**Concept of collection framework**

# **Java Collections Framework Tutorials**

The Java Collections Framework is a collection of interfaces and classes which helps in storing and processing the data efficiently. This framework has several useful classes which have tons of useful functions which makes a programmer task super easy. I have written several tutorials on Collections and below are the links of those. All the tutorials are shared with examples and source codes to help you understand better.

## **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

* ArrayList
* How to loop ArrayList
* How to get sublist of an ArrayList
* ArrayList Sorting
* Find length of an ArrayList

Arraylist class implements List interface. It is widely used because of the functionality and flexibility it offers. Most of the developers choose Arraylist over Array as it’s a very good alternative of traditional java arrays. ArrayList is a resizable-array implementation of the List interface. It implements all optional list operations, and permits all elements, including null.

The issue with arrays is that they are of fixed length so if it is full you cannot add any more elements to it, likewise if there are number of elements gets removed from it the memory consumption would be the same as it doesn’t shrink. On the other ArrayList can dynamically grow and shrink after addition and removal of elements. Apart from these benefits ArrayList class enables us to use predefined methods of it which makes our task easy.

**e.g**

**import java.util.\*;**

**public class Example1 {**

**public static void main(String args[]) {**

**/\*Creation of ArrayList: I'm going to add String**

**\*elements so I made it of string type \*/**

**ArrayList<String> a1 = new ArrayList<String>();//this way ArrayList is declared**

**/\*This is how elements should be added to the array list\*/**

**a1.add("Somnath");**

**a1.add("Harry");**

**a1.add("Raju");**

**/\* Displaying array list elements \*/**

**for(String s:a1){**

**System.out.println("Currently the array list has following elements:"+s);**

**}**

**/\*Add element at the given index\*/**

**a1.add(0, "Rahul");**

**a1.add(1, "Justin");**

**/\*Remove elements from array list like this\*/**

**a1.remove("Somnath");**

**a1.remove("Harry");**

**System.out.println("Current array list is:"+a1);**

**/\*Remove element from the given index\*/**

**a1.remove(1);**

**System.out.println("Current array list is:"+a1);//displaying all elements at one go**

**}**

**}**

#### LinkedList

* LinkedList
* Loop LinkedList
* Search elements in LinkedList
* Append elements of a list to LinkedList
* push() and pop() in LinkedList
* Iterate a LinkedList in Reverse order

LinkedList is a doubly-linked list implementation of the List and Deque interfaces. LinkedList allows for constant-time insertions or removals using iterators, but only sequential access of elements. In other words, LinkedList can be searched forward and backward but the time it takes to traverse the list is directly proportional to the size of the list.

#### Vector

* Vector
* Vector Enumeration Example
* Sort Vector
* Vector ListIterator example

Vector implements List Interface. Like ArrayList it also maintains insertion order but it is rarely used in non-thread environment as it is synchronized and due to which it gives poor performance in searching, adding, delete and update of its elements.

## Set

A Set is a Collection that cannot contain duplicate elements. There are three main implementations of Set interface: HashSet, TreeSet, and LinkedHashSet. HashSet, which stores its elements in a hash table, is the best-performing implementation; however it makes no guarantees concerning the order of iteration. TreeSet, which stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashSet. 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).

#### HashSet

* HashSet
* Iterate HashSet
* HashSet Sorting
* Converting HashSet to a TreeSet

This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element. This class is not synchronized.

**import java.util.HashSet;**

**public class HashSetExample {**

**public static void main(String args[]) {**

**// HashSet declaration**

**HashSet<String> hset =**

**new HashSet<String>();holding String type value**

**// Adding elements to the HashSet**

**hset.add("Somnath");**

**hset.add("Jordon");**

**hset.add("Hardy");**

**//Addition of duplicate element**

**hset.add("Jordon");**

**//Displaying HashSet elements**

**System.out.println(hset);**

**}**

**}**

**System.out.println("HashSet contains: ");**

**for(String temp : hset){**

**System.out.println(temp);**

**}//it will show no duplicate value as two Jordon wont be seen**

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
* List Vs Set

LinkedHashSet 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 gets sorted in the same sequence in which they have been added to the Set.

**List Vs Set**

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 can not 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.

#### TreeSet

* TreeSet
* HashSet vs TreeSet

TreeSet is similar to HashSet 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(...));

**HashSet vs TreeSet**

1) HashSet gives better performance (faster) than TreeSet for the operations like add, remove, contains, size etc. HashSet offers constant time cost while TreeSet offers log(n) time cost for such operations.

2) HashSet does not maintain any order of elements while TreeSet elements are sorted in ascending order by default.

1) Both HashSet and TreeSet does not hold duplicate elements, which means both of these are duplicate free.

2) If you want a sorted Set then it is better to add elements to HashSet and then convert it into TreeSet rather than creating a TreeSet and adding elements to it.

3) Both of these classes are non-synchronized that means they are not thread-safe and should be synchronized explicitly when there is a need of thread-safe operations.

## Map

A Map is an object that maps keys to values. A map cannot contain duplicate keys. There are three main implementations of Map interfaces: HashMap, TreeMap, and LinkedHashMap.  
HashMap: it makes no guarantees concerning the order of iteration  
TreeMap: It stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashMap.  
LinkedHashMap: It orders its elements based on the order in which they were inserted into the set (insertion-order).

#### HashMap

* HashMap
* HashMap Iterator example
* Sort HashMap by Keys and Values
* Check if particular key exists in HashMap
* Check if particular value exists in HashMap

**HashMap** is a Map based collection class that is used for storing Key & value pairs, it is denoted as HashMap<Key, Value> or HashMap<K, V>. This class makes no guarantees as to the order of the map. It is similar to the Hashtable class except that it is unsynchronized and permits nulls(null values and null key).

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 does not sort 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.

**import java.util.HashMap;**

**import java.util.Map;**

**import java.util.Iterator;**

**import java.util.Set;**

**public class Details {**

**public static void main(String args[]) {**

**/\* This is how to declare HashMap \*/**

**HashMap<Integer, String> hmap = new HashMap<Integer, String>();**

**/\*Adding elements to HashMap\*/**

**hmap.put(12, "Somnath");**

**hmap.put(2, "Jordon");**

**hmap.put(7, "Hardy");**

**/\* Display content using Iterator\*/**

**Set set = hmap.entrySet();**

**Iterator iterator = set.iterator();**

**while(iterator.hasNext()) {**

**Map.Entry mentry = (Map.Entry)iterator.next();**

**System.out.print("key is: "+ mentry.getKey() + " & Value is: ");**

**System.out.println(mentry.getValue());**

**}**

**/\* Get values based on key\*/**

**String var= hmap.get(2);**

**System.out.println("Value at index 2 is: "+var);**

**/\* Remove values based on key\*/**

**hmap.remove(7);**

**System.out.println("Map key and values after removal:");**

**Set set2 = hmap.entrySet();**

**Iterator iterator2 = set2.iterator();**

**while(iterator2.hasNext()) {**

**Map.Entry mentry2 = (Map.Entry)iterator2.next();**

**System.out.print("Key is: "+mentry2.getKey() + " & Value is: ");**

**System.out.println(mentry2.getValue());**

**}**

**}**

**}**

#### TreeMap

* TreeMap
* Iterate TreeMap
* Sort TreeMap
* Iterate TreeMap in Reverse order
* Get Sub Map from TreeMap

TreeMap is Red-Black tree based NavigableMap implementation. It is sorted according to the natural ordering of its keys.  
TreeMap class implements Map interface similar to HashMap class. The main difference between them is that HashMap is an unordered collection while TreeMap is sorted in the ascending order of its keys. TreeMap is unsynchronized collection class which means it is not suitable for thread-safe operations until unless synchronized explicitly.

#### LinkedHashMap

LinkedHashMap is a Hash table and linked list implementation of the Map interface, with predictable iteration order. This implementation differs from HashMap in that it maintains a doubly-linked list running through all of its entries. This linked list defines the iteration ordering, which is normally the order in which keys were inserted into the map (insertion-order). In last few tutorials we have discussed about HashMap and TreeMap. This class is different from both of them:

* HashMap doesn’t maintain any order.
* TreeMap sort the entries in ascending order of keys.
* LinkedHashMap maintains the insertion order.

## Iterator/ListIterator

Both Iterator and ListIterator are used to iterate through elements of a collection class. Using Iterator we can traverse in one direction (forward) while using ListIterator we can traverse the collection class on both the directions(backward and forward).

* Iterator
* ListIterator