4. List Interface

List interface is a child interface of the collection interface. The classes which need ordered data implements this interface, and ArrayList, Vector, Stack etc are the classes that implement this interface. This also allows duplicate data to be present in it.

Since all the subclasses implement the list, we can instantiate a list object with any of these classes. For example,

```
List <T> al = new ArrayList<> ();
List <T> ll = new LinkedList<> ();
List <T> v = new Vector<> ();

Where T is the type of the Object
```

Here, we will be discussing only ArrayList, and we will learn more about other classes after learning data structures.

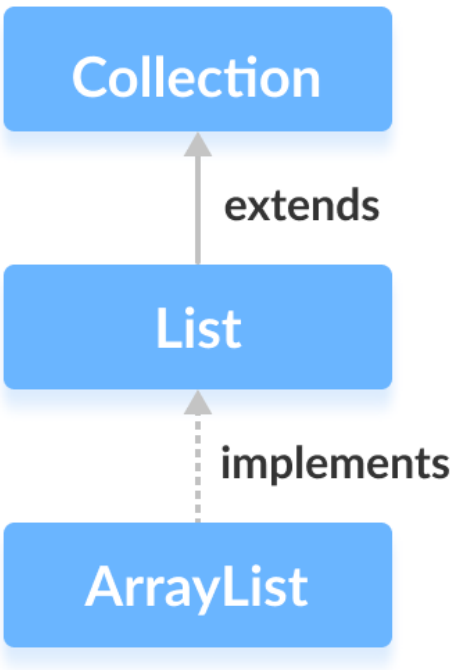
ArrayList Class

The ArrayList class is a resizable array, which can be found in java.util package.

The major difference between an ArrayList and an array is that the size of an array cannot be modified and if you want to add elements to an array you have to create a new one and copy all the old elements to the new array. While elements can be added and removed from an ArrayList whenever you want i.e it grows and shrink dynamically.

The ArrayList in Java can also have duplicate elements like the arrays. It implements the List interface so we can use all the methods of the List interface here. The ArrayList maintains the insertion order internally.

It inherits the AbstractList class and implements the List Interface.



The important points about the Java ArrayList class are:

- it can contain duplicate elements.
- The insertion order is preserved.
- ArrayList class is non-synchronized.
- Just like an array random access at any index is allowed.

Constructors of ArrayList

Constructor	Description
`ArrayList()`	It is used to build an empty array list.
`ArrayList(int`	It is used to build an array list that has the specified initial

capacity)	capacity.
ArrayList(Collection<? extends E> c)	It is used to build an array list that is initialized with the elements of the collection c.

Example to show methods and constructor of ArrayList

```
// Java program to demonstrate the
// working of ArrayList in Java

import java.io.*;
import java.util.ArrayList; // import the ArrayList class

class ArrayListExample {
	public static void main(String[] args)
	{
		// Declaring the ArrayList with
		ArrayList<Integer> list = new ArrayList<Integer>(); // used
default constructor

		// Add Items
		// To add an element in the ArrayList, use the add() method
		list.add(3);
	list.add(2);
	list.add(1);
		list.add(4);
		System.out.println(list);

		// Access an Item
	  // To access an element in the ArrayList, use the get() method and
refer to the index number:
	  int ele1 = list.get(0);
	  int ele2 = list.get(1);
	  System.out.println("First element is: " + ele1);
	  System.out.println("Second element is: " + ele2);

	  // Change an Item
	  // To modify an element, use the set() method and refer to the
index number:
	  list.set(0, 100); // set 100 at 0 index
	  System.out.println(list);

	  // Remove an Item
	  // To remove an element, use the remove() method and refer to the
index number:
```

```
        list.remove(2);
        System.out.println(list);

        // ArrayList Size
        // To find out how many elements an ArrayList have, use the size
method:
        int size = list.size();
        System.out.println("Size of list is:"+size);

        //Loop Through an ArrayList
        //Loop through the elements of an ArrayList with a for loop,
        //and use the size() method to specify how many times the loop
should run:
        for(int i = 0 ; i < list.size() ; i++) {
            System.out.println("Element at index " + i + " :" +
list.get(i));
        }

        // To remove all the elements in the ArrayList, use the clear()
method:
        list.clear();
        System.out.println(list);
      }
}


Output:
[3, 2, 1, 4]
First element is: 3
Second element is: 2
[100, 2, 1, 4]
[100, 2, 4]
Size of list is:3
Element at index 0 :100
Element at index 1 :2
Element at index 2 :4
[]
```

How does ArrayList work internally?

Since ArrayList is a dynamic array and we do not have to specify the size while creating it, the size of the array automatically increases when we dynamically add and remove items. Though the actual library implementation may be more complex, the following is a very basic idea explaining the working of the arraylist when the arraylist becomes full and if we try to add an item:

- Creates a bigger-sized memory on heap memory (for example memory of double size).

- Copies the current memory elements to the new memory.
- The new item is added now as there is bigger memory available now.
- Delete the old memory.

Methods of ArrayList

Methods	Description
`void add(int index, E element)`	Inserts the specified element at the specified position in this list.
`boolean add(E e)`	Appends the specified element to the end of this list.
`boolean addAll(Collection<? extends E> c)`	Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator.
`boolean addAll(int index, Collection<? extends E> c)`	Inserts all of the elements in the specified collection into this list, starting at the specified position.
`void clear()`	Removes all of the elements from this list.

void ensureCapacity(int requiredCapacity)	It is used to enhance the capacity of an ArrayList instance.
E get(int index)	Returns the element at the specified position in this list.
boolean isEmpty()	Returns true if this list contains no elements.
int lastIndexOf(Object o)	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
Object[] toArray()	Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[] toArray(T[] a)	Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.

Object clone()	Returns a shallow copy of this ArrayList instance.
Boolean contains(Object o)	It returns true if the list contains the specified element
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
E remove(int index)	It is used to remove the element present at the specified position in the list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element.
boolean removeAll(Collection<?> c)	It is used to remove all the elements from the list.
boolean removeIf(Predicate<?	It is used to remove all the elements from the list that

super E> filter)	satisfies the given predicate.
protected void removeRange(int fromIndex, int toIndex)	It is used to remove all the elements that lies within the given range.
void replaceAll(UnaryOperator<E> operator)	It is used to replace all the elements from the list with the specified element.
void retainAll(Collection<?> c)	It is used to retain all the elements in the list that are present in the specified collection.
E set(int index, E element)	It is used to replace the specified element in the list, present at the specified position.
void sort(Comparator<? super E> c)	It is used to sort the elements of the list on the basis of the specified comparator.

Spliterator<E> spliterator()	It is used to create spliterator over the elements in a list.
List<E> subList(int fromIndex, int toIndex)	Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.
int size()	Returns the number of elements present in the list.
void trimToSize()	Trims the capacity of this ArrayList instance to be the list's current size.