Array List

- *ArrayList* is a *List* implementation built atop an array that can dynamically grow and shrink as we add/remove elements.
- *ArrayList* implementation has the following properties:

    - Random access takes *O(1)* time
    - Adding an element takes amortized constant time *O(1)*
    - Inserting/Deleting takes *O(n)* time
    - Searching takes *O(n)* time for an unsorted array and *O(log n)* for a sorted one

    Syntax:

```
List<String> list = new ArrayList<>();
```

Constructors in ArrayList

- **Default no-arg constructor** :We can create an empty *ArrayList* instance using the no-arg constructor:

```
List<String> list = new ArrayList<>();
```

- **Constructor Accepting Initial Capacity:**

```
List<String> list = new ArrayList<>(20);
```

- **Constructor Accepting *Collection:***

```
ArrayList list1 = new ArrayList<Integer>();
list1.add(1);

List<Integer> list = new ArrayList<>(list1);
```

<u>Adding Elements to the *ArrayList*</u>

- Using add() method:

```
List<Long> list = new ArrayList<>();

list.add(1L);
list.add(2L);
list.add(1, 3L); // adding to index 1 using overloaded
method

assertThat(Arrays.asList(1L, 3L, 2L), equalTo(list));
```

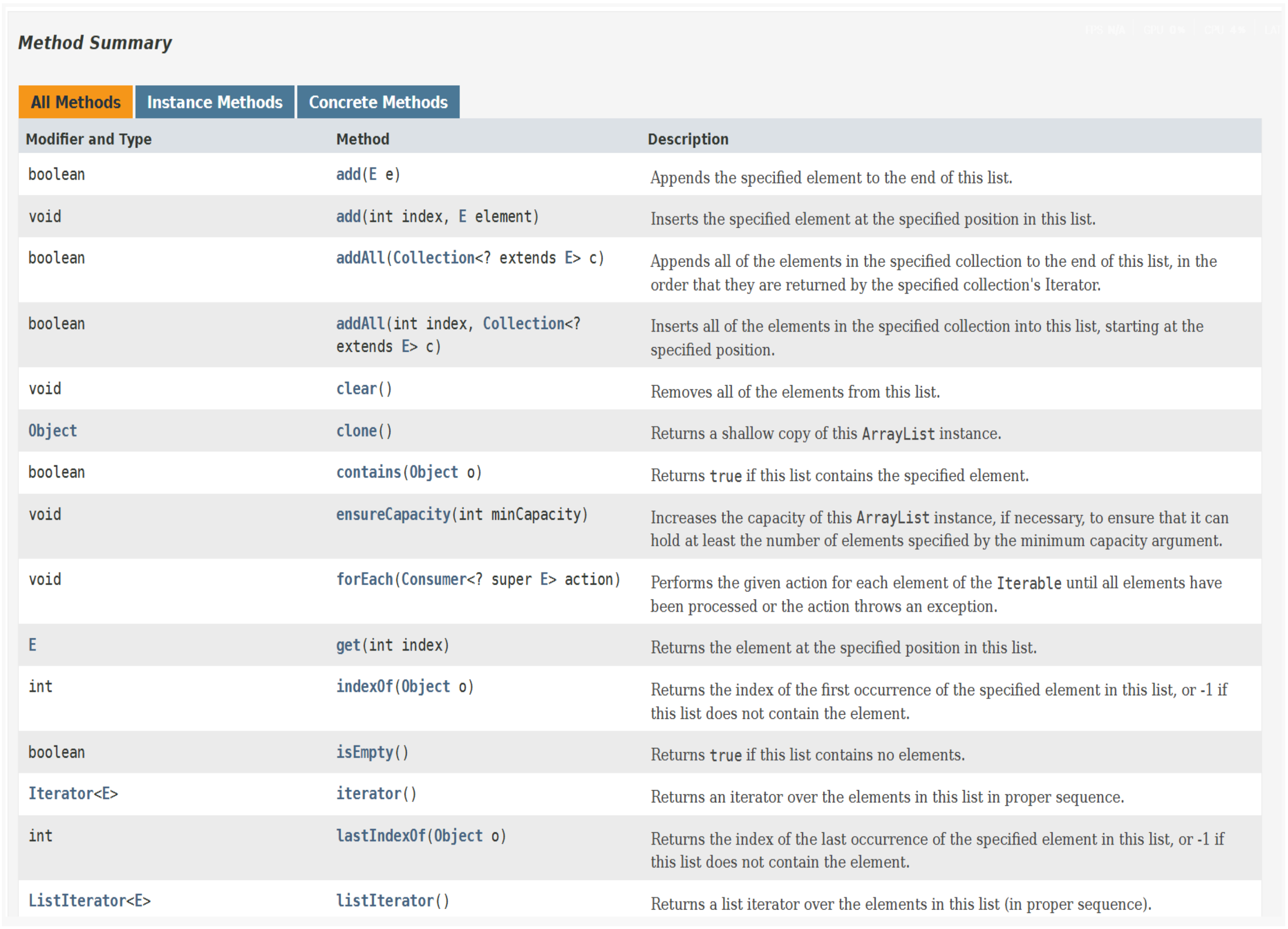
- Adding a Collection using addAll(Collection) method

  ```
  List<String> list1 = new ArrayList<>();
  list1.add("science");
  list1.add("Maths");
  list1.add("Social Science");

  List<String> list2 = new ArrayList<>();

  list2.add("C");
  list2.add("java");
  ```

  Suppose we need list2 to be complete list of list1 and list2,so we will add list1 directly to list2 instead of iterating/looping 3 times.

  ```
  list2.addAll(list1);
  ```

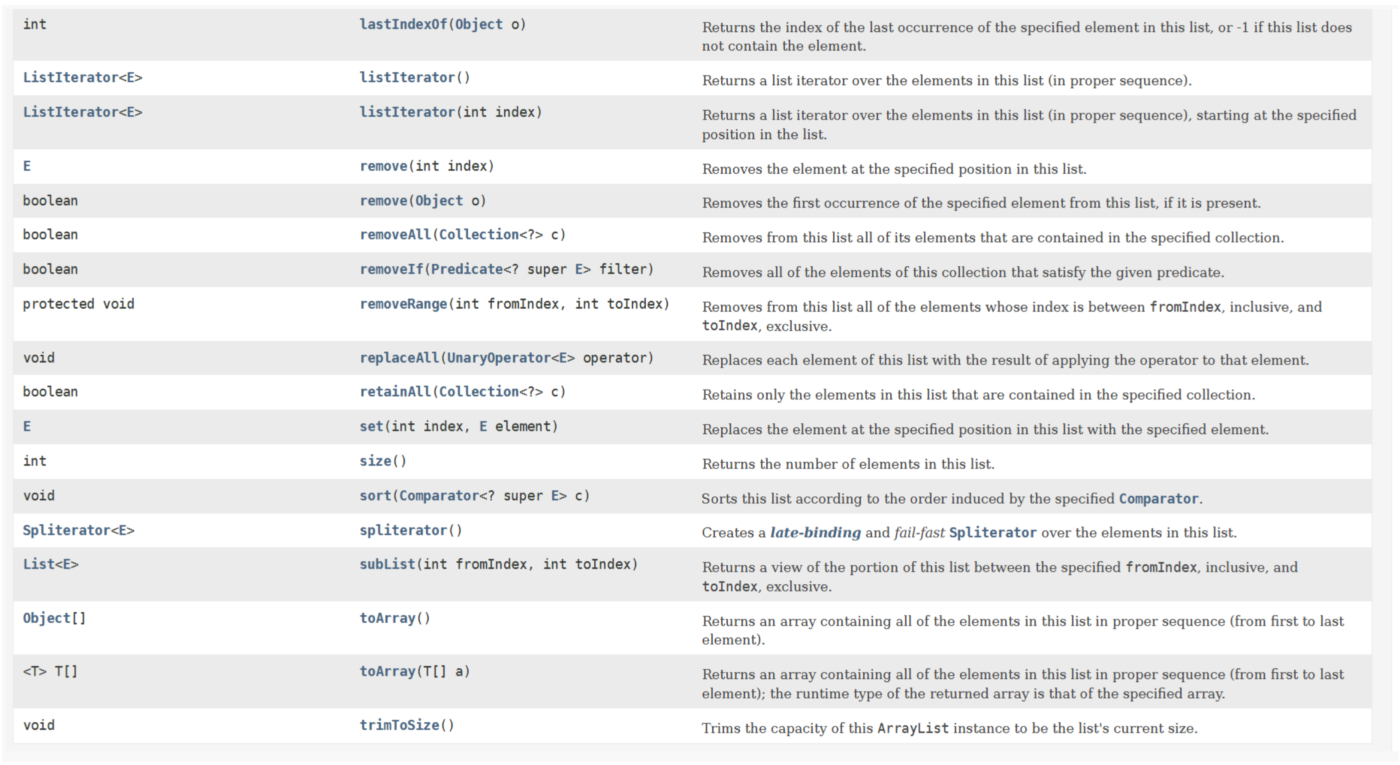
- All the methods in Array list are listed below but do not need to remember all only remember a few which are mostly used like the above once.



Method Summary

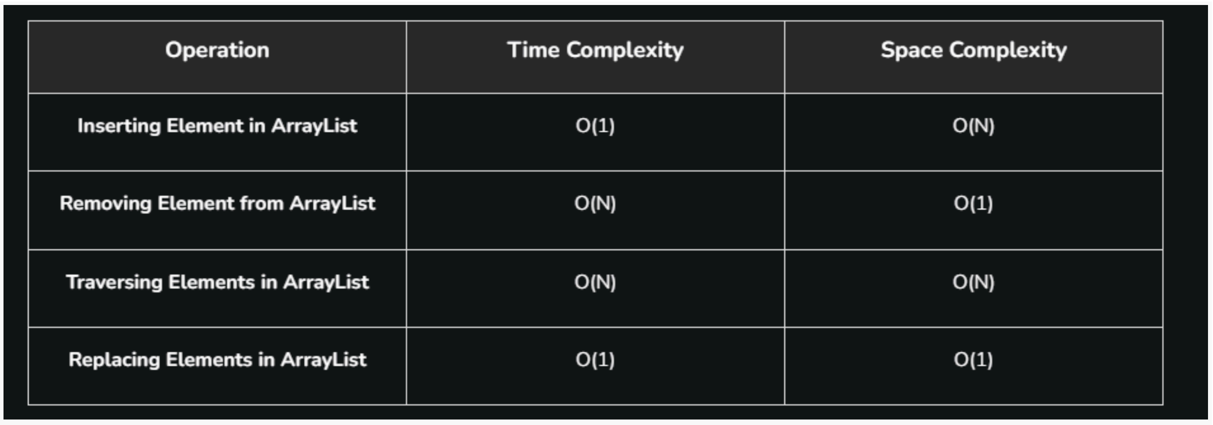
All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
`boolean`	`add(E e)`	Appends the specified element to the end of this list.
`void`	`add(int index, E element)`	Inserts the specified element at the specified position in this list.
`boolean`	`addAll(Collection<? extends E> c)`	Appends 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)`	Inserts all of the elements in the specified collection into this list, starting at the specified position.
`void`	`clear()`	Removes all of the elements from this list.
`Object`	`clone()`	Returns a shallow copy of this `ArrayList` instance.
`boolean`	`contains(Object o)`	Returns `true` if this list contains the specified element.
`void`	`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.
`void`	`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.
`E`	`get(int index)`	Returns the element at the specified position in this list.
`int`	`indexOf(Object o)`	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
`boolean`	`isEmpty()`	Returns `true` if this list contains no elements.
`Iterator<E>`	`iterator()`	Returns an iterator over the elements in this list in proper sequence.
`int`	`lastIndexOf(Object o)`	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
`ListIterator<E>`	`listIterator()`	Returns a list iterator over the elements in this list (in proper sequence).



`int`	`lastIndexOf(Object o)`	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
`ListIterator<E>`	`listIterator()`	Returns a list iterator over the elements in this list (in proper sequence).
`ListIterator<E>`	`listIterator(int index)`	Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
`E`	`remove(int index)`	Removes the element at the specified position in this list.
`boolean`	`remove(Object o)`	Removes the first occurrence of the specified element from this list, if it is present.
`boolean`	`removeAll(Collection<?> c)`	Removes from this list all of its elements that are contained in the specified collection.
`boolean`	`removeIf(Predicate<? super E> filter)`	Removes all of the elements of this collection that satisfy the given predicate.
`protected void`	`removeRange(int fromIndex, int toIndex)`	Removes from this list all of the elements whose index is between `fromIndex`, inclusive, and `toIndex`, exclusive.
`void`	`replaceAll(UnaryOperator<E> operator)`	Replaces each element of this list with the result of applying the operator to that element.
`boolean`	`retainAll(Collection<?> c)`	Retains only the elements in this list that are contained in the specified collection.
`E`	`set(int index, E element)`	Replaces the element at the specified position in this list with the specified element.
`int`	`size()`	Returns the number of elements in this list.
`void`	`sort(Comparator<? super E> c)`	Sorts this list according to the order induced by the specified `Comparator`.
`Spliterator<E>`	`spliterator()`	Creates a ***late-binding*** and *fail-fast* `Spliterator` over the elements in this list.
`List<E>`	`subList(int fromIndex, int toIndex)`	Returns a view of the portion of this list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive.
`Object[]`	`toArray()`	Returns an array containing all of the elements in this list in proper sequence (from first to last element).
`<T> T[]`	`toArray(T[] a)`	Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.
`void`	`trimToSize()`	Trims the capacity of this `ArrayList` instance to be the list's current size.

Complexity of ArrayList



Operation	Time Complexity	Space Complexity
Inserting Element in ArrayList	O(1)	O(N)
Removing Element from ArrayList	O(N)	O(1)
Traversing Elements in ArrayList	O(N)	O(N)
Replacing Elements in ArrayList	O(1)	O(1)