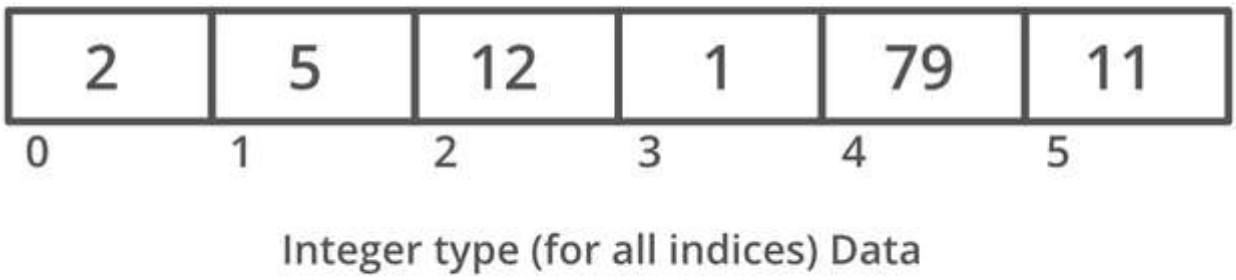
ArrayList in Java

Difficulty Level : Easy • Last Updated : 09 Feb, 2023

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 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.

Illustration:



Integer type (for all indices) Data

Example 1: The following implementation demonstrates how to create and use an ArrayList with mention it's size.

Java

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

import java.io.*;
import java.util.*;

class ArrayListExample {
    public static void main(String[] args)
    {
        // Size of the
        // ArrayList
        int n = 5;

        // Declaring the ArrayList with
        // initial size n
        ArrayList<Integer> arrli
            = new ArrayList<Integer>(n);

        // Appending new elements at
        // the end of the list
        for (int i = 1; i <= n; i++)
            arrli.add(i);

        // Printing elements
        System.out.println(arrli);

        // Remove element at index 3
        arrli.remove(3);
```

```
        System.out.println(arrli);

        // Printing elements one by one
        for (int i = 0; i < arrli.size(); i++)
            System.out.print(arrli.get(i) + " ");
    }
}
```

Output

```
[1, 2, 3, 4, 5]
[1, 2, 3, 5]
1 2 3 5
```

Example 2: The following implementation demonstrates how to create and use an ArrayList without the size mentioned.

Java

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

import java.io.*;
import java.util.*;

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

        // Declaring the ArrayList without mentioned its
        // size
        ArrayList<Integer> arrli = new ArrayList<>();

        // Appending new elements at
        // the end of the list
        for (int i = 1; i <= 5; i++)
            arrli.add(i);

        // Printing elements
        System.out.println(arrli);

        // Remove element at index 3
        arrli.remove(3);
```

```
        // Printing elements one by one
        for (int i = 0; i < arrli.size(); i++)
            System.out.print(arrli.get(i) + " ");
    }
}
```

Output

```
[1, 2, 3, 4, 5]
[1, 2, 3, 5]
1 2 3 5
```

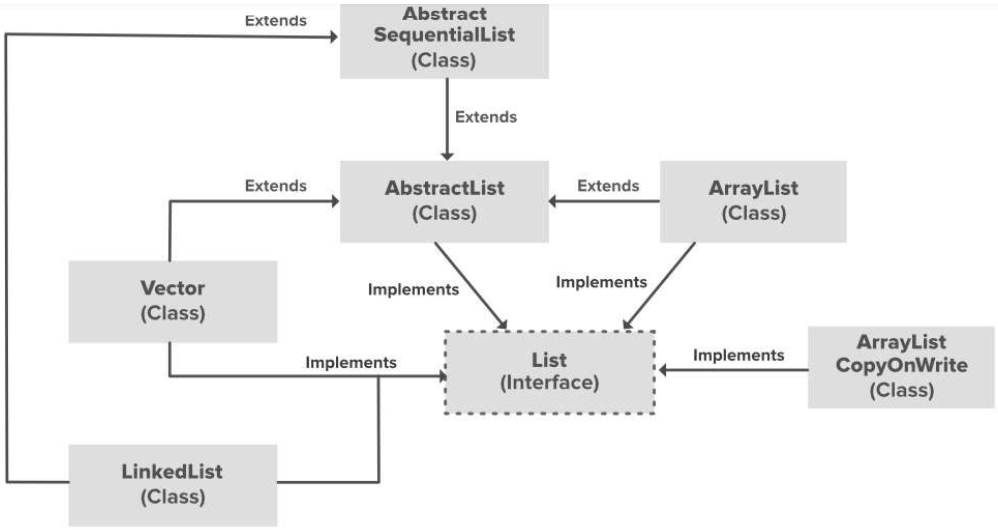
Output:

Output is the same as Example 1. 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 array when the array 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.

Important Features of ArrayList:

- 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.



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 the ArrayList

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:

1. ArrayList(): This constructor is used to build an empty array list. If we wish to create an empty ArrayList with the name **arr**, then, it can be created as:

2. ArrayList(Collection c): This constructor is used to build an array list initialized with the elements from the collection c. Suppose, we wish to create an ArrayList arr which contains the elements present in the collection c, then, it can be created as:

```
ArrayList arr = new ArrayList(c);
```

3. ArrayList(int capacity): This constructor is used to build an array list with initial capacity being specified. Suppose we wish to create an ArrayList with the initial size being N, then, it can be created as:

```
ArrayList arr = new ArrayList(N);
```

Methods in Java ArrayList

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.
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.
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 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.
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.

Note: *You can also create a generic ArrayList:*

```
// Creating generic integer ArrayList
ArrayList<Integer> arrli = new ArrayList<Integer>();
```

Some Key Point 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.

Let's see how to perform some basic operations on the ArrayList as listed which we are going to discuss further alongside implementing every operation.

- Adding element to List/ Add element
- Changing elements/ Set element
- Removing elements/Delete element
- Iterating elements
- get elements
- add elements in between two number
- Sorting elements
- ArrayList size

Operation 1: Adding Elements

In order to add an element to an ArrayList, we can use the add() method. This method is overloaded to perform multiple operations based on different parameters. They are as follows:

in the ArrayList.

Example:

Java

```
// Java Program to Add elements to An ArrayList

// Importing all utility classes
import java.util.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {
        // Creating an Array of string type
        ArrayList<String> al = new ArrayList<>();

        // Adding elements to ArrayList
        // Custom inputs
        al.add("Geeks");
        al.add("Geeks");

        // Here we are mentioning the index
        // at which it is to be added
        al.add(1, "For");

        // Printing all the elements in an ArrayList
        System.out.println(al);
    }
}
```

Output

```
[Geeks, For, Geeks]
```

Operation 2: Changing Elements

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

Example

Java

```java
// Java Program to Change elements in ArrayList

// Importing all utility classes
import java.util.*;

// main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {
        // Creating an Arraylist object of string type
        ArrayList<String> al = new ArrayList<>();

        // Adding elements to Arraylist
        // Custom input elements
        al.add("Geeks");
        al.add("Geeks");

        // Adding specifying the index to be added
        al.add(1, "Geeks");

        // Printing the Arraylist elements
        System.out.println("Initial ArrayList " + al);

        // Setting element at 1st index
        al.set(1, "For");

        //  Printing the updated Arraylist
        System.out.println("Updated ArrayList " + al);
    }
}
```

Output

```
Initial ArrayList [Geeks, Geeks, Geeks]
Updated ArrayList [Geeks, For, Geeks]
```

Operation 3: Removing Elements

parameters. They are as follows:

- **remove(Object):** This method is used to simply remove an object from the ArrayList. If there are multiple such objects, then the first occurrence of the object is removed.
- **remove(int index):** Since an ArrayList is indexed, this method takes an integer value which simply removes the element present at that specific index in the ArrayList. After removing the element, all the elements are moved to the left to fill the space and the indices of the objects are updated.

Example

Java

```
// Java program to Remove Elements in ArrayList

// Importing all utility classes
import java.util.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {
        // Creating an object of arraylist class
        ArrayList<String> al = new ArrayList<>();

        // Adding elements to ArrayList
        // Custom addition
        al.add("Geeks");
        al.add("Geeks");
        // Adding element at specific index
        al.add(1, "For");

        // Printing all elements of ArrayList
        System.out.println("Initial ArrayList " + al);

        // Removing element from above ArrayList
        al.remove(1);

        // Printing the updated Arraylist elements
        System.out.println("After the Index Removal " + al);
```

```
        // Now printing updated ArrayList
        System.out.println("After the Object Removal "
                           + al);
    }
}
```

Output

```
Initial ArrayList [Geeks, For, Geeks]
After the Index Removal [Geeks, Geeks]
After the Object Removal [Geeks]
```

Operation 4: Iterating the ArrayList

There are multiple ways to iterate through the ArrayList. The most famous ways are by using the basic for loop in combination with a get() method to get the element at a specific index and the advanced for loop.

Example

Java

```
// Java program to Iterate the elements
// in an ArrayList

// Importing all utility classes
import java.util.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {
        // Creating an Arraylist of string type
        ArrayList<String> al = new ArrayList<>();

        // Adding elements to ArrayList
        //  using standard add() method
        al.add("Geeks");
        al.add("Geeks");
        al.add(1, "For");
```

```
            System.out.print(al.get(i) + " ");
        }

        System.out.println();

        // Using the for each loop
        for (String str : al)
            System.out.print(str + " ");
    }
}
```

Output

```
Geeks For Geeks
Geeks For Geeks
```

Operation : 5-Get elements

Java

```
/*package whatever //do not write package name here */

import java.io.*;
import java.util.*;

class GFG {
    public static void main (String[] args) {
       ArrayList<Integer> list = new ArrayList();
     // add the number
     list.add(9);
     list.add(5);
     list.add(6);
     System.out.println(list);
     // get method
     Integer n= list.get(1);
     System.out.println("at indext 1 number is:"+n);
    }
}
```

Output

```
[9, 5, 6]
```

Java

```
import java.io.*;
import java.util.*;
class GFG {
    public static void main(String[] args)
    {
        ArrayList<Integer> list = new ArrayList();
        list.add(1);
        list.add(2);
        list.add(4);
        System.out.println(list);
        // insert missing element 3
        list.add(2, 3);
        System.out.println(list);
    }
}
```

Output

```
[1, 2, 4]
[1, 2, 3, 4]
```

Operation 7-Sorting element.

Java

```
import java.io.*;
import java.util.*;

class GFG {
    public static void main(String[] args)
    {
        ArrayList<Integer> list = new ArrayList();
        list.add(2);
        list.add(4);
        list.add(3);
        list.add(1);
        System.out.println("Before sorting list:");
        System.out.println(list);
        Collections.sort(list);
        System.out.println("after sorting list:");
        System.out.println(list);
```

Output

```
Before sorting list:
[2, 4, 3, 1]
after sorting list:
[1, 2, 3, 4]
```

Operation 8-Size of elements:

Java

```
import java.io.*;
import java.util.*;
class GFG {
    public static void main(String[] args)
    {
        ArrayList<Integer> list = new ArrayList();

        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        int b = list.size();
        System.out.println("The size is :" + b);
    }
}
```

Output

```
The size is :4
```

ArrayList in Java is a class in the Java Collection framework that implements the List interface. Here are the advantages and disadvantages of using ArrayList in Java:

Advantages:

1. Dynamic size: ArrayList can dynamically grow and shrink in size, making it easy to add or remove elements as needed.

3. Fast access: ArrayList provides fast access to elements, as it is implemented as an array under the hood.
4. Ordered collection: ArrayList preserves the order of elements, allowing you to access elements in the order they were added.
5. Supports null values: ArrayList can store null values, making it useful in cases where the absence of a value needs to be represented.

Disadvantages:

1. Slower than arrays: ArrayList is slower than arrays for certain operations, such as inserting elements in the middle of the list.
2. Increased memory usage: ArrayList requires more memory than arrays, as it needs to maintain its dynamic size and handle resizing.
3. 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.
4. 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.

Reference :

It depends on what you want to learn. If you're looking for a reference book for a specific subject, it would be helpful to specify what that subject is. Here are a few popular reference books in different fields:

1. Computer Science: "The Art of Computer Programming" by Donald Knuth
2. Mathematics: "Concrete Mathematics: A Foundation for Computer Science" by Donald Knuth, Oren Patashnik
3. History: "The Oxford Dictionary of World History" by John Keay
4. Medicine: "The Merck Manual of Diagnosis and Therapy"
5. Law: "Black's Law Dictionary" by Bryan A. Garner
6. Science: "The Oxford Dictionary of Science" edited by Michael Allaby

Must Read: Array vs ArrayList in Java

Previous

List Interface in Java with Examples

Next

Vector Class in Java

Related Articles

1. ArrayList of ArrayList in Java
2. How to clone an ArrayList to another ArrayList in Java?
3. Copy Elements of One ArrayList to Another ArrayList in Java
4. Java.util.ArrayList.add() Method in Java
5. Java.util.ArrayList.addall() method in Java
6. Java.util.Arraylist.indexOf() in Java
7. Java Program to Empty an ArrayList in Java
8. Array vs ArrayList in Java

10. Vector vs ArrayList in Java

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Improved By : KaashyapMSK, solankimayank, sagartomar9927, abhisheksonu1001, simmytarika5, aminul01, kamleshjoshi18, avinashrat55252, rathoadavinash

Article Tags : Java - util package, Java-ArrayList, Java-Collections, java-list, Java, Technical Scripter

Practice Tags : Java, Java-Collections

Improve Article | Report Issue

Start Your Coding Journey Now!