Array vs ArrayList in Java

Array is a fixed length data structure whereas ArrayList is a variable length Collection class. We cannot change length of array once created in Java but ArrayList can be changed.

We cannot store primitives in ArrayList, it can only store objects. But array can contain both primitives and objects in Java. Since Java 5, primitives are automatically converted in objects which is known as auto-boxing.

1. **import** java.util.\*;
2. **public** **class** ListExample {
3. **public** **static** **void** main(String[] args) {
4. List<Integer> list=**new** ArrayList<>();
5. list.add(Integer.valueOf(10));//storing Integer object
6. list.add(20);//Now compiler converts it into Integer.valueOf(20) which is object
7. list.add(30);
9. System.out.println("Traversing List...");
10. **for**(Integer i:list){
11. System.out.println(i);
12. }
13. }
14. }

Output:

Traversing List...

10

20

30

Difference between length of array and size() of ArrayList in Java

ArrayList doesn't have length() method, the size() method of ArrayList provides the number of objects available in the collection.

Array has length property which provides the length or capacity of the Array. It is the total space allocated during the intialization of the array.

1. **import** java.util.ArrayList;
2. **public** **class** LengthVsSize {
3. **public** **static** **void** main(String[] args) {
4. //creating array of 10 elements
5. **int** arr[]=**new** **int**[10];
6. //storing 2 elements
7. arr[0]=10;
8. arr[1]=12;
9. //printing length of array
10. System.out.println(arr.length);//10
12. //Creating ArrayList
13. ArrayList<String> list=**new** ArrayList<String>();
14. //storing 2 elements
15. list.add("ankit");
16. list.add("nippun");
17. //printing size of ArrayList
18. System.out.println(list.size());//2
19. }
20. }

Output:

10

2

How to convert ArrayList to Array and Array to ArrayList in java?

Let's see a simple example to convert ArrayList to Array and Array to ArrayList in Java:

1. **public** **class** LengthVsSizeArrayList {
2. **public** **static** **void** main(String[] args) {
3. //creating Arraylist
4. List<String> fruitList = **new** ArrayList<>();
5. //adding String Objects to fruitsList ArrayList
6. fruitList.add("Mango");
7. fruitList.add("Banana");
8. fruitList.add("Apple");
9. fruitList.add("Strawberry");
10. fruitList.add("Pineapple");
11. System.out.println("Converting ArrayList to Array" );
12. String[] item = fruitList.toArray(**new** String[fruitList.size()]);
13. **for**(String s : item){
14. System.out.println(s);
15. }
16. System.out.println("Converting Array to ArrayList" );
17. List<String>l2 = **new** ArrayList<>();
18. l2 =  Arrays.asList(item);
19. System.out.println(l2);
20. }
21. }

Output:

Converting ArrayList to Array

Mango

Banana

Apple

Strawberry

Pineapple

Converting Array to ArrayList

[Mango, Banana, Apple, Strawberry, Pineapple]

How to make ArrayList Read Only?

The read-only means unmodifiable view of Collection in which we can not perform any operation which will change the collection through add(), remove() or set() method. We can obtain read-only collection from the existing collection by calling Collections.unmodifiableCollection() method.

1. **public** **class** UnmodifiableArrayList {
2. **public** **static** **void** main(String[] args) {
3. List<String>fruitList = **new** ArrayList<String>();
5. fruitList.add("Mango");
6. fruitList.add("Banana");
7. fruitList.add("Apple");
8. fruitList.add("Strawberry");
9. fruitList.add("Pineapple");
11. List<String>unmodifiableList= Collections.unmodifiableList(fruitList);
12. unmodifiableList.add("INDIA");
13. System.out.println(fruitList);
14. }
15. }

Output:

Exception in thread "main"java.lang.UnsupportedOperationException

at java.util.Collections$UnmodifiableCollection.add(Collections.java:1055)

at collectionInterview.list.UnmodifiableArrayList.main(UnmodifiableArrayList.java:20)

Here we have converted the existing list fruitList to unmodifiable List. If we alter the "unmodifiableList", it will cause UnsupportedOperationException. We can still change the list with fruitList reference. like calling:

1. fruitList.add("INDIA");

How to remove duplicates from ArrayList in Java?

To remove dupliates from ArrayList, we can convert it into Set. Since Set doesn't contain duplicate elements, it will have only unique elements.

Let's see an example to remove duplicates from ArrayList:

1. **public** **class** RemoveDuplicateArrayList {
2. **public** **static** **void** main(String[] args) {
3. List<String> l = **new** ArrayList<String>();
4. l.add("Mango");
5. l.add("Banana");
6. l.add("Mango");
7. l.add("Apple");
8. System.out.println(l.toString());
9. Set<String> s = **new** LinkedHashSet<String>(l);
10. System.out.println(s);
11. }
12. }

Output:

Before converting to set

[Mango, Banana, Mango, Apple]

After converting to set

[Mango, Banana, Apple]

How to reverse ArrayList in Java?

The reverse method of Collections class can be used to reverse any collection. It is a static method. Let's see the signature of reverse method:

1. **public** **static** **void** reverse(Collection c)

Let's see a simple example to reverse ArrayList in Java:

1. **public** **class** ReverseArrayList {
2. **public** **static** **void** main(String[] args) {
3. List<String> l = **new** ArrayList<String>();
4. l.add("Mango");
5. l.add("Banana");
6. l.add("Mango");
7. l.add("Apple");
8. System.out.println("Before Reversing");
9. System.out.println(l.toString());
11. Collections.reverse(l);
12. System.out.println("After Reversing");
13. System.out.println(l);
14. }
15. }

Output:

Before Reversing

[Mango, Banana, Mango, Apple]

After Reversing

[Apple, Mango, Banana, Mango]

How to Sort Java ArrayList in Descending Order

By using Collections.reverseOrder(Comparator<T>cmp) method, we can sort the collection in reverse order. The reverseOrder() method does the reversing on the basis of given Comparator. In case of null, it will reverse collection in natural ordering.

Let's see a simple example to sort the ArrayList in descending order.

***SmartPhone.java***

1. **import** java.util.Comparator;
2. **public** **class** SmartPhone {
3. String brand;
4. String model;
5. intprice;
6. intrating;
7. SmartPhone(String brand,String model,intprice, intrating){
8. **this**.brand = brand;
9. **this**.model = model;
10. **this**.price = price;
11. **this**.rating = rating;
13. }
14. **public** String getBrand() {
15. returnbrand;
16. }
17. **public** **void** setBrand(String brand) {
18. **this**.brand = brand;
19. }
20. **public** String getModel() {
21. returnmodel;
22. }
23. **public** **void** setModel(String model) {
24. **this**.model = model;
25. }
26. **public** **int** getPrice() {
27. returnprice;
28. }
29. **public** **void** setPrice(intprice) {
30. **this**.price = price;
31. }
32. **public** **int** getRating() {
33. returnrating;
34. }
35. **public** **void** setRating(intrating) {
36. **this**.rating = rating;
37. }
38. **public** String toString() {
39. **return**"SmartPhone [brand=" + brand + ", model=" + model + ", price=" + price + ", rating=" + rating + "]";
40. }
41. **public** **int** compareTo(SmartPhone sp) {
42. returnthis.price - sp.price;
44. }
45. }
46. **class** RatingComparator **implements** Comparator<SmartPhone> {
47. @Override
48. **public** **int** compare(SmartPhone obj1, SmartPhone obj2) {
49. **return** (obj1.rating<obj2.rating) ? -1 : (obj1.rating>obj2.rating) ? 1 : 0;
50. }
51. }

***ArrayListLearning.java***

1. **public** **class** ArrayListLearning {
2. @SuppressWarnings("unchecked")
3. **public** **static** **void** main(String[] args) {
5. List<SmartPhone> phoneList = **new** ArrayList<>();
6. SmartPhone ph1 = **new** SmartPhone("Apple", "6s", 50000, 10);
7. SmartPhone ph2 = **new** SmartPhone("lg", "pro2", 40000, 9);
8. SmartPhone ph3 = **new** SmartPhone("MI", "3s", 10000, 6);
9. SmartPhone ph4 = **new** SmartPhone("Letv", "le2", 12000, 7);
11. phoneList.add(ph1);
12. phoneList.add(ph2);
13. phoneList.add(ph3);
14. phoneList.add(ph4);
15. System.out.println("Actual List");
16. System.out.println(phoneList);
17. System.out.println("Sorting the list as comparator");
18. Collections.sort(phoneList, **new** RatingComparator());
20. System.out.println(phoneList);
21. System.out.println("Reversing the Comparator sorting");
22. Comparator<SmartPhone> cmp = Collections.reverseOrder(**new** RatingComparator());
24. Collections.sort(phoneList, cmp);
25. System.out.println("Printing the reverse list");
26. System.out.println(phoneList);
27. }
29. }

Output:

Actual List

[SmartPhone [brand=Apple, model=6s, price=50000, rating=10],

SmartPhone [brand=lg, model=pro2, price=40000, rating=9],

SmartPhone [brand=MI, model=3s, price=10000, rating=6],

SmartPhone [brand=Letv, model=le2, price=12000, rating=7]]

Sorting the list as comparator

[SmartPhone [brand=MI, model=3s, price=10000, rating=6],

SmartPhone [brand=Letv, model=le2, price=12000, rating=7],

SmartPhone [brand=lg, model=pro2, price=40000, rating=9],

SmartPhone [brand=Apple, model=6s, price=50000, rating=10]]

Reversing the Comparator sorting

Printing the reverse list

[SmartPhone [brand=Apple, model=6s, price=50000, rating=10],

SmartPhone [brand=lg, model=pro2, price=40000, rating=9],

SmartPhone [brand=Letv, model=le2, price=12000, rating=7],

SmartPhone [brand=MI, model=3s, price=10000, rating=6]]

How to Synchronize ArrayList in Java?

We can use Collections.synchronizedList(List<T>) method to synchronize collections in java. The synchronizedList(List<T>) method is used to return a synchronized (thread-safe) list backed by the specified list.

1. **import** java.util.\*;
2. **public** **class** SyncronizeArrayList {
3. **public** **static** **void** main(String args[]) {
4. // Non Synchronized ArrayList
5. List<String> fruitList = **new** ArrayList<String>();
7. fruitList.add("Mango");
8. fruitList.add("Banana");
9. fruitList.add("Apple");
10. fruitList.add("Strawberry");
11. fruitList.add("Pineapple");
13. // Synchronizing ArrayList in Java
14. furitList = Collections.synchronizedList(fruitList);
16. // we must use synchronize block to avoid non-deterministic behavior
17. **synchronized** (fruitList) {
18. Iterator<String> itr = fruitList.iterator();
19. **while** (itr.hasNext()) {
20. System.out.println(itr.next());
21. }
22. }
23. }
24. }

Output:

Mango

Banana

Apple

Strawberry

Pineapple

# When to use ArrayList and LinkedList in Java

ArrayList provides constant time for search operation, so it is better to use ArrayList if searching is more frequent operation than add and remove operation. The LinkedList provides constant time for add and remove operations. So it is better to use LinkedList for manipulation.

ArrayList has O(1) time complexity to access elements via the get and set methods.

LinkedList has O(n/2) time complexity to access the elements.

LinkedLinked class implements Deque interface also, so you can get the functionality of double ended queue in LinkedList. The ArrayList class doesn't implement Deque interface.

In sort, ArrayList is better to access data wherease LinkedList is better to manipulate data. Both classes implements List interface.

### ArrayList Example

1. **import** java.util.\*;
2. **public** **class** ListExample {
3. **public** **static** **void** main(String[] args) {
4. //ArrayList is better to store and view data
5. List<String> list=**new** ArrayList<>();
6. list.add("ankit");
7. list.add("peter");
8. list.add("mayank");
10. System.out.println("Traversing ArrayList...");
11. **for**(String s:list){
12. System.out.println(s);
13. }
14. }
15. }

Output:

Traversing ArrayList...

ankit

peter

mayank

### LinkedList Example

1. **import** java.util.\*;
2. **public** **class** ListExample2 {
3. **public** **static** **void** main(String[] args) {
4. //LinkedList is better to manipulate data
5. List<String> list=**new** LinkedList<>();
6. list.add("ankit");
7. list.add("peter");
8. list.add("mayank");
9. System.out.println("After adding: "+list);
10. list.remove("peter");
11. System.out.println("After removing: "+list);
12. list.set(1,"vivek");
13. System.out.println("After changing: "+list);
14. }
15. }

Output:

After adding: [ankit, peter, mayank]

After removing: [ankit, mayank]

After changing: [ankit, vivek]