Java Collection – List Interface

By Umesh Sir

Contact us 7758094241

List

- ★ What is List Interface?
- ★ ArrayList Class.
- ★ Generic VS Non-Generic?
- ★ Collection Differences VS Different JDK Versions?
- ★ Vector Class
- ★ LinkedList Class
- ★ Difference between ArrayList and Vector
- ★ Difference between Arraylist and linked list

★ ArrayList Class

- ★ ArrayList is same like array which can grows dynamically in memory.
- ★ It uses dynamic array for storing the elements.
- ★ It extends AbstractList Class and implements List Interface.
- ★ It is not synchronized.
- ★ Initial Capacity of ArrayList is 10 and it increases it's capacity by 50%.
- ★ ArrayList loadFactor = 50%
- ★ ArrayList Methods : add() and get() etc.

★ Generic VS Non-Generic?

★ Non-generic before JDK 1.5. After 1.5 it is Generic

★ Generic Collection allows you to have only one type of object in collection. Now it is type safe type casting is not required at run time.

e.g -Non-Generic

```
ArrayList al = new ArrayList();
```

e.g -Generic

```
ArrayList<String> al = new ArrayList<String>();
```

★ Collection Differences VS Different JDK Versions?

★ 1] JDK 1.4 and Below

ArrayList al = new ArrayList();

Boxing -Converting Primitive to Object

Integer i = new Integer(4);

al.add(i);

For retrieving element need unboxing.

Unboxing -Converting from object to primitive value.

Integer iv = (Integer)al.get(0);

Int val = iv.intValue(); //It will give primitive value

★ Collection Differences VS Different JDK Versions?

★ 1] JDK 1.5 and above

Generics were Introduced So, No need of boxing and Unboxing after JDK 1.5.

```
ArrayList<Integer> al = new ArrayList<Integer>();

al.add(5); // new Integer(5); It calls Internally Autoboxing
```

- Autoboxing -Automatically convert primitive to Object.

For retrieving element No need of Unboxing. It automatically convert Object to Primitive.

```
int val = al.get(0);
```

★ Vector Class

- ★ Vector is similar to ArrayList but vector synchronised by default.
- ★ The default capacity is 10 but it increased by double every time.
- ★ Vector is legacy class means it comes with JDK version.

```
Vector v = new Vector();
        v.add(1);
        v.add(2);
```

★ LinkedList Class

★ LinkedList contains a group of elements in the form of nodes. It has three fields.

Address/link	Data	Address/link

★ LinkedList is used to store and retrieve data. Insertion and deletion of elements in anywhere.

```
LinkedList l= new LinkedList();
        l.add(1);
        l.add(2);
```

★ Difference between ArrayList and Vector

ArrayList	Vector
1) ArrayList is **not synchronized**.	Vector is **synchronized**.
2) ArrayList **increments 50%** of current array size if the number of elements exceeds from its capacity.	Vector **increments 100%** means doubles the array size if the total number of elements exceeds than its capacity.
3) ArrayList is **not a legacy** class. It is introduced in JDK 1.2.	Vector is a **legacy** class.
4) ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object.
5) ArrayList uses the **Iterator** interface to traverse the elements.	A Vector can use the **Iterator** interface or **Enumeration** interface to traverse the elements.

★ Difference between Arraylist and linked list

ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) An ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.
5) The memory location for the elements of an ArrayList is contiguous.	The location for the elements of a linked list is not contagious.
6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList.	There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized.
7) To be precise, an ArrayList is a resizable array.	LinkedList implements the doubly linked list of the list interface.

★ Difference between Arraylist and linked list

★ Points to Remember :- The following are some important points to remember regarding an ArrayList and LinkedList.

- When the rate of addition or removal rate is more than the read scenarios, then go for the LinkedList. On the other hand, when the frequency of the read scenarios is more than the addition or removal rate, then ArrayList takes precedence over LinkedList.

- Memory overhead in the LinkedList is more as compared to the ArrayList. It is because, in a LinkedList, we have two extra links (next and previous) as it is required to store the address of the previous and the next nodes, and these links consume extra space.Such links are not present in an ArrayList.

Thank you