**

**Softech** **Solutions Inc.**

www.softechsolutionsgroup.com

[saney.alam@ softechsolutionsgroup.com](mailto:saney.alam@techsolutions.tech)

Core Java - Collections

***Lecture Notes***

# A Complete Guide for Beginners

Core Java - Collections

|  |  |
| --- | --- |
| ***Table of Contents Page*** | |
| ***Collection Framework*** | ***02*** |
| ***ArrayList*** | ***02*** |
|  | ***04*** |
|  | ***05*** |
|  | ***07*** |
|  | ***08*** |
|  |  |

## **Collections Framework**

The [Java Collections Framework](http://docs.oracle.com/javase/tutorial/collections/intro/index.html) 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.

## **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/)

## **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).

## **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).

## **ArrayList**

ArrayList is a class which 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.

The issue with arrays is that they are of fixed length so if it is full we 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 as per the need. Apart from these benefits ArrayList class enables us to use predefined methods of it which makes our task easy.

**Example**

import java.util.\*;

public class ArrayListExample {

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> obj = new ArrayList<String>();

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

obj.add("Clark");

obj.add("Harry");

obj.add("Steve");

obj.add("Gates");

obj.add("Sundar");

/\* Displaying array list elements \*/

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

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

obj.add(0, "Rahul");

obj.add(1, "Justin");

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

obj.remove("Steve");

obj.remove("Harry");

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

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

obj.remove(1);

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

}

}

**Output**

Currently the array list has following elements:[Clark, Harry, Steve, Gates, Sundar]

Current array list is:[Rahul, Justin, Clark, Gates, Gates, Sundar]

Current array list is:[Rahul, Clark, Gates, Gates, Sundar]

## **Methods of ArrayList class**

In the above example we have used methods such as add and remove. However there are numbers of methods available which can be used directly using object of ArrayList class. Let’s discuss few of the important methods.

1) **add( Object o)**: This method adds an object o to the ArrayList.

obj.add("hello");

This statement would add a string hello in the ArrayList at last position.

2) **add(int index, Object o)**: It adds the object o to the array list at the given index.

obj.add(2, "bye");

It will add the string bye to the 2nd index (3rd position as the array list starts with index 0) of array list.

3) **remove(Object o)**: Removes the object o from the ArrayList.

obj.remove("Chaitanya");

This statement will remove the string “Chaitanya” from the ArrayList.

4) **remove(int index)**: Removes element from a given index.

obj.remove(3);

It would remove the element of index 3 (4th element of the list – List starts with o).

5) **set(int index, Object o)**: Used for updating an element. It replaces the element present at the specified index with the object o.

obj.set(2, "Tom");

It would replace the 3rd element (index =2 is 3rd element) with the value Tom.

6)**int indexOf(Object o)**: Gives the index of the object o. If the element is not found in the list then this method returns the value -1.

int pos = obj.indexOf("Tom");

This would give the index (position) of the string Tom in the list.

7) **Object get(int index)**: It returns the object of list which is present at the specified index.

String str= obj.get(2);

Function get would return the string stored at 3rd position (index 2) and would be assigned to the string “str”. We have stored the returned value in string variable because in our example we have defined the ArrayList is of String type. If you are having integer array list then the returned value should be stored in an integer variable.

8) **int size()**: It gives the size of the ArrayList – Number of elements of the list.

int numberofitems = obj.size();

9) **boolean contains(Object o)**: It checks whether the given object o is present in the array list if its there then it returns true else it returns false.

obj.contains("Steve");

It would return true if the string “Steve” is present in the list else we would get false.

10) **clear():** It is used for removing all the elements of the array list in one go. The below code will remove all the elements of ArrayList whose object is obj.

obj.clear();

## **LinkedList**

## LinkedList is an implementation of List interface. Earlier we learnt about [ArrayList class](http://beginnersbook.com/2013/12/java-arraylist/) which also implements List Interface.

**Example**

import java.util.\*;

public class LinkedListExample {

public static void main(String args[]) {

/\* Linked List Declaration \*/

LinkedList<String> linkedlist = new LinkedList<String>();

/\*add(String Element) is used for adding

\* the elements to the linked list\*/

linkedlist.add("Item1");

linkedlist.add("Item5");

linkedlist.add("Item3");

linkedlist.add("Item6");

linkedlist.add("Item2");

/\*Display Linked List Content\*/

System.out.println("Linked List Content: " +linkedlist);

/\*Add First and Last Element\*/

linkedlist.addFirst("First Item");

linkedlist.addLast("Last Item");

System.out.println("LinkedList Content after addition: " +linkedlist);

/\*This is how to get and set Values\*/

Object firstvar = linkedlist.get(0);

System.out.println("First element: " +firstvar);

linkedlist.set(0, "Changed first item");

Object firstvar2 = linkedlist.get(0);

System.out.println("First element after update by set method: " +firstvar2);

/\*Remove first and last element\*/

linkedlist.removeFirst();

linkedlist.removeLast();

System.out.println("LinkedList after deletion of first and last element: " +linkedlist);

/\* Add to a Position and remove from a position\*/

linkedlist.add(0, "Newly added item");

linkedlist.remove(2);

System.out.println("Final Content: " +linkedlist);

}

}

**Output:**

Linked List Content: [Item1, Item5, Item3, Item6, Item2]

LinkedList Content after addition: [First Item, Item1, Item5, Item3, Item6, Item2, Last Item]

First element: First Item

First element after update by set method: Changed first item

LinkedList after deletion of first and last element: [Item1, Item5, Item3, Item6, Item2]

Final Content: [Newly added item, Item1, Item3, Item6, Item2]