**ArrayList**

is same as **dynamic arrays** with the ability to **resize** itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container.

* ArrayList elements are placed in contiguous storage so that they can be accessed and traversed using **[iterators](https://www.geeksforgeeks.org/iterators-in-java/)**.
* In ArrayList, data is inserted at the end or at specific index.
* Inserting at the end takes differential time, as sometimes there may be a need of extending the array.
* Removing the last element takes only constant time because no resizing happens.
* Inserting and erasing at the beginning or in the middle is linear in time.

### Functions to be implemented in the Dynamic array class:

Certain functions associated with the ArrayList that we will implement are:

1. **void add(int data)**: This function takes one element and inserts it at the last. Amortized time complexity is **O(1)**.
2. **void add(int data, int index):** It inserts data at the specified index. Time complexity is **O(1)**.
3. **void addAll(Collection):** It inserts data at the specified index. Time complexity is **O(1)**.
4. **int get(int index):** It is used to get the element at the specified index. Time complexity is **O(1)**.
5. **void pop():** It deletes the last element. Time complexity is **O(1)**.
6. **removeAt(int index):**
7. **int size():** It returns the size of the ArrayList i.e, number of elements in the ArrayList. Time complexity is **O(1)**.
8. **int getcapacity():** It returns the capacity of the ArrayList. Time complexity is **O(1)**.
9. **void print():** It is used to print array elements. Time complexity is **O(N)**, where N is the size of the ArrayList.

| **Method** | **Running Time** |
| --- | --- |
| size() | O(1) |
| isEmpty() | O(1) |
| get(i) | O(1) |
| set(i, e) | O(1) |
| add(i, e) | O(n) |
| remove(i) | O(n) |

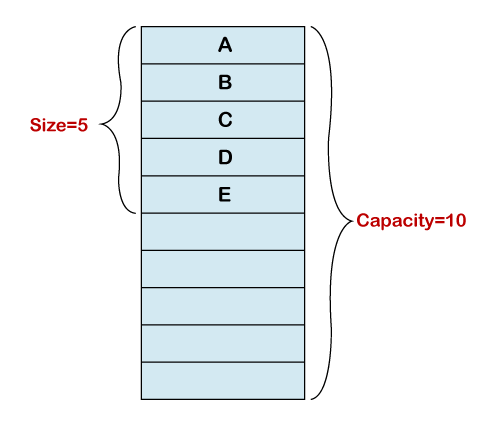
## **Working of Dynamic Array**

In the dynamic array, the elements are stored contiguously from the starting of the array and the remaining space remains unused. We can add the elements until the reserved spaced is completely consumed. When the reserved space is consumed and required to add some elements. In such a case, the fixed-sized array needs to be increased in size. Note that before appending the element, we allocate a bigger array, copy the elements from the array, and return the newly created array.

Another way to add an element is that first, create a function that creates a new array of double size, copies all the elements from the old array, and returns the new array. Similarly, we can also shrink the size of the dynamic array.

## **Size vs. Capacity**

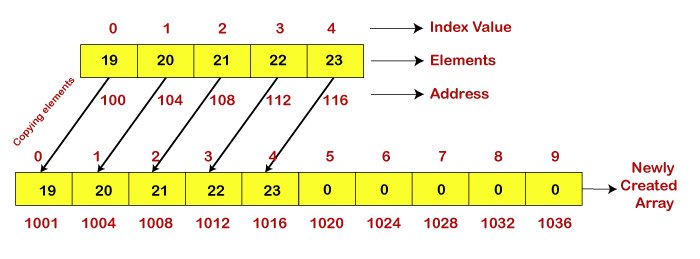
The initialization of a dynamic array creates a fixed-size array. In the following figure, the array implementation has 10 indices. We have added five elements to the array. Now, the underlying array has a length of five. Therefore, the length of the dynamic array size is 5 and its capacity is 10. The dynamic array keeps track of the endpoint.



## **Features of Dynamic Array**

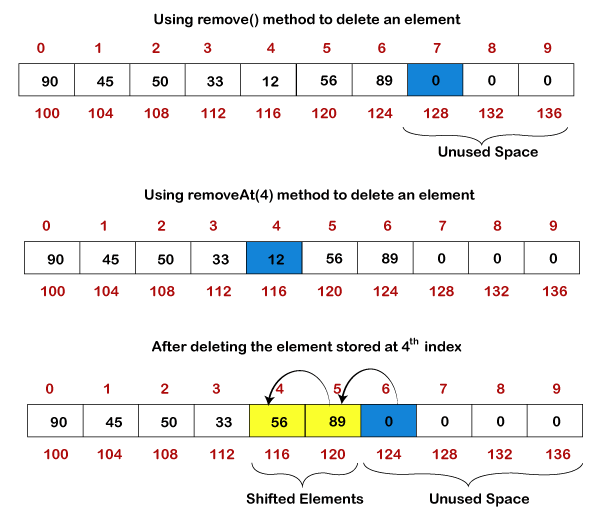
### **Add Element in a Dynamic Array**

In the dynamic array, we can create a fixed-size array if we required to add some more elements in the array. Usually, it creates a new array of double size. After that, it copies all the elements to the newly created array. We use the following approach:



### **Delete an Element from a Dynamic Array**

If we want to remove an element from the array at the specified index, we use the **removeAt(i)** method. The method parses the index number of that element which we want to delete. After deleting the element, it shifts the remaining elements (elements that are right to the deleted element) to the left from the specified index number. We also use the remove() method that deletes an element from the end of the array. After shifting the elements, it stores **0** at the palace of the last element. Let's understand it through an example, as we have shown in the following figure.



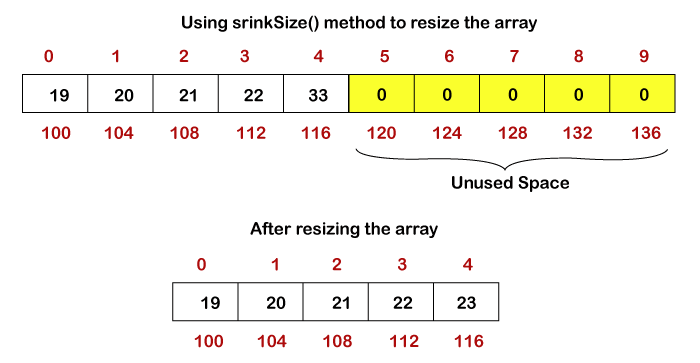
Resizing a Dynamic Array in Java

We need to resize an array in two scenarios if:

* The array uses extra memory than required.
* The array occupies all the memory and we need to add elements.

In the first case, we use the **srinkSize()** method to resize the [array](https://www.javatpoint.com/array-in-java). It reduces the size of the array. It free up the extra or unused memory. In the second case, we use the **growSize()** method to resize the array. It increases the size of the array.

It is an expensive operation because it requires a bigger array and copies all the elements from the previous array after that return the new array.



Suppose in the above array, it is required to add six more elements and, in the array, no more memory is left to store elements. In such cases, we grow the array using the **growSize()** method.

**Java ArrayList**

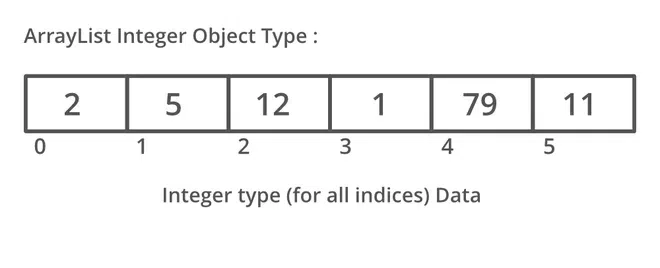
is a part of the Java **collection framework** and it is a class of java.util package. It provides us with dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed. This class is found in **java.util** package. The main **advantages of Java ArrayList** are, if we declare an array then it’s needed to mention the size but in ArrayList, it is not needed to mention the size of ArrayList if you want to mention the size then you can do it.

**What is ArrayList in Java?**

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 the Collection framework, it has many features not available with arrays.



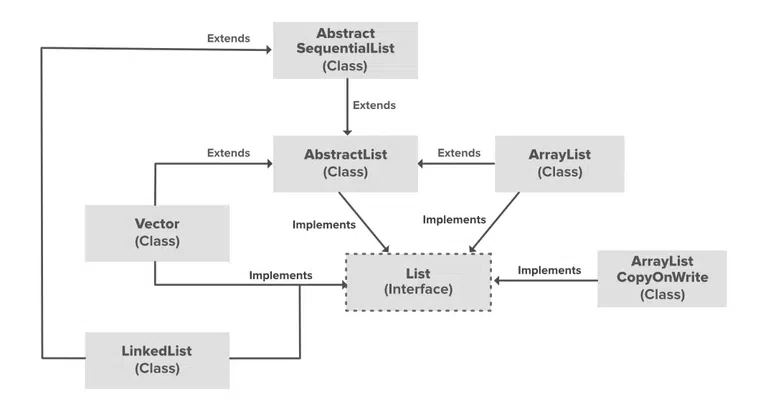
**Illustration:**



## **Important Features of ArrayList in Java**

* ArrayList inherits [AbstractList](https://www.geeksforgeeks.org/abstractlist-in-java-with-examples/) class and implements the [List interface](https://www.geeksforgeeks.org/list-interface-java-examples/).
* ArrayList is initialized by size. However, the size is increased automatically if the collection grows or shrinks if the [objects](https://www.geeksforgeeks.org/classes-objects-java/) are removed from the collection.
* Java ArrayList allows us to randomly access the list.
* ArrayList can not be used for [primitive types](https://www.geeksforgeeks.org/data-types-in-java/), like int, char, etc. We need a [wrapper class](https://www.geeksforgeeks.org/wrapper-classes-java/) for such cases.
* ArrayList in Java can be seen as a [vector in C++](https://www.geeksforgeeks.org/vector-in-cpp-stl/).
* ArrayList is **not Synchronized**. Its equivalent synchronized class in Java is [Vector](https://www.geeksforgeeks.org/java-util-vector-class-java/).

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



### **Some Key Points of ArrayList**

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.