

You May Like

Sponsored Links by Taboola

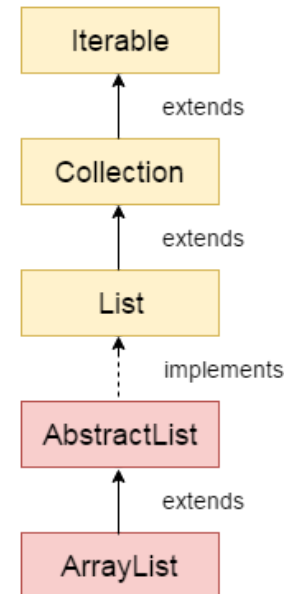
What You Probably Didn't Know About Your Favorite Wrestlers Now

Java ArrayList

Java **ArrayList** class uses a *dynamic array* for storing the elements. It is like an array, but there is *no size limit*. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the *java.util* package. It is like the Vector in C++.

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

It inherits the AbstractList class and implements **List interface**.



You May Like

Sponsored Links

You Wouldn't Recognize These Celebs Without Makeup

The important points about Java ArrayList class are:

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non **synchronized**.
- Java ArrayList allows random access because array works at the index basis.
- In ArrayList, manipulation is little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.

Hierarchy of ArrayList class



As shown in the above diagram, Java ArrayList class extends AbstractList class which implements List interface. The List interface extends the Collection and Iterable interfaces in hierarchical order.

ArrayList class declaration

Let's see the declaration for java.util.ArrayList class.

```
public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable
```

Constructors of ArrayList

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection<? extends E> c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Methods of ArrayList

Method	Description
void add (int index, E element)	It is used to insert the specified element at the specified position in a list.
boolean add (E e)	It is used to append the specified element at the end of a list.
boolean addAll (Collection<? extends E> c)	It is used to append 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)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void clear ()	It is used to remove 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)	It is used to fetch the element from the particular position of the list.
boolean isEmpty ()	It returns true if the list is empty, otherwise false.
Iterator ()	
listIterator ()	
int lastIndexOf (Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.



<code>Object[] toArray()</code>	It is used to return an array containing all of the elements in this list in the correct order.
<code><T> T[] toArray(T[] a)</code>	It is used to return an array containing all of the elements in this list in the correct order.
<code>Object clone()</code>	It is used to return a shallow copy of an ArrayList.
<code>boolean contains(Object o)</code>	It returns true if the list contains the specified element
<code>int indexOf(Object o)</code>	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.
<code>E remove(int index)</code>	It is used to remove the element present at the specified position in the list.
<code>boolean remove(Object o)</code>	It is used to remove the first occurrence of the specified element.
<code>boolean removeAll(Collection<?> c)</code>	It is used to remove all the elements from the list.
<code>boolean removeIf(Predicate<? super E> filter)</code>	It is used to remove all the elements from the list that satisfies the given predicate.
<code>protected void removeRange(int fromIndex, int toIndex)</code>	It is used to remove all the elements lies within the given range.
<code>void replaceAll(UnaryOperator<E> operator)</code>	It is used to replace all the elements from the list with the specified element.
<code>void retainAll(Collection<?> c)</code>	It is used to retain all the elements in the list that are present in the specified collection.
<code>E set(int index, E element)</code>	It is used to replace the specified element in the list, present at the specified position.
<code>void sort(Comparator<? super E> c)</code>	It is used to sort the elements of the list on the basis of specified comparator.
<code>Splitter<E> spliterator()</code>	It is used to create spliterator over the elements in a list.
<code>List<E> subList(int fromIndex, int toIndex)</code>	It is used to fetch all the elements lies within the given range.
<code>int size()</code>	It is used to return the number of elements present in the list.
<code>void trimToSize()</code>	It is used to trim the capacity of this ArrayList instance to be the list's current size.

Java Non-generic Vs. Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.



Java new generic collection allows you to have only one type of object in a collection. Now it is type safe so typecasting is not required at runtime.

Let's see the old non-generic example of creating java collection.

```
ArrayList list=new ArrayList();//creating old non-generic arraylist
```

Let's see the new generic example of creating java collection.

```
ArrayList<String> list=new ArrayList<String>();//creating new generic arraylist
```

In a generic collection, we specify the type in angular braces. Now ArrayList is forced to have the only specified type of objects in it. If you try to add another type of object, it gives *compile time error*.

For more information on Java generics, click here [Java Generics Tutorial](#).

Java ArrayList Example

```
import java.util.*;
public class ArrayListExample1{
    public static void main(String args[]){
        ArrayList<String> list=new ArrayList<String>();//Creating arraylist
        list.add("Mango");//Adding object in arraylist
        list.add("Apple");
        list.add("Banana");
        list.add("Grapes");
        //Printing the arraylist object
        System.out.println(list);
    }
}
```

Test it Now

Output:

```
[Mango, Apple, Banana, Grapes]
```

Iterating ArrayList using Iterator

Let's see an example to traverse ArrayList elements using the Iterator interface.

```
import java.util.*;
public class ArrayListExample2{
    public static void main(String args[]){
        ArrayList<String> list=new ArrayList<String>();//Creating arraylist
        list.add("Mango");//Adding object in arraylist
        list.add("Apple");
        list.add("Banana");
        list.add("Grapes");
        //Traversing list through Iterator
```



```
Iterator itr=list.iterator();//getting the Iterator
while(itr.hasNext()){//check if iterator has the elements
    System.out.println(itr.next());//printing the element and move to next
}
}
}
```

Test it Now

Output:

```
Mango
Apple
Banana
Grapes
```

Iterating ArrayList using For-each loop

Let's see an example to traverse the ArrayList elements using the for-each loop

```
import java.util.*;
public class ArrayListExample3{
    public static void main(String args[]){
        ArrayList<String> list=new ArrayList<String>();//Creating arraylist
        list.add("Mango");//Adding object in arraylist
        list.add("Apple");
        list.add("Banana");
        list.add("Grapes");
        //Traversing list through for-each loop
        for(String fruit:list)
            System.out.println(fruit);
    }
}
```

Output:

Test it Now

```
Mango
Apple
Banana
Grapes
```

Get and Set ArrayList

The *get()* method returns the element at the specified index, whereas the *set()* method changes the element.

```
import java.util.*;
public class ArrayListExample4{
```



```
public static void main(String args[]){
    ArrayList<String> al=new ArrayList<String>();
    al.add("Mango");
    al.add("Apple");
    al.add("Banana");
    al.add("Grapes");
    //accessing the element
    System.out.println("Returning element: "+al.get(1)); //it will return the 2nd element, because index starts from 0
    //changing the element
    al.set(1, "Dates");
    //Traversing list
    for(String fruit:al)
        System.out.println(fruit);
}
```

Test it Now

Output:

```
Returning element: Apple
Mango
Dates
Banana
Grapes
```

How to Sort ArrayList

The *java.util* package provides a utility class **Collections** which has the static method `sort()`. Using the **Collections.sort()** method, we can easily sort the ArrayList.

```
import java.util.*;
class SortArrayList{
    public static void main(String args[]){
        //Creating a list of fruits
        List<String> list1=new ArrayList<String>();
        list1.add("Mango");
        list1.add("Apple");
        list1.add("Banana");
        list1.add("Grapes");
        //Sorting the list
        Collections.sort(list1);
        //Traversing list through the for-each loop
        for(String fruit:list1)
            System.out.println(fruit);

        System.out.println("Sorting numbers...");
        //Creating a list of numbers
```



```
List<Integer> list2=new ArrayList<Integer>();  
list2.add(21);  
list2.add(11);  
list2.add(51);  
list2.add(1);  
//Sorting the list  
Collections.sort(list2);  
//Traversing list through the for-each loop  
for(Integer number:list2)  
    System.out.println(number);  
}  
  
}
```

Output:

```
Apple  
Banana  
Grapes  
Mango  
Sorting numbers...  
1  
11  
21  
51
```

Ways to iterate the elements of the collection in Java

There are various ways to traverse the collection elements:

1. By Iterator interface.
2. By for-each loop.
3. By ListIterator interface.
4. By for loop.
5. By forEach() method.
6. By forEachRemaining() method.

Iterating Collection through remaining ways

Let's see an example to traverse the ArrayList elements through other ways

```
import java.util.*;  
class ArrayList4{  
    public static void main(String args[]){  
        ArrayList<String> list=new ArrayList<String>();//Creating arraylist  
        list.add("Ravi");//Adding object in arraylist  
        list.add("Vijay");  
        list.add("Ravi");  
    }  
}
```



```
list.add("Ajay");

System.out.println("Traversing list through List Iterator:");
//Here, element iterates in reverse order
ListIterator<String> list1=list.listIterator(list.size());
while(list1.hasPrevious())
{
    String str=list1.previous();
    System.out.println(str);
}
System.out.println("Traversing list through for loop:");
for(int i=0;i<list.size();i++)
{
    System.out.println(list.get(i));
}

System.out.println("Traversing list through forEach() method:");
//The forEach() method is a new feature, introduced in Java 8.
list.forEach(a->{ //Here, we are using lambda expression
    System.out.println(a);
});

System.out.println("Traversing list through forEachRemaining() method:");
Iterator<String> itr=list.iterator();
itr.forEachRemaining(a-> //Here, we are using lambda expression
{
    System.out.println(a);
});
}
```

Output:

```
Traversing list through List Iterator:
Ajay
Ravi
Vijay
Ravi
Traversing list through for loop:
Ravi
Vijay
Ravi
Ajay
Traversing list through forEach() method:
Ravi
Vijay
Ravi
Ajay
Traversing list through forEachRemaining() method:
```




```
Ravi  
Vijay  
Ravi  
Ajay
```

User-defined class objects in Java ArrayList

Let's see an example where we are storing Student class object in an array list.

```
class Student{  
    int rollNo;  
    String name;  
    int age;  
    Student(int rollNo,String name,int age){  
        this.rollNo=rollNo;  
        this.name=name;  
        this.age=age;  
    }  
}
```

```
import java.util.*;  
class ArrayList5{  
    public static void main(String args[]){  
        //Creating user-defined class objects  
        Student s1=new Student(101,"Sonoo",23);  
        Student s2=new Student(102,"Ravi",21);  
        Student s3=new Student(103,"Hanumat",25);  
        //creating arraylist  
        ArrayList<Student> al=new ArrayList<Student>();  
        al.add(s1); //adding Student class object  
        al.add(s2);  
        al.add(s3);  
        //Getting Iterator  
        Iterator itr=al.iterator();  
        //traversing elements of ArrayList object  
        while(itr.hasNext()){  
            Student st=(Student)itr.next();  
            System.out.println(st.rollNo+" "+st.name+" "+st.age);  
        }  
    }  
}
```

Output:

```
101 Sonoo 23  
102 Ravi 21
```



Java ArrayList Serialization and Deserialization Example

Let's see an example to serialize an ArrayList object and then deserialize it.

```
import java.io.*;
import java.util.*;
class ArrayList6 {

    public static void main(String [] args)
    {
        ArrayList<String> al=new ArrayList<String>();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ajay");

        try
        {
            //Serialization
            FileOutputStream fos=new FileOutputStream("file");
            ObjectOutputStream oos=new ObjectOutputStream(fos);
            oos.writeObject(al);
            fos.close();
            oos.close();

            //Deserialization
            FileInputStream fis=new FileInputStream("file");
            ObjectInputStream ois=new ObjectInputStream(fis);
            ArrayList list=(ArrayList)ois.readObject();
            System.out.println(list);
        }catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

Output:

```
[Ravi, Vijay, Ajay]
```

Java ArrayList example to add elements

Here, we see different ways to add an element.

```
import java.util.*;
class ArrayList7{
```



```
public static void main(String args[]){
    ArrayList<String> al=new ArrayList<String>();
    System.out.println("Initial list of elements: "+al);
    //Adding elements to the end of the list
    al.add("Ravi");
    al.add("Vijay");
    al.add("Ajay");
    System.out.println("After invoking add(E e) method: "+al);
    //Adding an element at the specific position
    al.add(1, "Gaurav");
    System.out.println("After invoking add(int index, E element) method: "+al);
    ArrayList<String> al2=new ArrayList<String>();
    al2.add("Sonoo");
    al2.add("Hanumat");
    //Adding second list elements to the first list
    al.addAll(al2);
    System.out.println("After invoking addAll(Collection<? extends E> c) method: "+al);
    ArrayList<String> al3=new ArrayList<String>();
    al3.add("John");
    al3.add("Rahul");
    //Adding second list elements to the first list at specific position
    al.addAll(1, al3);
    System.out.println("After invoking addAll(int index, Collection<? extends E> c) method: "+al);
}
}
```

Output:

```
Initial list of elements: []
After invoking add(E e) method: [Ravi, Vijay, Ajay]
After invoking add(int index, E element) method: [Ravi, Gaurav, Vijay, Ajay]
After invoking addAll(Collection<? extends E> c) method:
[Ravi, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addAll(int index, Collection<? extends E> c) method:
[Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
```

Java ArrayList example to remove elements

Here, we see different ways to remove an element.

```
import java.util.*;
class ArrayList8 {

    public static void main(String [] args)
    {
        ArrayList<String> al=new ArrayList<String>();
        al.add("Ravi");
```



```

    al.add("Vijay");
    al.add("Ajay");
    al.add("Anuj");
    al.add("Gaurav");
    System.out.println("An initial list of elements: "+al);
    //Removing specific element from arraylist
    al.remove("Vijay");
    System.out.println("After invoking remove(object) method: "+al);
    //Removing element on the basis of specific position
    al.remove(0);
    System.out.println("After invoking remove(index) method: "+al);

    //Creating another arraylist
    ArrayList<String> al2=new ArrayList<String>();
    al2.add("Ravi");
    al2.add("Hanumat");
    //Adding new elements to arraylist
    al.addAll(al2);
    System.out.println("Updated list : "+al);
    //Removing all the new elements from arraylist
    al.removeAll(al2);
    System.out.println("After invoking removeAll() method: "+al);
    //Removing elements on the basis of specified condition
    al.removeIf(str -> str.contains("Ajay")); //Here, we are using Lambda expression
    System.out.println("After invoking removeIf() method: "+al);
    //Removing all the elements available in the list
    al.clear();
    System.out.println("After invoking clear() method: "+al);
}
}

```

Output:

```

An initial list of elements: [Ravi, Vijay, Ajay, Anuj, Gaurav]
After invoking remove(object) method: [Ravi, Ajay, Anuj, Gaurav]
After invoking remove(index) method: [Ajay, Anuj, Gaurav]
Updated list : [Ajay, Anuj, Gaurav, Ravi, Hanumat]
After invoking removeAll() method: [Ajay, Anuj, Gaurav]
After invoking removeIf() method: [Anuj, Gaurav]
After invoking clear() method: []

```

Java ArrayList example of retainAll() method

```

import java.util.*;

class ArrayList9{
    public static void main(String args[]){
        ArrayList<String> al=new ArrayList<String>();

```



```
al.add("Ravi");
al.add("Vijay");
al.add("Ajay");
ArrayList<String> al2=new ArrayList<String>();
al2.add("Ravi");
al2.add("Hanumat");
al.retainAll(al2);
System.out.println("iterating the elements after retaining the elements of al2");
Iterator itr=al.iterator();
while(itr.hasNext()){
    System.out.println(itr.next());
}
}
```

Output:

```
iterating the elements after retaining the elements of al2
Ravi
```

Java ArrayList example of isEmpty() method

```
import java.util.*;
class ArrayList10{

    public static void main(String [] args)
    {
        ArrayList<String> al=new ArrayList<String>();
        System.out.println("Is ArrayList Empty: "+al.isEmpty());
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ajay");
        System.out.println("After Insertion");
        System.out.println("Is ArrayList Empty: "+al.isEmpty());
    }
}
```

Output:

```
Is ArrayList Empty: true
After Insertion
Is ArrayList Empty: false
```

Java ArrayList Example: Book

Let's see an ArrayList example where we are adding books to list and printing all the books.



```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}

public class ArrayListExample20 {
public static void main(String[] args) {
    //Creating list of Books
    List<Book> list=new ArrayList<Book>();
    //Creating Books
    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
    Book b2=new Book(102,"Data Communications and Networking","Forouzan","Mc Graw Hill",4);
    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
    //Adding Books to list
    list.add(b1);
    list.add(b2);
    list.add(b3);
    //Traversing list
    for(Book b:list){
        System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
    }
}
}
```

Test it Now

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications and Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

Related Topics

[How to Sort ArrayList in Java](#)

[Difference between Array and ArrayList](#)

[When to use ArrayList and LinkedList in Java](#)

[Difference between ArrayList and LinkedList](#)

[Difference between ArrayList and Vector](#)



[How to Compare Two ArrayList in Java](#)[How to reverse ArrayList in Java](#)[When to use ArrayList and LinkedList in Java](#)[How to make ArrayList Read Only](#)[Difference between length of array and size\(\) of ArrayList in Java](#)[How to Synchronize ArrayList in Java](#)[How to convert ArrayList to Array and Array to ArrayList in java](#)[Array vs ArrayList in Java](#)[How to Sort Java ArrayList in Descending Order](#)[How to remove duplicates from ArrayList in Java](#)[← prev](#)[next →](#)

 [For Videos Join Our Youtube Channel: Join Now](#)

Help Others, Please Share



Learn Latest Tutorials



Apache Solr
Tutorial

[Solr](#)



MongoDB
tutorial

[MongoDB](#)



Gimp Tutorial
Gimp

[Gimp](#)



Verilog
Tutorial

[Verilog](#)



Teradata
Tutorial

[Teradata](#)



PhoneGap
Tutorial

[PhoneGap](#)



Gmail Tutorial
Gmail


[Gmail](#)




Vue.js Tutorial
Vue.js

[Vue.js](#)






PLC tutorial
PLC



Adobe
Illustrator
Tutorial
Illustrator




Postman
Tutorial
Postman


Preparation



Aptitude
Aptitude



Logical
Reasoning
Reasoning



Verbal Ability
Verbal A.



Interview
Questions
Interview



Company
Interview
Questions
Company

Trending Technologies




Artificial
Intelligence
Tutorial
AI




AWS Tutorial
AWS



Selenium
tutorial
Selenium




Cloud tutorial
Cloud




Hadoop
tutorial
Hadoop



ReactJS
Tutorial
ReactJS



Data Science
Tutorial
D. Science



Angular 7
Tutorial
Angular 7



Blockchain
Tutorial
Blockchain



Git Tutorial
Git



Machine
Learning Tutorial
ML



DevOps
Tutorial
DevOps


B.Tech / MCA




DBMS tutorial
DBMS



Data
Structures
tutorial
DS





















DAA tutorial
DAA



Operating
System tutorial
OS



 Computer Network tutorial C. Network	 Compiler Design tutorial Compiler D.	 Computer Organization and Architecture COA	 Discrete Mathematics Tutorial D. Math.
 Ethical Hacking Tutorial E. Hacking	 Computer Graphics Tutorial C. Graphics	 Software Engineering Tutorial Software E.	 html tutorial Web Tech.
 Cyber Security tutorial Cyber Sec.	 Automata Tutorial Automata	 C Language tutorial C	 C++ tutorial C++
 Java tutorial Java	 .Net Framework tutorial .Net	 Python tutorial Python	 List of Programs Programs
 Control Systems tutorial Control S.	 Data Mining Tutorial Data Mining		