**What is need for Collections in Java?**

Before going to understand the need for collections in java, first, we understand different ways to store values in Java application by JVM. There is a total of four ways to store values in Java application by JVM.

**1. Using variable approach:**

Suppose you want to store one value in the memory then you will create one variable .

The purpose of variable x is to store one int value 10. If we want to store two int values, we will create two variables like this:

int x = 10;

int y = 20;

Similarly, for three values, the third variable will be required, and so on.

There is no problem to store until the third or fourth value. But if we want to store 5000 values then declaring 5000 variables in a program is the worst kind of programming practice and it will not be in the readable format.

Thus, the limitations of using the variable approach are as follows:

➲ The limitation of a variable is that it can store only one value at a time.

➲ The readability and reusability of the code will be down.

➲ JVM will take more time for execution.

Hence, the problem facing this approach can be overcome by using the second approach “Class object”.

**2. Using class object approach:**

Using a class object, we can store multiple “fixed” number of values of different types.

For example, suppose we have created a class named Employee and declare two variables inside the class. Look at the code.

class Employee {

      int eNo;

      String eName;

}

// Creating an object of Employee class.

Employee e1 = new Employee();

So can you think how many values can store in this employee object?

The answer is only two but if you will want to store the third value, it will not possible. Therefore, this approach is only suitable to store a fixed number of different values. To overcome this problem, we should use the third technique “Array object”.

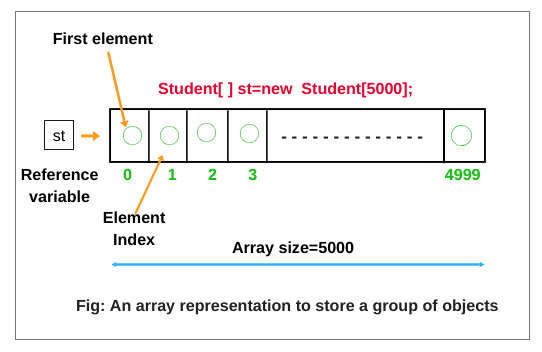
1. **Using array object approach:**

We can store a group of objects into an array.

Let’s take an example to understand it. Suppose we want to store 5000 objects of Student class into an array.

For this purpose, we need to create an array object of Student type like this:

Student[ ] st = new Student[5000];

This array can store 5000 Student objects.[](https://www.scientecheasy.com/2020/09/java-collections-framework.html/)

The biggest advantage of an array is that we can store a huge number of values by using a single variable st and retrieve them easily.

Array mechanism helps to improve the readability of the code in java programming but there are several types of problems and limitations with the array. They are as follows:

1. We can easily store multiple “fixed” numbers of values of homogeneous data type i.e. Array can store only similar types of data. Suppose if we create an Employee type array object.

Employee[ ] emp = new Employee[5000]; // It will hold only employee type objects.

For example:

       emp[0] = new Employee(); // valid.

       emp[1] = new Customer(); // invalid because here, we are providing the customer type object.

We can resolve this problem by using an object array.

Object[ ] ob = new Object[5000];

     ob[0] = new Employee(); // valid.

     ob[1] = new Customer(); // valid.

From the above code, it is clear that we cannot store different class objects into the same array. This is because an array can store only one data type of elements (objects).

2. An array is static in nature. It is fixed in length and size. We cannot change (increase/decrease) the size of the array based on our requirements once they created.

Hence, to use an array, we must know the size of an array to store a group of objects in advance, which may not always be possible.

3. We can add elements at the end of an array easily. But, adding and deleting elements or objects in the middle of array is difficult.

4. We cannot insert elements in some sorting order using array concept because array does not support any method. We will have to write the sorting code for this but in the case of collection, ready-made method support is available for sorting using Tree set.

5. We cannot search a particular element using an array, whether the particular element is present or not in the array index. For this, we will have to write the searching code using array but in the case of collection, one readymade method called contains() method is available.

Due to all these above limitations of array, programmers need a better mechanism to store a group of objects. So, the alternative option is a collection object or container object in java.

**4. Using collection object:**

By using collection object, we can store the same or different data without any size limitation. Thus, technically, we can define the collections as:

A **collection in java** is a container object that is used for storing multiple homogeneous and heterogeneous, duplicate, and unique elements without any size limitation.

COLLECTIONS:

# A collection is a group of objects. In Java, these objects are called elements of the collection. Let’s understand it with some realtime examples.

## 1. In childhood, you had a kiddy bank. In the kiddy bank, you had collected a lot of coins. This kiddy bank is called collection and the coins are nothing but objects.

## 2. During school time, you put all the important books in the school bag before going to school. Here Schoolbag is a collection and books are objects.

## 3. A classroom is a collection of students. So, the classroom is nothing but a collection and students are objects.

## 4. We know that the entire world is a collection of humans, animals, and different things. The world is a collection and humans, animals and different things are different objects.

## Without using collection concepts, you cannot develop any production level software application in Java.

## Therefore, every Java learner has to learn the collections framework.

#### What is a framework in Java

* It provides readymade architecture.
* It represents a set of classes and interfaces.
* It is optional.

#### What is Collection framework

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

Interfaces and its implementations, i.e., classes

Algorithm

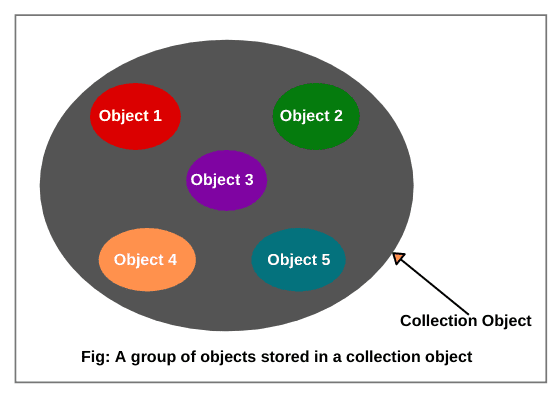
# Java Collection framework provides many

# interfaces (Set, List, Queue, Deque) and

# classes ([ArrayList](https://www.javatpoint.com/java-arraylist), Vector, [LinkedList](https://www.javatpoint.com/java-linkedlist), [PriorityQueue](https://www.javatpoint.com/java-priorityqueue), HashSet, LinkedHashSet, TreeSet).

# [JVM](https://www.scientecheasy.com/2021/03/what-is-jvm.html/) (Java Virtual Machine) stores the reference of other objects into a collection object.

A group of objects stored in a collection object is shown in the below figure. In the figure, we are storing 5 objects in a collection object.

[](https://www.scientecheasy.com/2020/09/java-collections-framework.html/)

A collection object has a class that is known as collection class or container class.

All collection classes are present in java.util package. Here, util stands for utility.

A group of collection classes is called collections framework in java.

**Types of Objects Stored in Collection (Container) Object**

There are two types of objects that can be stored in a collection or container object. They are as follows:

**1. Homogeneous objects:**

Homo means same. Homogeneous objects are a group of multiple objects that belong to the same class.

For example, suppose we have created three objects Student s1, Student s2, and Student s3 of the same class ‘Student’. Since these three objects belong to the same class that’s why they are called homogeneous objects.

**2. Heterogeneous objects:**

Hetero means different. Heterogeneous objects are a group of different objects that belong to different classes.

For example, suppose we have created two different objects of different classes such as one object Student s1, and another one object Employee e1. Here, student and employee objects together are called a collection of heterogeneous objects.

These objects can also be further divided into two types. They are as follows:

**1. Duplicate objects:**

The multiple objects of a class that contains the same data are called duplicate objects. For example, suppose we create two person objects Person p1 and Person p2. Both of these objects have the same data.

Person p1 = new Person( "abc");

Person p2 = new Person("abc");

Since the above two objects have the same data “abc” therefore, these are called duplicate objects.

**2. Unique objects:**

The multiple objects of a class that contains different data are called unique objects. For example:

Person p1 = new Person("abcd");

Person p2 = new Person("abcde");

A unique or duplicate object depends on its internal data.

## Advantage of Collections Framework in Java

The advantages of the collections framework in java are as follows:

1. The collections framework reduces the development time and the burden of designers, programmers, and users.
2. Your code is easier to maintain because it provides useful data structure and interfaces which reduce programming efforts.
3. The size of the container is growable in nature.
4. It implements high-performance of useful data structures and algorithms that increase the performance.
5. It enables software reuse.

## Limitation of Collections Framework in Java

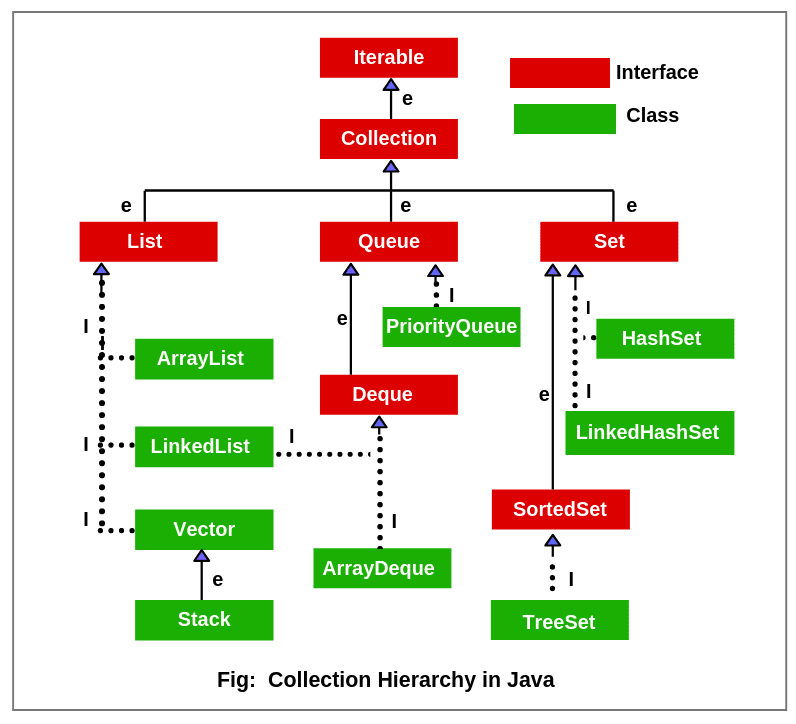
There are two limitations of the collections framework in Java. They are as follows:

1. Care must be taken to use the appropriate cast operation.
2. Compile type checking is not possible.

# Collection Hierarchy in Java

The hierarchy of the entire collection framework consists of four core interfaces such as Collection, List, Set, Map, and two specialized interfaces named SortedSet and SortedMap for sorting.

All the interfaces and classes for the collection framework are located in [java.util package](https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html). The diagram of Java collection hierarchy is shown in the below figure.

[](https://www.scientecheasy.com/2020/09/collection-hierarchy-in-java.html/)

e➝ extends, I➝ implements

**Extends:** Extends is a keyword that is used for developing inheritance between two classes and two interfaces.

**Implements:** Implements is a keyword used for developing inheritance between class and interface.

Iterable Interface

The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract method. i.e.,

|  |
| --- |
| Iterator interface provides the facility of iterating the elements in a forward direction only.  The iterator() method does not take any parameters.  returns an iterator to loop through the arraylist elements  **Note**: The iterator returned by the method is stored in the variable of interface Iterator type. |

**Methods of Iterator interface**

There are only three methods in the Iterator interface. They are:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | Public boolean hasNext() | It returns true if the iterator has more elements otherwise it returns false. |
| 2 | public Object next() | It returns the element and moves the cursor pointer to the next element. |
| 3 | public void remove() | It removes the last elements returned by the iterator. It is less used. |

# iterator()

The Java ArrayList iterator() method returns an iterator to access each element of the arraylist in a proper sequence.

**The syntax of the iterator() method is:**

arraylist.iterator()

Example 1: Java ArrayList iterator()

import java.util.ArrayList;

import java.util.Iterator;

class Main {

public static void main(String[] args){

ArrayList<String> languages = new ArrayList<>();

// Add elements in the array list

languages.add("Java");

languages.add("Python");

languages.add("JavaScript");

languages.add("Swift");

// Create a variable of Iterator

// store the iterator returned by iterator()

Iterator<String> iterate = languages.iterator();

System.out.print("ArrayList: ");

// loop through ArrayList till it has all elements

// Use methods of Iterator to access elements

while(iterate.hasNext()){

System.out.print(iterate.next());

System.out.print(", ");

}

}

}

**Output**

ArrayList: Java, Python, JavaScript, Swift,

Here, we have created a variable named iterate of the Iterator interface. The variable stores the iterator returned by the iterator() method.

Using iterate, we can access the elements of the arraylist.

* **hasNext**: returns true if there is a next element in the arraylist
* **next()**: returns the next element in the arraylist

## Iterate through an ArrayList

We can use the [Java for-each loop](https://www.programiz.com/java-programming/enhanced-for-loop) to loop through each element of the arraylist. For example,

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// creating an array list

ArrayList<String> animals = new ArrayList<>();

animals.add("Cow");

animals.add("Cat");

animals.add("Dog");

System.out.println("ArrayList: " + animals);

// iterate using for-each loop

System.out.println("Accessing individual elements: ");

for (String language : animals) {

System.out.print(language);

System.out.print(", ");

}

}

}

**Output**

ArrayList: [Cow, Cat, Dog]

Accessing individual elements:

Cow, Cat, Dog,

## Collection Interface in Java

➲ The basic interface of the collections framework is the Collection interface which is the root interface of all collections in the API (Application programming interface).

It is placed at the top of the collection hierarchy in java. It provides the basic operations for adding and removing elements in the collection.

The Collection interface extends the Iterable interface. The iterable interface has only one method called iterator(). The function of the iterator method is to return the iterator object. Using this iterator object, we can iterate over the elements of the collection.

➲ [List](https://www.scientecheasy.com/2020/09/java-list-interface.html/), Queue, and [Set](https://www.scientecheasy.com/2020/09/java-set.html/) have three component which extends the Collection interface. A map is not inherited by Collection interface.

**List Interface**

➲ A **list in Java** is a collection for storing elements in sequential order. Sequential order means the first element, followed by the second element, followed by the third element, and so on.

➲ Java list is a sub-interface of the collection interface that is available in java.util package. Sub interface means an interface that extends another interface is called sub interface. Here, the list interface extends the collection interface.

➲ List maintains an order of elements means the order is retained in which we add elements, and the same sequence we will get while retrieving elements.

➲ We can insert elements into the list at any location. The list allows storing duplicate elements in Java.

➲ [ArrayList](https://www.scientecheasy.com/2020/09/arraylist-in-java.html/), [vector](https://www.scientecheasy.com/2020/09/vector-in-java.html/), and [LinkedList](https://www.scientecheasy.com/2020/09/java-linkedlist.html/) are three concrete subclasses that implement the list interface.

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

To instantiate the List interface, we must use :

1. List <data-type> list1= **new** ArrayList();
2. List <data-type> list2 = **new** LinkedList();
3. List <data-type> list3 = **new** Vector();
4. List <data-type> list4 = **new** Stack();

There are various methods in List interface that can be used to insert, delete, and access the elements from the list.

## How to create Generic List Object in Java

After the introduction of Generic in Java 1.5, we can restrict the type of object that can be stored in the list. The general syntax for creating a list of objects with a generic type parameter is as follows:

1. List<data type> list = new ArrayList<data type>(); // General sysntax.

For example:

   a. List<String> list = new ArrayList<String>(); // Creating a list of objects of String type using ArrayList.

   b. List<Integer> list = new LinkedList<Integer>(); Creating a list of objects of Integer type using LinkedList.

   c. List<String> list1 = new LinkedList<String>();

   d. List<obj> list = new ArrayList<obj>(); // obj is the type of object.

For example:

    List<Book> list=new ArrayList<Book>(); // Book is the type of object.

2. Starting from Java 1.7, we can use a diamond operator.

    a. List<String> str = new ArrayList<>();

    b. List<Integer> list = new LinkedList<>();

## Java List Initialization

After creating a list, we need to initialize the list by adding elements to it. There are three methods to initialize the list. They are as follows:

* Using Arrays.asList
* Using Normal way
* Using Anonymous Inner class

ollowing are few key points to note about ArrayList in Java -

* An ArrayList is a re-sizable array, also called a dynamic array. It grows its size to accommodate new elements and shrinks the size when the elements are removed.
* ArrayList internally uses an array to store the elements. Just like arrays, It allows you to retrieve the elements by their index.
* Java ArrayList allows duplicate and null values.
* Java ArrayList is an ordered collection. It maintains the insertion order of the elements.
* You cannot create an ArrayList of primitive types like int, char etc. You need to use boxed types like Integer, Character, Boolean etc.
* Java ArrayList is not synchronized. If multiple threads try to modify an ArrayList at the same time, then the final outcome will be non-deterministic. You must explicitly synchronize access to an ArrayList if multiple threads are gonna modify it.

**ArrayList**

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed.

**Creating an ArrayList**

Before using ArrayList, we need to import the java.util.ArrayList package first. Here is how we can create arraylists in Java:

ArrayList<Type> arrayList= new ArrayList<>();

Here, Type indicates the type of an arraylist.

For example,

// create Integer type arraylist

ArrayList<Integer> arrayList = new ArrayList<>();

// create String type arraylist

ArrayList<String> arrayList = new ArrayList<>();

In the above program, we have used Integer not int. It is because we cannot use primitive types while creating an arraylist. Instead, we have to use the corresponding wrapper classes.

Here, Integer is the corresponding wrapper class of int

### Example: Create ArrayList in Java

import java.util.ArrayList;

class Main {

public static void main(String[] args){

// create ArrayList

ArrayList<String> languages = new ArrayList<>();

// Add elements to ArrayList

languages.add("Java");

languages.add("Python");

languages.add("Swift");

System.out.println("ArrayList: " + languages);

}

}

**Output**

ArrayList: [Java, Python, Swift]

In the above example, we have created an ArrayList named languages.

Here, we have used the add() method to add elements to the arraylist.

**Basic Operations on ArrayList**

The ArrayList class provides various methods to perform different operations on arraylists. We will look at some commonly used arraylist operations in this tutorial:

* Add elements
* Access elements
* Change elements
* Remove elements

### 1. Add Elements to an ArrayList

To add a single element to the arraylist, we use the add() method of the ArrayList class. For example,

**Syntax of ArrayList add()**

The syntax of the add() method is:

arraylist.add(int index, E element)

Here, arraylist is an object of ArrayList class.

**add() Parameters**

The ArrayList add() method can take two parameters:

* **index** (optional) - index at which the element is inserted
* **element** - element to be inserted

If the index parameter is not passed, the element is appended to the end of the arraylist

import java.util.ArrayList;

class Main {

public static void main(String[] args){

// create ArrayList

ArrayList<String> languages = new ArrayList<>();

// add() method without the index parameter

languages.add("Java");

languages.add("C");

languages.add("Python");

System.out.println("ArrayList: " + languages);

}

}

**Output**

ArrayList: [Java, C, Python]

**. boolean add(Object o):** It starts to add the specified element from zero location. If the element is already present at zero location, it will add the next element in one position.

**void add(int index, Object o):** This method adds/inserts the specified element at a particular position in the list. For example, suppose we want to add element “d” at 3rd position, we will call add(int index, Object o ) method like this:

list.add(3,"d"); // It will add element "d" at 3rd position as shown in figure.

# addAll()

The addAll() method adds all the elements of a collection to the arraylist.

## Syntax of ArrayList addAll()

The syntax of the addAll() method is:

arraylist.addAll(int index, Collection c)

Here, arraylist is an object of the ArrayList class.

**addAll() Parameters**

The ArrayList addAll() method can take two parameters:

* **index** (optional) - index at which all elements of a collection is inserted
* **collection** - collection that contains elements to be inserted

If the index parameter is not passed the collection is appended at the end of the arraylist.

### Example

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an arraylist

ArrayList<String> languages = new ArrayList<>();

languages.add("Java");

languages.add("Python");

System.out.println("Languages: " + languages);

// create another arraylist

ArrayList<String> programmingLang = new ArrayList<>();

// add all elements from languages to programmingLang

programmingLang.addAll(languages);

System.out.println("Programming Languages: " + programmingLang);

}

}

// Output: Languages: [Java, Python]

// Programming Languages: [Java, Python]

**boolean addAll(Collection c):** Here, Collection c represents a group of elements. This method is used to add/insert a group of elements at the end of the last element.

**. boolean addAll(int index, Collection c):** This method is used to add/insert a group of elements at a particular position in the list and shift the subsequent elements to the right by increasing their indices.

### 2. Access ArrayList Elements

To access an element from the arraylist, we use the get() method of the ArrayList class.

**object get(int index):** This method is used to return element/object stored at a specified position in the list. The return type of get() method is Object and input type is int[index of List].

# Java ArrayList get()

The Java ArrayList get() method returns the element present in specified position.

The syntax of the get() method is:

arraylist.get(int index)

Here, arraylist is an object of the ArrayList class.

## get() Parameter

The get() method takes single parameter.

* **index** - position of the element to be accessed
* In the example, we have used the get() method with parameter 1. Here, the method returns the element at **index 1**.
* import java.util.ArrayList;
* class Main {
* public static void main(String[] args) {
* ArrayList<String> animals = new ArrayList<>();
* // add elements in the arraylist
* animals.add("Cat");
* animals.add("Dog");
* animals.add("Cow");
* System.out.println("ArrayList: " + animals);
* // get the element from the arraylist
* String str = animals.get(1);
* System.out.print("Element at index 1: " + str);
* }
* }
* **Output**
* ArrayList: [Cat, Dog, Cow]

Element at index 1: Dog

### 3. Change ArrayList Elements

To change elements of the arraylist, we use the set() method of the ArrayList class.

**object set(int index, Object o):** This method replaces the existing element at the specified position in the list with new specified element.

The Java ArrayList set() method replaces the element present in a specified position with the specified element in an arraylist.

The syntax of the set() method is:

arraylist.set(int index, E element)

Here, arraylist is an object of the ArrayList class.

set() Parameters

The set() method takes two parameters.

* **index** - position of the element to be replaced
* **element** - new element that is to be stored at index
* For example,
* import java.util.ArrayList;
* class Main {
* public static void main(String[] args) {
* ArrayList<String> languages = new ArrayList<>();
* // add elements in the array list
* languages.add("Java");
* languages.add("Kotlin");
* languages.add("C++");
* System.out.println("ArrayList: " + languages);
* // change the element of the array list
* languages.set(2, "JavaScript");
* System.out.println("Modified ArrayList: " + languages);
* }
* }
* **Output**
* ArrayList: [Java, Kotlin, C++]
* Modified ArrayList: [Java, Kotlin, JavaScript]
* In the above example, we have created an ArrayList named languages. Notice the line,
* language.set(2, "JavaScript");

## ArrayList set() Vs. add()

The syntax of the add() and set() method looks quite similar.

// syntax of add()

arraylist.add(int index, E element)

// syntax of set()

arraylist.set(int index, E element)

And, both the methods are adding a new element to the arraylist. This is why some people consider both methods similar.

However, there is a major difference between them.

* The set() method adds a new element at the specified position by replacing the older element at that position.
* The add() method adds a new element at the specified position by shifting the older element towards the right position.

**Example 2: ArrayList set() Vs. add()**

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<String> languages1 = new ArrayList<>();

// add elements to ArrayList

languages1.add("Python");

languages1.add("English");

languages1.add("JavaScript");

// create another ArrayList similar to languages1

ArrayList<String> languages2 = new ArrayList<>();

// adds all elements from languages1 to languages2

languages2.addAll(languages1);

System.out.println("ArrayList: " + languages1);

// use of set()

languages1.set(1, "Java");

System.out.println("ArrayList after set(): " + languages1);

// use of add()

languages2.add(1, "Java");

System.out.println("ArrayList after add(): " + languages2);

}

}

**Output**

ArrayList: [Python, English, JavaScript]

ArrayList after set(): [Python, Java, JavaScript]

ArrayList after add(): [Python, Java, English, JavaScript]

# ArrayList indexOf()

The Java ArrayList indexOf() method returns the position of the specified element in the arraylist.

**. int indexOf(Object o):** It is used to return the index of a particular element of the first occurrence in the list. If the element is not present in the list then it will return -1. It takes as an argument as an element and returns as an integer value of that element as it is index value.

The syntax of the indexOf() method is:

arraylist.indexOf(Object obj)

Here, arraylist is an object of the ArrayList class.

indexOf() Parameter

The indexOf() method takes a single parameter.

* **obj** - element whose position is to be returned

If the same element obj is present in multiple location, then the position of the element that appears first in the arraylist is returned.

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<Integer> numbers = new ArrayList<>();

// insert element to the arraylist

numbers.add(22);

numbers.add(13);

numbers.add(35);

System.out.println("Number ArrayList: " + numbers);

// find the position of 13

int position1 = numbers.indexOf(13);

System.out.println("Index of 13: " + position1);

// find the position of 50

int position2 = numbers.indexOf(50);

System.out.println("Index of 50: " + position2);

}

}

**Output**

Number ArrayList: [22, 13, 35]

Index of 13: 1

Index of 50: -1

## indexOf() Return Value

* returns the position of the specified element from the arraylist

**Note**: If the specified element doesn't exist in the list, the indexOf() method returns **-1**.

In the above example, we have created an arraylist named numbers. Notice the expressions,

// returns 1

numbers.indexOf(13)

// returns -1

numbers.indexOf(50)

Here, the indexOf() method successfully returns the position of element **13**. However, the element **50** doesn't exist in the arraylist. Hence, the method returns **-1**.

## Example 2: Get the Position of the First Occurrence of an Element

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<String> languages = new ArrayList<>();

// insert element to the arraylist

languages.add("JavaScript");

languages.add("Python");

languages.add("Java");

languages.add("C++");

languages.add("Java");

System.out.println("Programming Languages: " + languages);

// get the position of Java

int position = languages.indexOf("Java");

System.out.println("First Occurrence of Java: " + position);

}

}

**Output**

Programming Languages: [JavaScript, Python, Java, C++, Java]

First Occurrence of Java: 2

In the above example, we have created an arraylist named languages. Here, we have used the indexOf() method to get the position of the element Java.

However, Java is present in two different locations in the list. In this case, the method returns the position of Java, where it appears for the first time (i.e **2**).

### And, if we want to get the last occurrence of Java, we can use the lastIndexOf() method.

### 4. Remove ArrayList Elements

To remove an element from the arraylist, we can use the remove() method of the ArrayList class.

In this tutorial, we will learn about the Java ArrayList remove() method with the help of examples.

The remove() method removes the single element from the arraylist.

## Syntax of ArrayList remove()

The syntax of the remove() method is:

// remove the specified element

arraylist.remove(Object obj)

// remove element present in the specified index

arraylist.remove(int index)

Here, arraylist is an object of the ArrayList class.

## remove() Parameters

The remove() method takes a single parameter.

* **obj** - element that is to be removed from the arraylist, **OR**
* **index** - position from where element is to be removed

If the same element obj is present in multiple location, then the element that appear first in the arraylist is removed.

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<Integer> primeNumbers = new ArrayList<>();

primeNumbers.add(2);

primeNumbers.add(3);

primeNumbers.add(5);

System.out.println("ArrayList: " + primeNumbers);

// remove element at index 2

int removedElement = primeNumbers.remove(2);

System.out.println("Removed Element: " + removedElement);

}

}

// Output: ArrayList: [2, 3, 5]

// Removed Element: 5

**object remove(int index):** It is used to remove an element at a specified position in the list.

ArrayList clear()

The Java ArrayList clear() method removes all the elements from an arraylist.

The syntax of the clear() method is:

arraylist.clear()

Here, arraylist is an object of the ArrayList class.

clear() Parameters

The clear() method does not take any parameters.

clear() Return Value

The clear() method does not return any value. Rather, it makes changes to the arraylist.

Example 1: import java.util.ArrayList;

class Main {

public static void main(String[] args){

// create an arraylist

ArrayList<String> languages = new ArrayList<>();

languages.add("Java");

languages.add("JavaScript");

languages.add("Python");

System.out.println("Programming Languages: " + languages);

// remove all elements

languages.clear();

System.out.println("ArrayList after clear(): " + languages);

}

}

**Output**

Programming Languages: [Java, JavaScript, Python]

ArrayList after clear(): []

# ArrayList removeAll()

The Java ArrayList removeAll() method removes all the elements from the arraylist that are also present in the specified collection.

The syntax of the removeAll() method is:

arraylist.removeAll(Collection c);

Here, arraylist is an object of the ArrayList class.

## removeAll() Parameters

The removeAll() method takes a single parameter.

* **collection** -all elements present in collection are deleted from the arraylist.

## removeAll() Return Value

* returns true if elements are deleted from the arraylist
* throws ClassCastException if the class of elements present in arraylist is incompatible with the class of elements in specified **collection**
* throws NullPointerException if the arraylist contains null element and the specified **collection** does not allow null elements

## Example 1: Remove all elements from an ArrayList

import java.util.ArrayList;

class Main {

public static void main(String[] args){

// create an arraylist

ArrayList<String> languages = new ArrayList<>();

// add elements to arraylist

languages.add("Java");

languages.add("JavaScript");

languages.add("Python");

System.out.println("Programming Languages: " + languages);

// remove all elements from arraylist

languages.removeAll(languages);

System.out.println("ArrayList after removeAll(): " + languages);

}

}

**Output**

Programming Languages: [Java, JavaScript, Python]

ArrayList after removeAll(): []

In the above example, we have created an arraylist named languages. The arraylist stores the name of programming languages. Notice the line,

languages.removeAll(languages);

Here, we are passing the ArrayList languages as an argument of the removeAll() method. Hence, the method removes all the elements from the arraylist.

**Note**: The clear() method is preferred to remove all elements from the arraylist.

## ArrayList clear() Vs. removeAll()

The ArrayList also provides the removeAll() method that also remove all elements from the arraylist. For example,

import java.util.ArrayList;

class Main {

public static void main(String[] args){

// create an arraylist

ArrayList<Integer> oddNumbers = new ArrayList<>();

// add elements to arraylist

oddNumbers.add(1);

oddNumbers.add(3);

oddNumbers.add(5);

System.out.println("Odd Number ArrayList: " + oddNumbers);

// remove all elements

oddNumbers.removeAll(oddNumbers);

System.out.println("ArrayList after removeAll(): " + oddNumbers);

}

}

**Output**

Odd Number ArrayList: [1, 3, 5]

ArrayList after removeAll(): []

In the above example, we have created an arraylist named oddNumbers. Here, we can see that the removeAll() method is used to remove all the elements from the arraylist.

Both the removeAll() and clear() method are performing the same task. However, the clear() method is used more than removeAll(). It is because clear() is faster and efficient compared to removeAll().

## Methods of ArrayList Class

In the previous section, we have learned about the add(), get(), set(), and remove() method of the ArrayList class.

Besides those basic methods, here are some more ArrayList methods that are commonly used.

|  |  |
| --- | --- |
| Methods | Descriptions |
| [size()](https://www.programiz.com/java-programming/library/arraylist/size) | Returns the length of the arraylist. |
| [sort()](https://www.programiz.com/java-programming/library/arraylist/sort) | Sort the arraylist elements. |
| [clone()](https://www.programiz.com/java-programming/library/arraylist/clone) | Creates a new arraylist with the same element, size, and capacity. |
| [contains()](https://www.programiz.com/java-programming/library/arraylist/contains) | Searches the arraylist for the specified element and returns a boolean result. |
| [ensureCapacity()](https://www.programiz.com/java-programming/library/arraylist/ensurecapacity) | Specifies the total element the arraylist can contain. |
| [isEmpty()](https://www.programiz.com/java-programming/library/arraylist/isempty) | Checks if the arraylist is empty. |
| [indexOf()](https://www.programiz.com/java-programming/library/arraylist/indexof) | Searches a specified element in an arraylist and returns the index of the element. |

# Java ArrayList size()

In this tutorial, we will learn about the Java ArrayList size() method with the help of examples.

The size() method returns the number of elements present in the arraylist.

## Syntax of ArrayList size()

The syntax of the size() method is:

arraylist.size()

Here, arraylist is an object of the ArrayList class.

## size() Parameters

The size() method does not take any parameters.

## size() Return Value

* returns the number of elements present in the arraylist.

## Example: Get the Length of an ArrayList

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<String> languages = new ArrayList<>();

// insert element to the arraylist

languages.add("JavaScript");

languages.add("Java");

languages.add("Python");

System.out.println("ArrayList: " + languages);

// get the number of elements of arraylist

int size = languages.size();

System.out.println("Length of ArrayList: " + size);

}

}

**Output**

ArrayList: [JavaScript, Java, Python]

Length of ArrayList: 3

In the above example, we have created an arraylist named languages. Here, we have used the size() method to get the number of elements present in the arraylist.

### Example

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<Integer> primeNumbers = new ArrayList<>();

primeNumbers.add(2);

primeNumbers.add(3);

primeNumbers.add(5);

primeNumbers.add(7);

System.out.println("ArrayList: " + primeNumbers);

// get the number of elements of arraylist

int size = primeNumbers.size();

System.out.println("Length of ArrayList: " + size);

}

}

// Output: ArrayList: [2, 3, 5, 7]

// Length of ArrayList: 4

# Java ArrayList sort()

In this tutorial, we will learn about the Java ArrayList sort() method with the help of examples.

The sort() method sorts the elements in an arraylist according to the specified order.

## Syntax of ArrayList sort()

The syntax of the sort() method is:

arraylist.sort(Comparator c)

Here, arraylist is an object of the ArrayList class.

## sort() Parameters

The sort() method takes a single parameter.

* **comparator** - specifies the sort order of the arraylist

## sort() Return Values

The sort() method does not return any value. Rather it only changes the order of elements in an arraylist.

### Example

import java.util.ArrayList;

import java.util.Comparator;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(7);

numbers.add(3);

numbers.add(9);

numbers.add(-33);

System.out.println("Unsorted ArrayList: " + numbers);

// sort the ArrayList in ascending order

numbers.sort(Comparator.naturalOrder());

System.out.println("Sorted ArrayList: " + numbers);

}

}

// Output: Unsorted ArrayList: [7, 3, 9, -33]

// Sorted ArrayList: [-33, 3, 7, 9]

## .,,., Example 1: Sort the ArrayList in Natural Order

import java.util.ArrayList;

import java.util.Comparator;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<String> languages = new ArrayList<>();

// add elements to ArrayList

languages.add("Python");

languages.add("Swift");

languages.add("C");

languages.add("JavaScript");

System.out.println("Unsorted ArrayList: " + languages);

// sort the ArrayList in ascending order

languages.sort(Comparator.naturalOrder());

System.out.println("Sorted ArrayList: " + languages);

}

}

**Output**

Unsorted ArrayList: [Python, Swift, C, JavaScript]

Sorted ArrayList: [C, JavaScript, Python, Swift]

In the above example, we have used the sort() method to sort the arraylist named languages. Notice the line,

languages.sort(Comparator.naturalOrder());

Here, the naturalOrder() method of the [Java Comparator Interface](https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html) specifies that elements are sorted in natural order (i.e. ascending order).

The Comparator interface also provides a method to sort elements in descending order. For example,

## Example 2: Sort the ArrayList in Reverse Order

import java.util.ArrayList;

import java.util.Comparator;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<String> languages = new ArrayList<>();

// add elements to ArrayList

languages.add("Python");

languages.add("Swift");

languages.add("C");

languages.add("JavaScript");

System.out.println("Unsorted ArrayList: " + languages);

// sort the ArrayList in ascending order

languages.sort(Comparator.reverseOrder());

System.out.println("Sorted ArrayList: " + languages);

}

}

**Output**

Unsorted ArrayList: [Python, Swift, C, JavaScript]

Sorted ArrayList: [Swift, Python, JavaScript, C]

Here, the reverseOrder() method of the Comparator interface specifies that elements are sorted in reverse order (i.e. descending order).

**Note**: The Collections.sort() method is the more convenient method for sorting an arraylist

# Java ArrayList clone()

The Java ArrayList clone() method makes the shallow copy of an array list.

Here, the shallow copy means it creates copy of arraylist object. To learn more on shallow copy, visit [Java Shallow Copy](https://stackoverflow.com/questions/1175620/in-java-what-is-a-shallow-copy).

The syntax of the clone() method is:

arraylist.clone()

Here, arraylist is an object of the ArrayList class.

## clone() Parameters

The clone() method does not have any parameters.

## clone() Return Value

* returns a copy of the ArrayList object

## Example 1: Make a Copy of ArrayList

import java.util.ArrayList;

class Main {

public static void main(String[] args){

// create an arraylist

ArrayList<Integer> number = new ArrayList<>();

number.add(1);

number.add(3);

number.add(5);

System.out.println("ArrayList: " + number);

// create copy of number

ArrayList<Integer> cloneNumber = (ArrayList<Integer>)number.clone();

System.out.println("Cloned ArrayList: " + cloneNumber);

}

}

**Output**

ArrayList: [1, 3, 5]

Cloned ArrayList: [1, 3, 5]

In the above example, we have created an arraylist named number. Notice the expression,

(ArrayList<Integer>)number.clone()

Here,**number.clone()** - returns a copy of the object number

* **(ArrayList<Integer>)** - converts value returned by clone() into an arraylist of Integer type (To learn more, visit [Java Typecasting](https://www.programiz.com/java-programming/typecasting))

## Example 2: Print the Return Value of clone()

import java.util.ArrayList;

class Main {

public static void main(String[] args){

// create an arraylist

ArrayList<Integer> prime = new ArrayList<>();

prime.add(2);

prime.add(3);

prime.add(5);

System.out.println("Prime Number: " + prime);

// print the return value of clone()

System.out.println("Return value of clone(): " + prime.clone());

}

}

**Output**

Prime Number: [2, 3, 5]

Return value of clone(): [2, 3, 5]

In the above example, we have created an arraylist named prime. Here, we have printed the value returned by clone()

# Java ArrayList contains()

In this tutorial, we will learn about the Java ArrayList contains() method with the help of examples.

The contains() method checks if the specified element is present in the arraylist.

## Syntax of ArrayList contains()

The syntax of the contains() method is:

arraylist.contains(Object obj)

Here, arraylist is an object of the ArrayList class.

contains() Parameter

The contains() method takes a single parameter.

* **obj** - element that is checked if present in the arraylist

contains() Return Value

returns **true** if the specified element is present in the arraylist.

* returns **false** if the specified element is not present in the arraylist.

### Example

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<String> languages = new ArrayList<>();

languages.add("Java");

languages.add("Python");

languages.add("JavaScript");

System.out.println("ArrayList: " + languages);

// checks if 3 is present in the arraylist

System.out.print("Is Java present in the arraylist: ");

System.out.println(languages.contains("Java"));

}

}

// Output: ArrayList: [Java, Python, JavaScript]

// Is Java present in the arraylist: true

## Example 1: contains() Method with Integer ArrayList

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<Integer> numbers = new ArrayList<>();

// insert element to the arraylist

numbers.add(2);

numbers.add(3);

numbers.add(5);

System.out.println("Number ArrayList: " + numbers);

// checks if 3 is present in the arraylist

System.out.print("Is 3 present in the arraylist: ");

System.out.println(numbers.contains(3));

// checks if 1 is present in the arraylist

System.out.print("Is 1 present in the arraylist: ");

System.out.println(numbers.contains(1));

}

}

**Output**

Number ArrayList: [2, 3, 5]

Is 3 present in the arraylist: true

Is 1 present in the arraylist: false

n the above example, we have created an Integer arraylist named number. Notice the expressions,

// returns true

number.contains(3)

// returns false

number.contains(1)

Here, the contains() method checks if **3** is present in the list. Since **3** is present, the method returns true. However, **1** is not present in the list so the method returns false.

## Example 2: contains() Method with String ArrayList

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<String> languages = new ArrayList<>();

// insert element to the arraylist

languages.add("Java");

languages.add("JavaScript");

languages.add("Python");

System.out.println("Programming Languages: " + languages);

// checks if Java is present in languages

System.out.print("Is Java present in the arraylist: ");

System.out.println(languages.contains("Java"));

// checks if C++ is present in languages

System.out.print("Is C++ present in the arraylist: ");

System.out.println(languages.contains("C++"));

}

}

**Output**

Programming Languages: [Java, JavaScript, Python]

Is Java present in the arraylist: true

Is C++ present in the arraylist: false

In the above example, we have used the contains() method to check if elements Java and C++ are present in the arraylist languages.

Since Java is present in the arraylist, the method returns true. However, C++ is not present in the list. Hence, the method returns false.

**Note**: The contains() method internally uses the equals() method to find the element. Hence, if the specified element matches with the element in arraylist, the method returns true.

# Java ArrayList ensureCapacity()

The Java ArrayList ensureCapacity() method sets the size of an arraylist with the specified capacity.

The syntax of the ensureCapacity() method is:

arraylist.ensureCapacity(int minCapacity)

Here, arraylist is an object of the ArrayList class.

## ensureCapacity() Parameters

The ensureCapacity() takes a single parameter.

* **minCapacity** - the specified minimum capacity of the arraylist

## ensureCapacity() Return Values

The ensureCapacity() method does not return any value.

## Example 1: Java ArrayList ensureCapacity()

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

ArrayList<String> languages= new ArrayList<>();

// set the capacity of the arraylist

languages.ensureCapacity(3);

// Add elements in the ArrayList

languages.add("Java");

languages.add("Python");

languages.add("C");

System.out.println("ArrayList: " + languages);

}

}

**Output**

ArrayList: [Java, Python, C]

In the above example, we have created an arraylist named languages. Notice the line,

languages.ensureCapacity(3);

Here, the ensureCapacity() method resized the arraylist to store 3 elements.

However, ArrayList in Java is dynamically resizable. That is, if we add more than 3 elements in the arraylist, it will automatically resize itself. For example,

## Example 2: Working of ensureCapacity()

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

ArrayList<String> languages= new ArrayList<>();

// set the capacity of the arraylist

languages.ensureCapacity(3);

// Add elements in the ArrayList

languages.add("Java");

languages.add("Python");

languages.add("C");

// add 4th element

languages.add("Swift");

System.out.println("ArrayList: " + languages);

}

}

**Output**

ArrayList: [Java, Python, C, Swift]

In the above example, we use the ensureCapacity() method to resize the arraylist to store 3 elements. However, when we add 4th element in the arraylist, the arraylist automatically resizes.

So, **why do we need to resize arraylist using the ensureCapacity() method if the arraylist can automatically resize itself?**

It is because if we use the ensureCapacity() to resize the arraylist, then the arraylist will be resized at once with the specified capacity. Otherwise, the arraylist will be resized every time when an element is added.

# Java ArrayList isEmpty()

The Java ArrayList isEmpty() method checks if the arraylist is empty.

The syntax of the isEmpty() method is:

arraylist.isEmpty()

Here, arraylist is an object of the ArrayList class.

## isEmpty() Parameters

The isEmpty() method does not take any parameters.

## isEmpty() Return Value

* **returns true** if the arraylist does not contain any elements
* **returns false** if the arraylist contains some elements

## Example: Check if ArrayList is Empty

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<String> languages = new ArrayList<>();

System.out.println("Newly Created ArrayList: " + languages);

// checks if the ArrayList has any element

boolean result = languages.isEmpty(); // true

System.out.println("Is the ArrayList empty? " + result);

// add some elements to the ArrayList

languages.add("Python");

languages.add("Java");

System.out.println("Updated ArrayList: " + languages);

// checks if the ArrayList is empty

result = languages.isEmpty(); // false

System.out.println("Is the ArrayList empty? " + result);

}

}

**Output**

Newly Created ArrayList: []

Is the ArrayList empty? true

Updated ArrayList: [Python, Java]

Is the ArrayList empty? false

In the above example, we have created a arraylist named languages. Here, we have used the isEmpty() method to check whether the arraylist contains any elements or not.

Initially, the newly created arraylist does not contain any element. Hence, isEmpty() returns true. However, after adding some elements (**Python**, **Java**), the method returns false.

# Java ArrayList trimToSize()

The Java ArrayList trimToSize() method trims (sets) the capacity of the arraylist equal to the number of elements in the arraylist.

The syntax of the trimToSize() method is:

arraylist.trimToSize();

Here, arraylist is an object of the ArrayList class.

## trimToSize() Parameters

The trimToSize() method does not take any parameters

## trimToSize() Return Value

The trimToSize() method does not return any value. Rather, it only changes the capacity of the arraylist.

## Example 1: Java ArrayList trimToSize()

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<String> languages = new ArrayList<>();

// add element to ArrayList

languages.add("Java");

languages.add("Python");

languages.add("JavaScript");

System.out.println("ArrayList: " + languages);

// trim capacity to 3

languages.trimToSize();

System.out.println("Size of ArrayList: " + languages.size());

}

}

**Output**

ArrayList: [Java, Python, JavaScript]

Size of ArrayList: 3

In the above example, we have created two arraylists named languages. The arraylist contains 3 elements. Notice the line,

languages.trimToSize();

Here, the trimToSize() method sets the capacity of arraylist equal to the number of elements in languages (i.e. 3).

We have used the [ArrayList size()](https://www.programiz.com/java-programming/library/arraylist/size) method to get the number of elements in the arraylist.

## Advantage of ArrayList trimToSize()

We know that the capacity of ArrayList is dynamically changed. So **what is the advantage of using ArrayList trimToSize() method?**

To understand the advantage of trimToSize() method, we need to look into the working of ArrayList.

Internally, ArrayList uses an array to store all its elements. Now, at some point, the array will be filled. When the internal array is full, a new array is created with 1.5 times more capacity than the current array. And, all elements are moved to the new array.

For example, suppose the internal array is full and we have to add only **1** element. Here, the ArrayList will expand with the same ratio (i.e. 1.5 times the previous array).

In this case, there will be some unassigned space in the internal array. Hence, the trimToSize() method removes the unassigned space and changes the capacity of arraylist equal to the number of elements in the arraylist.

This working of ArrayList trimToSize() method is not visible to the user.

# Java ArrayList subList()

The Java ArrayList subList() method extracts a portion of the arraylist and returns it.

The syntax of the subList() method is:

arraylist.subList(int fromIndex, int toIndex)

Here, arraylist is an object of the ArrayList class.

## subList() Parameters

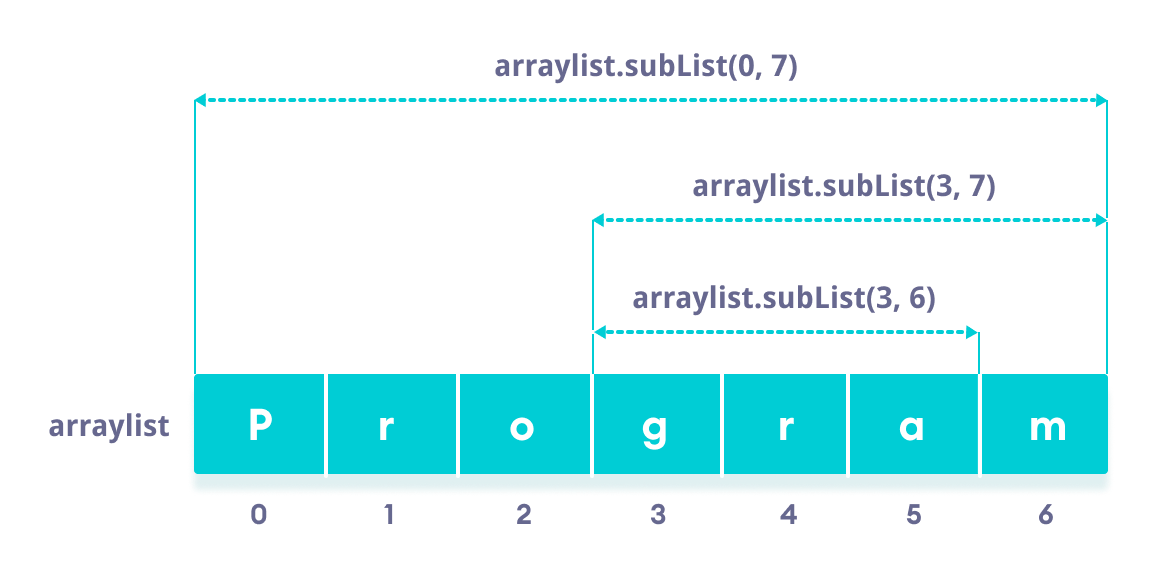
The subList() method takes two parameters.

* **fromIndex** - the starting position from where elements are extracted
* **toIndex** - the ending position up to which elements are extracted

## subList() Return Value

* returns a portion of arraylist from the given arraylist
* throws IndexOutOfBoundsException, if fromIndex is less than 0 or toIndex is greater than the size of arraylist
* throws IllegalArgumentException, if fromIndex is greater than toIndex.

**Note**: The portion of arraylist contains elements starting at fromIndex and extends up to element at toIndex-1. That is, the element at toIndex is not included.

Working of ArrayList subList()

## Example 1: Get a Sub List From an ArrayList

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<String> languages = new ArrayList<>();

// add some elements to the ArrayList

languages.add("JavaScript");

languages.add("Java");

languages.add("Python");

languages.add("C");

System.out.println("ArrayList: " + languages);

// element from 1 to 3

System.out.println("SubList: " + languages.subList(1, 3));

}

}

**Output**

ArrayList: [JavaScript, Java, Python, C]

SubList: [Java, Python]

In the above example, we have used the subList() method to get elements from index 1 to 3 (excluding 3).

## Example 2: Split a Single ArrayList into Two ArrayLists

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<Integer> ages = new ArrayList<>();

// add some elements to the ArrayList

ages.add(10);

ages.add(12);

ages.add(15);

ages.add(19);

ages.add(23);

ages.add(34);

System.out.println("List of Age: " + ages);

// ages below 18

System.out.println("Ages below 18: " + ages.subList(0, 3));

// ages above 18

System.out.println("Ages above 18: " + ages.subList(3, ages.size()));

}

}

**Output**

List of Age: [10, 12, 15, 19, 23, 34]

Ages below 18: [10, 12, 15]

Ages above 18: [19, 23, 34]

In the above example, we have created an arraylist named ages. Here, we have used the subList() method to split the arraylist into two arraylists: **Ages below 18** and **Ages above 18**.

Note that we have used the ages.size() method to get the length of the arraylist.

# Java ArrayList lastIndexOf()

The Java ArrayList lastIndexOf() method returns the position of the last occurrence of the specified element.

**int lastIndexOf(Object o):** It returns the index of the last occurrence of a specified element in the list. If the list does not contain that particular element, it will return -1.

The syntax of the lastIndexOf() method is:

arraylist.lastIndexOf(Object obj)

Here, arraylist is an object of the ArrayList class.

## lastIndexOf() Parameter

The lastIndexOf() method takes a single parameter.

* **obj** - element whose position is to be returned

If the same element obj is present in multiple locations, then the position of the element that appears last is returned.

## lastIndexOf() Return Value

* returns the position of the last occurrence of the specified element from the arraylist

**Note**: If the specified element doesn't exist in the list, the lastIndexOf() method returns **-1**.

## Example: Get the Last Occurrence of ArrayList Element

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// create an ArrayList

ArrayList<String> languages = new ArrayList<>();

// insert element to the ArrayList

languages.add("JavaScript");

languages.add("Python");

languages.add("Java");

languages.add("C++");

languages.add("Java");

System.out.println("Programming Languages: " + languages);

// get the position of Java occurred last

int position1 = languages.lastIndexOf("Java");

System.out.println("Last Occurrence of Java: " + position1);

// C is not in the ArrayList

// Returns -1

int position2 = languages.lastIndexOf("C");

System.out.println("Last Occurrence of C: " + position2);

}

}

**Output**

Programming Languages: [JavaScript, Python, Java, C++, Java]

Last Occurrence of Java: 4

Last Occurrence of C: -1

In the above example, we have created an arraylist named languages. Notice the expressions,

// returns 4

languages.lastIndexOf("Java")

// returns -1

languages.lastIndexOf("C")

Here, the lastIndexOf() method successfully returns the position of the last occurrence of **Java** (i.e. 4). However, element **C** doesn't exist in the arraylist. Hence, the method returns **-1**.

And, if we want to get the first occurrence of Java, we can use the indexOf() method.