map interface. HashMap is similar to **Hashtable** with two exceptions – HashMap methods are **unsynchornized** and it allows null key and null values unlike **Hashtable**. It is used for maintaining key and value mapping.

It is not an ordered collection which means it does not return the keys and values in the same order in which they have been inserted into the HashMap. It neither does any kind of sorting to the stored keys and Values. You must need to import java.util.HashMap or its super class in order to use the HashMap class and methods.

**TreeMap**

TreeMap: It stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashMap.

LinkedHashMap

LinkedHashMap: It orders its elements based on the order in which they were inserted into the set (insertion-order).