**ArrayList**

* [ArrayList – Introduction](https://howtodoinjava.com/java-arraylist/)
* [ArrayList – add()](https://howtodoinjava.com/java/collections/arraylist/arraylist-add-example/)
* [ArrayList – addAll()](https://howtodoinjava.com/java/collections/arraylist/arraylist-addall-method-example/)
* [ArrayList – clear()](https://howtodoinjava.com/java/collections/arraylist/clear-empty-arraylist/)
* [ArrayList – clone()](https://howtodoinjava.com/java/collections/arraylist/arraylist-clone-deep-copy/)
* [ArrayList – contains()](https://howtodoinjava.com/java/collections/arraylist/arraylist-contains/)
* [ArrayList – ensureCapacity()](https://howtodoinjava.com/java/collections/arraylist/arraylist-ensurecapacity-method/)
* [ArrayList – forEach()](https://howtodoinjava.com/java/collections/arraylist/arraylist-foreach/)
* [ArrayList – get()](https://howtodoinjava.com/java/collections/arraylist/arraylist-get-method-example/)
* [Arraylist – indexOf()](https://howtodoinjava.com/java/collections/arraylist/arraylist-indexof-method/)
* [Arraylist – lastIndexOf()](https://howtodoinjava.com/java/collections/arraylist/arraylist-lastindexof-example/)
* [ArrayList – listIterator()](https://howtodoinjava.com/java/collections/arraylist/arraylist-listiterator/)
* [ArrayList – remove()](https://howtodoinjava.com/java/collections/arraylist/arraylist-remove-example/)
* [ArrayList – removeAll()](https://howtodoinjava.com/java/collections/arraylist/arraylist-removeall/)
* [ArrayList – removeIf()](https://howtodoinjava.com/java/collections/arraylist/arraylist-removeif/)
* [ArrayList – retainAll()](https://howtodoinjava.com/java/collections/arraylist/arraylist-retainall-example/)
* [ArrayList – sort()](https://howtodoinjava.com/java/collections/arraylist/arraylist-sort-objects-by-field/)
* [ArrayList – spliterator()](https://howtodoinjava.com/java/collections/arraylist/arraylist-spliterator-example/)
* [ArrayList – subList()](https://howtodoinjava.com/java/collections/arraylist/arraylist-sublist/)
* [ArrayList – toArray()](https://howtodoinjava.com/java/collections/arraylist/convert-arraylist-to-array/)
* [ArrayList – Initialize arraylist](https://howtodoinjava.com/java/collections/arraylist/initialize-arraylist/)
* [ArrayList – Iteration](https://howtodoinjava.com/java/collections/arraylist/iterate-through-objects/)
* [ArrayList – Add/replace element](https://howtodoinjava.com/java/collections/arraylist/add-replace-element-at-index/)
* [ArrayList – Add multiple elements](https://howtodoinjava.com/java/collections/arraylist/add-multiple-elements-arraylist/)
* [ArrayList – Check empty list](https://howtodoinjava.com/java/collections/arraylist/check-arraylist-empty/)
* [ArrayList – Remove element](https://howtodoinjava.com/java/collections/arraylist/remove-element-from-arraylist/)
* [ArrayList – Replace element](https://howtodoinjava.com/java/collections/arraylist/replace-element-arraylist/)
* [ArrayList – Empty arraylist](https://howtodoinjava.com/java/collections/arraylist/empty-clear-arraylist/)
* [ArrayList – Synchronized arraylist](https://howtodoinjava.com/java/collections/arraylist/synchronize-arraylist/)
* [ArrayList – Compare two lists](https://howtodoinjava.com/java/collections/arraylist/compare-two-arraylists/)
* [ArrayList – Remove duplicates](https://howtodoinjava.com/java/collections/arraylist/remove-duplicate-elements/)
* [ArrayList – Merge two lists](https://howtodoinjava.com/java/collections/arraylist/merge-arraylists/)
* [ArrayList – Serialization](https://howtodoinjava.com/java/collections/arraylist/serialize-deserialize-arraylist/)
* [ArrayList – Swap two elements](https://howtodoinjava.com/java/collections/arraylist/swap-two-elements-arraylist/)
* [Convert ArrayList to Array](https://howtodoinjava.com/java/collections/arraylist/arraylist-to-array/)
* [Convert Array to ArrayList](https://howtodoinjava.com/java/collections/arraylist/array-to-arraylist/)
* [Convert HashSet to ArrayList](https://howtodoinjava.com/java/collections/arraylist/hashset-to-arraylist/)
* [Convert LinkedList to ArrayList](https://howtodoinjava.com/java/collections/arraylist/convert-linkedlist-arraylist/)
* [Convert Vector to ArrayList](https://howtodoinjava.com/java/collections/arraylist/convert-vector-to-arraylist/)
* [ArrayList vs LinkedList](https://howtodoinjava.com/java/collections/arraylist/linkedlist-vs-arraylist/)
* [ArrayList vs Vector](https://howtodoinjava.com/java/collections/arraylist/arraylist-vs-vector/)

It extends AbstractList which implements List interface. The List extends Collection and Iterable interfaces in hierarchical order. ArrayList Features

1. **Ordered**
2. **Index based**
3. **Dynamic resizing**
4. **Non synchronized** – ArrayList is not synchronized, by default. Programmer needs to use synchronized keyword appropiately or simply use **Vector** class.
5. **Duplicates allowed**

## How ArrayList Works

is implemented with a backing array.

The ArrayList offers to remove this sizing limitation. An ArrayList can be created with any initial size (default 16), and when we add more items, the size of the arraylist grows dynamically without any intervention by the programmer.

In array list, we are supposed to add only objects. But in case, we are required to add primitive data types such as int, float etc, we can use their wrapper classes for providing type information during arraylist initialization.

 ArrayList as *dynamic array*.

 Use generics for compile time type safety while adding the element to arraylist.

**add()** method first ensures that there is sufficient space in the arraylist. If list does not have space, then it grows the list by adding more spaces in underlying array. Then it add the element to specific array index.

**clear()** method does simple thing. It iterates the backing array inside arraylist and assign all elements 'null' value and set the size attribute to '0'.

The difference between an empty and a new arraylist is the **size of backing array**. As clear() method does not resize the backing array, so after clear method you may have a list which has backing array of a larger size (if list was pretty big before clear() method was called).

**ArrayList clone()** method is used to create a **shallow copy** of the list. In the new list, only object references are copied.

Creating a deep copy of a list is not straightforward. **Java does not support deep copying by default**. So we have to manually modify the code to enable the deep copying of classes and collections.

In Java, to support deep copy, we must override the clone() of model classes. In clone() method, we must ensure that when somebody invokes object.clone() method then it must return a deep copy of that model class (e.g. Employee class).

To create a deep copy of the class, divide all the class members into two categories of **mutable and immutable types**.

1. All immutable field members can be used as it is. They don’t require any special treatment. e.g. primitive classes, wrapper classes and String class.
2. For all immutable field members, we must create a new object of member and assign it’s value to cloned object.

The idea is to return an **immutable copy** of the class from clone() method.

## **ArrayList.indexOf() method**

This method returns the index of the first occurrence of the specified element in this list

It will return '-1' if the list does not contain the element.

We can use this method to find if an object is present in arraylist. If the object is present then return value will be greater than '-1‘.

This method is overloaded variant of previous method. It accepts one more argument 'fromIndex' at which to insert the first element from the specified collection.

list1.addAll(2, list2); //Elements will be inserted from index 2

## **How ArrayLists grow in size?**

Arrays are fixed size collection whereas arraylists dynamically grow as soon as backing array is full and more elements are added to list. This is done in two steps:

1. Create a new backing array with more size than previous array.
2. Copy all elements from old array to this new array.

So basically, before adding any new element with **add()** method, ArrayList perform a check whether there is any space left in array or not using ensureCapacity() method. If space is available then element is added; else a new backing array is created first.

. But if we do not pass any size, the default size is used which is 10.

 then always pass the size in list constructor. It eliminates theneed to resize the arraylist and thus improve performance.

## **Increase arraylist capacity – ArrayList ensureCapacity() method**

Normally, we do not need to do anything to increase the size because arraylist automatically manages the size for us and increases the size when needed.

But, to remove the resize operation again and again, we can ensure the desired capacity of arraylist after it has been created using **ensureCapacity()** method.

This method increases the capacity of the ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument to the method. It internally uses **grow()** method.

An invalid index argument will cause **IndexOutOfBoundsException** error.

*get the index of last occurrence of a element in the ArrayList*. We will be using **ArrayList.lastIndexOf()** method to get the **last index**.

**ArrayList listIterator()** returns a list iterator over the elements in this list. It is a **bi-directional** iterator which is **fail-fast** in nature.

ListIterator is fail fast. It means if we modibt the arraylist after list iterator is created, then it will throw ConcurrentModificationException on next() or previous() method call.

**ListIterator** supports to add and remove elements in the list while we are iterating over it.

 **listIterator.add(Element e)** – The element is inserted immediately before the element that would be returned by next() or after the element that would be returned previous() method.

 **listIterator.remove()** – Removes from the list the last element that was returned by next() or previous() method.

**ArrayList removeAll()** removes all of matching elements that are contained in the specified method argument collection. It removes all occurrences of matching elements, not only first occurrence.

**ArrayList removeIf()** iterate the list and removes all of the elements of this list that satisfy the given predicate.

**ArrayList retainAll()** retains only the elements in this list that are contained in the specified method argument collection. Rest all elements are removed from the list. This method is exact opposite to [removeAll()](https://howtodoinjava.com/java/collections/arraylist/arraylist-removeall/) method.

Internally, the retainAll() method iterate over all elements of arraylist. For each element, it pass element to contains() method of argument collection.

If element is not found in argument collection, it re-arranges the index. If element is found, it retains the element inside backing array.

**ArrayList sort()** method sorts the list according to the order induced by the specified **Comparator** instance. All elements in the list must must be mutually comparable.

Internally, the sort() method uses Arrays.sort() method to compare and sort the elements in the list.

# ArrayList spliterator()

 fail-fast

digits.spliterator();

#### trySplit() – Parallel processing

If you are working on concurrent application and list has large number of elements then it’s good idea to divide the list into two parts and process parallely.

trySplit() method splits the current spliterator into two and returns the new one. The elements it is pointing to are divided into two equal lists.

Keep in mind that the individual Spliterator is not thread safe, by default. It is responsibility of application code to create different threads and hand over the Spliterator instances.

|  |
| --- |
| Supports parallel processing. |

## ArrayList.subList() method

This method returns a view of the portion of this list between the specified fromIndex (inclusive) and toIndex (exclusive).

public class ArrayListExample

{

    public static void main(String[] args)

    {

        ArrayList<Integer> list = new ArrayList<>(Arrays.asList(0,1,2,3,4,5,6,7,8,9));

        list.subList(2, 6).clear();

        System.out.println(list);

    }

}

[0, 1, 6, 7, 8, 9]

String[] array = list.toArray(new String[list.size()]);

Object[] array = list.toArray();

#### List.of() – Immutable list – Java 9

We can use List.of() static factory methods to create immutable lists. Only drawback is that add operation is not supported in these lists.

## ArrayList vs Vector – Thread safety

**Vector is a synchronized collection and ArrayList is not**.

<https://howtodoinjava.com/java/collections/arraylist/arraylist-vs-vector/>

we can use Collections.synchronizedList()

By default when vector needs to increase capacity to add an element (when existing capacity is filled), it increases the capacity by **100%.** It means vector size grows to double of previous capacity. We can overide the default capacity using constructor public Vector(int initialCapacity, int capacityIncrement). Here second argument is the amount by which the capacity is increased when the vector overflows.

In ArrayList, by default capacity grows by 50% of existing capacity. In arraylist, we can define the initial capacity but not the capacity increment factor.

There is one difference on how these colelction handle the iteration while the collection is still modifying by program.

ArrayList provide iterators, which are **fail-fast**. As soon as we modify the arraylist structure (add or remove elements), the iterator will throw **ConcurrentModificationException** error.

Vector provide **iterator** as well as **enumeration**. Iterators are fail-fast by enumerations are not. If we modify the vector during iteration over enumeration, it does not fail.

# LinkedList vs ArrayList

 both implements java.util.List interface and provide capability to store and get objects as in ordered collections using simple API methods. Both are non synchronized classes

Adding element in ArrayList is O(1) operation if it doesn’t require resize of Array. If array is resized then it becomes O(log(n)).

Appending an element in LinkedList is O(1) operation, as it doesn’t require any navigation.

#### 2.2. Remove operation

When we remove an element from ArrayList (in backing array), it moves all elements on right. It makes it close to O(n) in worst case (remove first element) and O(1) in best case (remove last element).

LinkedList remove operation gives O(1) performance because it just need to reset the pointers of previous and next nodes. No copy or movement is required.

#### 2.3. Iteration

Iteration is the O(n) operation for both LinkedList and ArrayList where n is a number of an element.

#### 2.4. Get operation

ArrayList provides get(int index) method which directly find the element at given index location. It is of order **O(1)**.

LinkedList also provide get(int index) method BUT it first traverses all nodes to reach the correct node. It makes the performance variable. In best case it is O(1) and in worst case it is O(n).

## 3. LinkedList vs ArrayList – Conclusion

Until you are not dealing with very high volume of data, both the classes will give you same level of performance. Both provide ordered collection and both are non-synchronized as well.

LinkedList implements Deque interface as well, so it provides **queue** like **FIFO** functionality through methods such as peek() and poll().

As seen in performance comparison, ArrayList is better for storing and accessing data. LinkedList is better for manipulating data.

LinkedList implements it with a **doubly-linked list**. ArrayList implements it with a **dynamically resizing array**. This will lead further differences in performance.

#### Collections.unmodifiableList()

Use Collections.unmodifiableList() to get a immutable list

## 1. Swap two elements in arraylist – Collections.swap()

# serialize and deserialize ArrayList in Java

In Java, **ArrayList** class is serializable by default. It essentially means that we do not need to implement Serializable interface explicitly in order to **serialize ArrayList**.

We can directly use **ObjectOutputStream** to **serialize ArrayList**, and **ObjectInputStream** to deserialize an arraylist object.

**Note** – The elements stored in arraylist should also be serializable, else program will throw NotSerializableException.

## Merge two arraylists without duplicates

1. Use **LinkedHashSet**. A set allow only unique elements. Push both lists in a set and set will represent a list of all unique elements combined.

We are using LinkedHashSet because it will preserve the elements order as well.

1. This is two step process. Remove all elements of first list from second list, and then add first list to second list. It will give use the combined list without duplicate elements.
2. [**Collections.synchronizedList()**](https://docs.oracle.com/javase/10/docs/api/java/util/Collections.html#synchronizedList(java.util.List)) method – It returns synchronized list backed by the specified list.

[**CopyOnWriteArrayList**](https://docs.oracle.com/javase/10/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) class – It is a thread-safe variant of ArrayList. This class uses “snapshot” style iterator method. It uses a reference to the state of the backing array at the point that the iterator was created. This array never changes during the lifetime of the iterator.

The iterator will not reflect additions, removals, or changes to the list since the iterator was created. Element-changing operations on iterators themselves (remove, set, and add) are not supported.

By going through the sourcecode of both methods, we can safely say that clear() method give much better performance because of less number of statements it executes.

removeAll() method lack in performance because of extra call to contains() method.

https://howtodoinjava.com/java/collections/arraylist/synchronize-arraylist/