ArrayList in Java

Last Updated : 03 Jan, 2025

Java ArrayList is a part of **collections framework** and it is a class of java.util package. It provides us with dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in array is required. The main **advantage of ArrayList** is, unlike normal arrays, we don't need to mention the size when creating ArrayList. It automatically adjusts its capacity as elements are added or removed.

Example:

```
// Java Program to demonstrate ArrayList
import java.util.ArrayList;

class Main {
    public static void main (String[] args) {

          // Creating an ArrayList
          ArrayList<Integer> a = new ArrayList<Integer>();

          // Adding Element in ArrayList
          a.add(1);
          a.add(2);
          a.add(3);

          // Printing ArrayList
          System.out.println(a);

    }
}
```

Output

```
[1, 2, 3]
```

Table of Content

ArrayList is a Java class implemented using the List interface. Java ArrayList, as the name suggests, provides the functionality of a dynamic array where the size is not fixed as an array. Also, as a part of Collections framework, it has many features not available with arrays.

Syntax of ArrayList

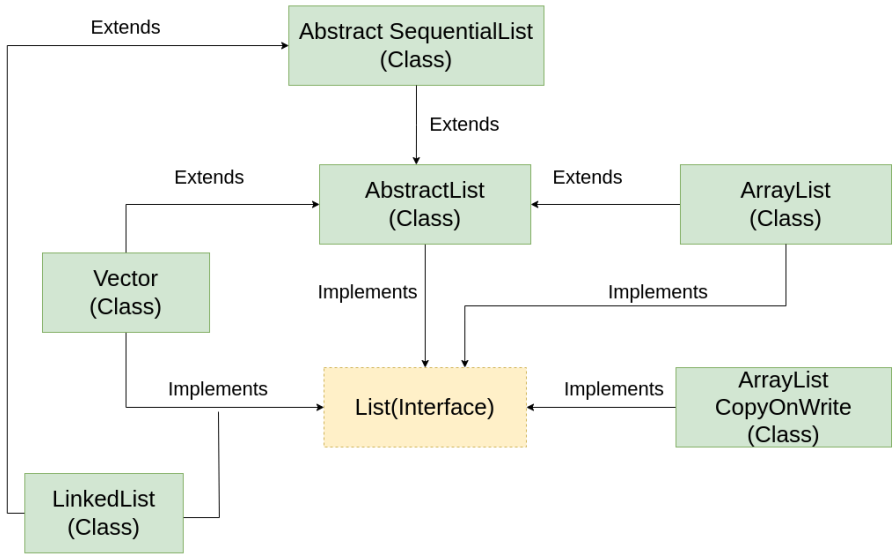
```
ArrayList<Integer> arr = new ArrayList<Integer>();
```

Note: You can also create a generic ArrayList

Important Features of ArrayList in Java

- ArrayList inherits AbstractList class and implements the List interface.
- ArrayList is initialized by size. However, the size is increased automatically if the collection grows or shrinks if the objects are removed from the collection.
- Java ArrayList allows us to randomly access the list.
- ArrayList can not be used for primitive types, like int, char, etc. We need a wrapper class for such cases.
- ArrayList in Java can be seen as a vector in C++.
- ArrayList is not Synchronized. Its equivalent synchronized class in Java is Vector.

Let's understand the **Java ArrayList in depth**. Look at the below image:



In the above illustration, AbstractList, CopyOnWriteArrayList, and AbstractSequentialList are the classes that implement the list interface. A separate functionality is implemented in each of the mentioned classes. They are:

1. **AbstractList:** This class is used to implement an unmodifiable list, for which one needs to only extend this AbstractList Class and implement only the *get()* and the *size()* methods.
2. **CopyOnWriteArrayList:** This class implements the list interface. It is an enhanced version of ArrayList in which all the modifications(add, set, remove, etc.) are implemented by making a fresh copy of the list.
3. **AbstractSequentialList:** This class implements the Collection interface and the AbstractCollection class. This class is used to implement an unmodifiable list, for which one needs to only extend this AbstractList Class and implement only the *get()* and the *size()* methods.

Constructors in ArrayList in Java

In order to Create an ArrayList, we need to create an object of the ArrayList class. The ArrayList class consists of various constructors which allow the possible creation of the array list. The following are the constructors available in this class:

Constructor	Description	Initialize and Declare ArrayList
ArrayList()	This constructor is used to build an empty array list.	ArrayList arr = new ArrayList();
ArrayList(Collection c)	This constructor is used to build an array list initialized with the elements from the collection c.	ArrayList arr = new ArrayList(c);
ArrayList(int capacity)	This constructor is used to build an array list with the initial capacity being specified.	ArrayList arr = new ArrayList(N);

Operations in ArrayList

Now, Using the constructors we have got ArrayList for further operations like Insertion , Deletion and Updation of the elements in ArrayList.

```
// Java Program Example to Demonstrate
// Addition, Deletion and Updation of Element
import java.util.*;

class Main {
    public static void main(String args[]){

        // Creating an Array of string type
        ArrayList<String> al = new ArrayList<>();

          // 1. Addition

        // Adding elements to ArrayList
        // at the end
        al.add("Geeks");
        al.add("Geeks");

          System.out.println("Orignal List : "+al);

          // Adding Elements at the specific
          // index
```

```
        al.add(1, "For");

          System.out.println("After Adding element at index 1 : "+ al);

          // 2. Deletion of Element

          // Removing Element using index
          al.remove(0);

          System.out.println("Element removed from index 0 : "+ al);

          // Removing Element using the value
          al.remove("Geeks");

          System.out.println("Element Geeks removed : "+ al);

          // 3. Updating Values

          // Updating value at index 0
          al.set(0, "GFG");


        // Printing all the elements in an ArrayList
        System.out.println("List after updation of value : "+al);
    }
}
```

Output

```
Orignal List : [Geeks, Geeks]
After Adding element at index 1 : [Geeks, For, Geeks]
Element removed from index 0 : [For, Geeks]
Element Geeks removed : [For]
List after updation of value : [GFG]
```

Let us understand how the three operations performed in above Program works.

1. Adding Elements in ArrayList

Adding elements seems bit complex when the size of ArrayList is not defined:

- 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.

Set initial capacity when possible: *Each time the list exceeds its capacity, it resizes by 50%. This resizing can be costly.*
Avoid Frequent Resizing: *Each resize involves creating a new array and copying all existing elements to it.*

2. Changing Elements in ArrayList

After adding the elements, if we wish to change the element, it can be done using the set() method. Since an ArrayList is indexed, the element which we wish to change is referenced by the index of the element. Therefore, this method takes an index and the updated element which needs to be inserted at that index.

3. Removing Elements in ArrayList

In order to remove an element from an ArrayList, we can use the remove() method. This method is overloaded to perform multiple operations based on different parameters.

Java ArrayList Methods

Method	Description
add(int index, Object element)	This method is used to insert a specific element at a specific position index in a list.
add(Object o)	This method is used to append a specific element to the end of a list.
addAll(Collection C)	This method is used to append all the elements from a specific collection to the end of the mentioned list, in such an order that the values are returned by the specified collection's iterator.
addAll(int index, Collection C)	Used to insert all of the elements starting at the specified position from a specific collection into the mentioned list.

Method	Description
clear()	This method is used to remove all the elements from any list.
clone()	This method is used to return a shallow copy of an ArrayList in Java.
contains(Object o)	Returns true if this list contains the specified element.
ensureCapacity(int minCapacity)	Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
forEach(Consumer<? super E> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
get(int index)	Returns the element at the specified position in this list.
indexOf(Object O)	The index the first occurrence of a specific element is either returned or -1 in case the element is not in the list.
isEmpty()	Returns true if this list contains no elements.
lastIndexOf(Object O)	The index of the last occurrence of a specific element is either returned or -1 in case the element is not in the list.
listIterator()	Returns a list iterator over the elements in this list (in proper sequence).
listIterator(int index)	Returns a list iterator over the elements in this list (in proper sequence), starting at the specified

Method	Description
	position in the list.
remove(int index)	Removes the element at the specified position in this list.
remove(Object o)	Removes the first occurrence of the specified element from this list, if it is present.
removeAll(Collection c)	Removes from this list all of its elements that are contained in the specified collection.
removeIf(Predicate filter)	Removes all of the elements of this collection that satisfy the given predicate.
removeRange(int fromIndex, int toIndex)	Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive.
retainAll(Collection<?> c)	Retains only the elements in this list that are contained in the specified collection.
set(int index, E element)	Replaces the element at the specified position in this list with the specified element.
size()	Returns the number of elements in this list.
spliterator?()	Creates a late-binding and fail-fast Spliterator over the elements in this list.
subList(int fromIndex, int toIndex)	Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.
toArray()	This method is used to return an array containing all of the elements in the list in the correct order.

Method	Description
toArray(Object[] O)	It is also used to return an array containing all of the elements in this list in the correct order same as the previous method.
trimToSize()	This method is used to trim the capacity of the instance of the ArrayList to the list's current size.

Some Key Points of ArrayList in Java

1. ArrayList is Underlined data Structure Resizable Array or Growable Array.
2. ArrayList Duplicates Are Allowed.
3. Insertion Order is Preserved.
4. Heterogeneous objects are allowed.
5. Null insertion is possible.

Complexity of Java ArrayList

Operation	Time Complexity	Space Complexity
Inserting Element in ArrayList	O(1)	O(N)
Removing Element from ArrayList	O(N)	O(1)
Traversing Elements in ArrayList	O(N)	O(N)
Replacing Elements in ArrayList	O(1)	O(1)

Below are the advantages and disadvantages of using ArrayList in Java:

Advantages of Java ArrayList

- **Dynamic size:** ArrayList can dynamically grow and shrink in size, making it easy to add or remove elements as needed.
- **Easy to use**: ArrayList is simple to use, making it a popular choice for many Java developers.
- **Fast access**: ArrayList provides fast access to elements, as it is implemented as an array under the hood.
- **Ordered collection**: ArrayList preserves the order of elements, allowing you to access elements in the order they were added.
- **Supports null values**: ArrayList can store null values, making it useful in cases where the absence of a value needs to be represented.

Disadvantages of Java ArrayList

- **Slower than arrays**: ArrayList is slower than arrays for certain operations, such as inserting elements in the middle of the list.
- **Increased memory usage**: ArrayList requires more memory than arrays, as it needs to maintain its dynamic size and handle resizing.
- **Not thread-safe:** ArrayList is not thread-safe, meaning that multiple threads may access and modify the list concurrently, leading to potential race conditions and data corruption.
- **Performance degradation**: ArrayList's performance may degrade as the number of elements in the list increases, especially for operations such as searching for elements or inserting elements in the middle of the list.

FAQs of ArrayList in Java

How is ArrayList different from an Array in Java?

An `ArrayList` can resize dynamically, while a traditional array has a fixed size. `ArrayList` also provides many useful methods like `add()`, `remove()`, and `size()`.

How to Access elements in an ArrayList?

Elements can be accessed using the `get()` method:
String element = list.get(0);

How to Remove an element from an ArrayList?

Use the `remove()` method to remove elements by index:
list.remove(0);

Is ArrayList Synchronized?

No, `ArrayList` is not synchronized. Use `Collections.synchronizedList(new ArrayList<>())` for thread-safe operations.

Can we Store null elements in an ArrayList?

Yes, `ArrayList` can store `null` elements.

How is data stored in ArrayList?

ArrayList can store data till the ArrayList size is full, after that the size of ArrayList is doubled if we want to store any more elements.