**List:**

**List:** is an Interface -- is an **ordered(sequence) collection**, List **may contain duplicates**. Allows null values. Elements are accessed by their position. using zero-based index.

There are **three main implementations** of **List Interface, ArrayList, LinkedList** and **Vector.**

**ArrayList :**

**Initialize an ArrayList:**

**Method 1: Normal way of ArrayList initialozation**

ArrayList<String> books = new ArrayList<String>();

books.add("Java Book1");

books.add("Java Book2");

books.add("Java Book3");

**Method 2: initialization using Arrays.asList**

ArrayList<String> names = new ArrayList<String>( Arrays.asList("Pratap", "Peter", "Harsh"));

**Method 3: Anonymous inner class method to initialize ArrayList**

ArrayList<String> cities = new ArrayList<String>()

{{ add("Delhi"); add("Agra"); add("Chennai"); }};

**Method 4: Use Collections.ncopies [**to initialize ArrayList with same value of elements]

ArrayList<Integer> intlist = new ArrayList<Integer>

(Collections.nCopies(10, 5));

**ArrayList Methods:**

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

al.add(8);

al.add(39);

al.add(44);

al.add(2)

1. **add(element) method –** to add to ArrayList

**al.add(element);**

1. **add(int index, object element) –** to add element at a given index

**al.add(3,49);**

1. **int size() –** to find the size of ArrayList

**int i= al.size();**

1. **Collections.sort(arralist) –** to sort the ArrayList [ strings can also be sorted ]

**Collections.sort(al);**

1. **Collections.reverseOrder()) –** to sort in descending order [strings can also be sorted]

**Collections.sort(arraylist, Collections.reverseOrder());**

1. **addAll(Collection c)-** to add all elements from one ArrayList to another ArrayList

**al2.addAll(al1);**

1. **addAll(int index, Colllection c) -** to add all elements from ArrayList at given index

**al2.zddAll(2,al1);**

1. **remove(element) –** to remove element from given index.

**al1.remove(3);**

1. **remove(int index ) –** to remove element from given index.

**al1.remove(3);**

1. **get(int index) –** fetching and element form the list of given index.

**al1.get(2)**

1. **int indexOf(Object ob1) –** returns index of the given element.

**al1.indexOf(4);**

1. **int lastIndexOf(Object ob1) –** returns index last occurrence of element

**al1.lastIndexOf(4);**

1. **boolean contains(Object ob1) –** returns true if contains and false if not present.

**al1.contains(49);**

**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.

LinkedList can be searched forward and backward but the time it takes to traverse the list I directly proportional to the size of the list. [ LinkedList methods are almost same except below methods]

/\* 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");

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

/\*Add First and Last Element\*/

linkedlist.addFirst("First Item");

linkedlist.addLast("Last Item");

/\*Remove first and last element\*/

linkedlist.removeFirst();

linkedlist.removeLast();

/\*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);

**Vector:** implements List interface. Like ArrayList it also maintains insertion order, but it is rarely used in non-thread environment as it is synchronized. Poor performance in searching, adding, deleting etc.

/\* Vector of initial capacity(size) of 2 \*/

Vector<String> vec = new Vector<String>(2);

/\* Adding elements to a vector\*/

vec.addElement("Apple");

vec.addElement("Orange");

vec.addElement("Mango");

vec.addElement("Fig");