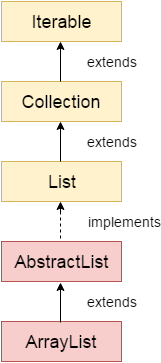
Java ArrayList class



Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

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 Java ArrayList class, manipulation is slow 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 Collection and Iterable interfaces in hierarchical order.

ArrayList class declaration

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

1. **public** **class** ArrayList<E> **extends** AbstractList<E> **implements** List<E>, RandomAccess, Cloneable, Serializable

Constructors of Java 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 Java 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. |
| 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. |
| Object[] toArray() | It is used to return an array containing all of the elements in this list in the  correct order. |
| <T> T[] toArray(T[] a) | It is used to return an array containing all of the elements in this list in the  correct order. |
| Object clone() | It is used to return a shallow copy of an ArrayList. |
| 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<? super E> filter) | It is used to remove all the elements from the list that satisfies the given  predicate. |
| protected void removeRange(int fromIndex, int toIndex) | It is used to remove all the elements 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  specified comparator. |
| Spliterator<E> spliterator() | It is used to create spliterator over the elements in a list. |
| List<E> subList(int fromIndex, int toIndex) | It is used to fetch all the elements lies within the given range. |
| int size() | It is used to return the number of elements present in the list. |
| void trimToSize() | 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.

1. ArrayList al=**new** ArrayList();//creating old non-generic arraylist

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

1. ArrayList<String> al=**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](https://www.javatpoint.com/generics-in-java" \t "_blank).

Java ArrayList Example

1. **import** java.util.\*;
2. **class** ArrayList1{
3. **public** **static** **void** main(String args[]){
4. ArrayList<String> list=**new** ArrayList<String>();//Creating arraylist
5. list.add("Ravi");//Adding object in arraylist
6. list.add("Vijay");
7. list.add("Ravi");
8. list.add("Ajay");
9. //Invoking arraylist object
10. System.out.println(list);
11. }
12. }
13. }

[Ravi, Vijay, Ravi, Ajay]

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 Iterator interface

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

1. **import** java.util.\*;
2. **class** ArrayList2{
3. **public** **static** **void** main(String args[]){
4. ArrayList<String> list=**new** ArrayList<String>();//Creating arraylist
5. list.add("Ravi");//Adding object in arraylist
6. list.add("Vijay");
7. list.add("Ravi");
8. list.add("Ajay");
9. //Traversing list through Iterator
10. Iterator itr=list.iterator();
11. **while**(itr.hasNext()){
12. System.out.println(itr.next());
13. }
14. }
15. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestCollection1" \t "_blank)**

Ravi

Vijay

Ravi

Ajay

Iterating Collection through the for-each loop

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

1. **import** java.util.\*;
2. **class** ArrayList3{
3. **public** **static** **void** main(String args[]){
4. ArrayList<String> al=**new** ArrayList<String>();
5. al.add("Ravi");
6. al.add("Vijay");
7. al.add("Ravi");
8. al.add("Ajay");
9. //Traversing list through for-each loop
10. **for**(String obj:al)
11. System.out.println(obj);
12. }
13. }

Ravi

Vijay

Ravi

Ajay

Iterating Collection through remaining ways

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

1. **import** java.util.\*;
2. **class** ArrayList4{
3. **public** **static** **void** main(String args[]){
4. ArrayList<String> list=**new** ArrayList<String>();//Creating arraylist
5. list.add("Ravi");//Adding object in arraylist
6. list.add("Vijay");
7. list.add("Ravi");
8. list.add("Ajay");
10. System.out.println("Traversing list through List Iterator:");
11. //Here, element iterates in reverse order
12. ListIterator<String> list1=list.listIterator(list.size());
13. **while**(list1.hasPrevious())
14. {
15. String str=list1.previous();
16. System.out.println(str);
17. }
18. System.out.println("Traversing list through for loop:");
19. **for**(**int** i=0;i<list.size();i++)
20. {
21. System.out.println(list.get(i));
22. }
24. System.out.println("Traversing list through forEach() method:");
25. //The forEach() method is a new feature, introduced in Java 8.
26. list.forEach(a->{ //Here, we are using lambda expression
27. System.out.println(a);
28. });
30. System.out.println("Traversing list through forEachRemaining() method:");
31. Iterator<String> itr=list.iterator();
32. itr.forEachRemaining(a-> //Here, we are using lambda expression
33. {
34. System.out.println(a);
35. });
36. }
37. }

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.

1. **class** Student{
2. **int** rollno;
3. String name;
4. **int** age;
5. Student(**int** rollno,String name,**int** age){
6. **this**.rollno=rollno;
7. **this**.name=name;
8. **this**.age=age;
9. }
10. }
11. **import** java.util.\*;
12. **class** ArrayList5{
13. **public** **static** **void** main(String args[]){
14. //Creating user-defined class objects
15. Student s1=**new** Student(101,"Sonoo",23);
16. Student s2=**new** Student(102,"Ravi",21);
17. Student s2=**new** Student(103,"Hanumat",25);
18. //creating arraylist
19. ArrayList<Student> al=**new** ArrayList<Student>();
20. al.add(s1);//adding Student class object
21. al.add(s2);
22. al.add(s3);
23. //Getting Iterator
24. Iterator itr=al.iterator();
25. //traversing elements of ArrayList object
26. **while**(itr.hasNext()){
27. Student st=(Student)itr.next();
28. System.out.println(st.rollno+" "+st.name+" "+st.age);
29. }
30. }
31. }

101 Sonoo 23

102 Ravi 21

103 Hanumat 25

Java ArrayList Serialization and Deserialization Example

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

1. **import** java.io.\*;
2. **import** java.util.\*;
3. **class** ArrayList6 {
5. **public** **static** **void** main(String [] args)
6. {
7. ArrayList<String> al=**new** ArrayList<String>();
8. al.add("Ravi");
9. al.add("Vijay");
10. al.add("Ajay");
12. **try**
13. {
14. //Serialization
15. FileOutputStream fos=**new** FileOutputStream("file");
16. ObjectOutputStream oos=**new** ObjectOutputStream(fos);
17. oos.writeObject(al);
18. fos.close();
19. oos.close();
20. //Deserialization
21. FileInputStream fis=**new** FileInputStream("file");
22. ObjectInputStream ois=**new** ObjectInputStream(fis);
23. ArrayList  list=(ArrayList)ois.readObject();
24. System.out.println(list);
25. }**catch**(Exception e)
26. {
27. System.out.println(e);
28. }
29. }
30. }

[Ravi, Vijay, Ajay]

Java ArrayList example to add elements

Here, we see different ways to add an element.

1. **import** java.util.\*;
2. **class** ArrayList7{
3. **public** **static** **void** main(String args[]){
4. ArrayList<String> al=**new** ArrayList<String>();
5. System.out.println("Initial list of elements: "+al);
6. //Adding elements to the end of the list
7. al.add("Ravi");
8. al.add("Vijay");
9. al.add("Ajay");
10. System.out.println("After invoking add(E e) method: "+al);
11. //Adding an element at the specific position
12. al.add(1, "Gaurav");
13. System.out.println("After invoking add(int index, E element) method: "+al);
14. ArrayList<String> al2=**new** ArrayList<String>();
15. al2.add("Sonoo");
16. al2.add("Hanumat");
17. //Adding second list elements to the first list
18. al.addAll(al2);
19. System.out.println("After invoking addAll(Collection<? extends E> c) method: "+al);
20. ArrayList<String> al3=**new** ArrayList<String>();
21. al3.add("John");
22. al3.add("Rahul");
23. //Adding second list elements to the first list at specific position
24. al.addAll(1, al3);
25. System.out.println("After invoking addAll(int index, Collection<? extends E> c) method: "+al);
27. }
28. }

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.

1. **import** java.util.\*;
2. **class** ArrayList8 {
4. **public** **static** **void** main(String [] args)
5. {
6. ArrayList<String> al=**new** ArrayList<String>();
7. al.add("Ravi");
8. al.add("Vijay");
9. al.add("Ajay");
10. al.add("Anuj");
11. al.add("Gaurav");
12. System.out.println("An initial list of elements: "+al);
13. //Removing specific element from arraylist
14. al.remove("Vijay");
15. System.out.println("After invoking remove(object) method: "+al);
16. //Removing element on the basis of specific position
17. al.remove(0);
18. System.out.println("After invoking remove(index) method: "+al);
20. //Creating another arraylist
21. ArrayList<String> al2=**new** ArrayList<String>();
22. al2.add("Ravi");
23. al2.add("Hanumat");
24. //Adding new elements to arraylist
25. al.addAll(al2);
26. System.out.println("Updated list : "+al);
27. //Removing all the new elements from arraylist
28. al.removeAll(al2);
29. System.out.println("After invoking removeAll() method: "+al);
30. //Removing elements on the basis of specified condition
31. al.removeIf(str -> str.contains("Ajay"));   //Here, we are using Lambda expression
32. System.out.println("After invoking removeIf() method: "+al);
33. //Removing all the elements available in the list
34. al.clear();
35. System.out.println("After invoking clear() method: "+al);
36. }
37. }

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

1. **import** java.util.\*;
2. **class** ArrayList9{
3. **public** **static** **void** main(String args[]){
4. ArrayList<String> al=**new** ArrayList<String>();
5. al.add("Ravi");
6. al.add("Vijay");
7. al.add("Ajay");
8. ArrayList<String> al2=**new** ArrayList<String>();
9. al2.add("Ravi");
10. al2.add("Hanumat");
11. al.retainAll(al2);
12. System.out.println("iterating the elements after retaining the elements of al2");
13. Iterator itr=al.iterator();
14. **while**(itr.hasNext()){
15. System.out.println(itr.next());
16. }
17. }
18. }

iterating the elements after retaining the elements of al2

Ravi

Java ArrayList example of isEmpty() method

1. **import** java.util.\*;
2. **class** ArrayList10{
4. **public** **static** **void** main(String [] args)
5. {
6. ArrayList<String> al=**new** ArrayList<String>();
7. System.out.println("Is ArrayList Empty: "+al.isEmpty());
8. al.add("Ravi");
9. al.add("Vijay");
10. al.add("Ajay");
11. System.out.println("After Insertion");
12. System.out.println("Is ArrayList Empty: "+al.isEmpty());
13. }
14. }

Is ArrayList Empty: true

After Insertion

Is ArrayList Empty: false

Java ArrayList example of set() and get() method

1. **import** java.util.\*;
2. **class** ArrayList11 {
4. **public** **static** **void** main(String [] args)
5. {
6. ArrayList<String> al=**new** ArrayList<String>();
7. al.add("Ravi");
8. al.add("Vijay");
9. al.add("Ajay");
10. System.out.println("Before update: "+al.get(1));
11. //Updating an element at specific position
12. al.set(1,"Gaurav");
13. System.out.println("After update: "+al.get(1));
14. }
15. }

Before update: Vijay

After update: Gaurav

Java ArrayList Example: Book

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

1. **import** java.util.\*;
2. **class** Book {
3. **int** id;
4. String name,author,publisher;
5. **int** quantity;
6. **public** Book(**int** id, String name, String author, String publisher, **int** quantity) {
7. **this**.id = id;
8. **this**.name = name;
9. **this**.author = author;
10. **this**.publisher = publisher;
11. **this**.quantity = quantity;
12. }
13. }
14. **public** **class** ArrayListExample {
15. **public** **static** **void** main(String[] args) {
16. //Creating list of Books
17. List<Book> list=**new** ArrayList<Book>();
18. //Creating Books
19. Book b1=**new** Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
20. Book b2=**new** Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);
21. Book b3=**new** Book(103,"Operating System","Galvin","Wiley",6);
22. //Adding Books to list
23. list.add(b1);
24. list.add(b2);
25. list.add(b3);
26. //Traversing list
27. **for**(Book b:list){
28. System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
29. }
30. }
31. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestCollection101" \t "_blank)**

Output:

101 Let us C Yashwant Kanetkar BPB 8

102 Data Communications & Networking Forouzan Mc Graw Hill 4

103 Operating System Galvin Wiley 6