

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

# List Interface in Java with Examples

Difficulty Level : Easy • Last Updated : 10 Jan, 2023

The List interface in Java provides a way to store the ordered collection. It is a child interface of [Collection](#). It is an ordered collection of objects in which duplicate values can be stored. Since List preserves the insertion order, it allows positional access and insertion of elements.

The List interface is found in `java.util` package and inherits the Collection interface. It is a factory of ListIterator interface. Through the ListIterator, we can iterate the list in forward and backward directions. The implementation classes of the List interface are ArrayList, LinkedList, Stack, and Vector. ArrayList and LinkedList are widely used in Java programming. The Vector class is deprecated since Java 5.



**Declaration:** The List interface is declared as:

```
public interface List<E> extends Collection<E> ;
```

**Let us elaborate on creating objects or instances in a List class.** Since **List** is an [interface](#), objects cannot be created of the type list. We always need a class that

Read    Discuss    Courses    Practice    Video

defined 'classes', **List** is an 'interface', implemented by the **ArrayList** class, pre-defined in the **java.util** package.

**Syntax:** This type of safelist can be defined as:

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

**Note:** Obj is the type of the object to be stored in List

**Example:**

## Java

```
// Java program to Demonstrate List Interface

// Importing all utility classes
import java.util.*;

// Main class
// ListDemo class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Creating an object of List interface
        // implemented by the ArrayList class
        List<Integer> l1 = new ArrayList<Integer>();

        // Adding elements to object of List interface
        // Custom inputs

        l1.add(0, 1);
        l1.add(1, 2);

        // Print the elements inside the object
        System.out.println(l1);

        // Now creating another object of the List
    }
}
```

# Start Your Coding Journey Now!

Read   Discuss   Courses   Practice   Video

```
// Again adding elements to object of List interface
// Custom inputs
l2.add(1);
l2.add(2);
l2.add(3);

// Will add list l2 from 1 index
l1.addAll(1, l2);

System.out.println(l1);

// Removes element from index 1
l1.remove(1);

// Printing the updated List 1
System.out.println(l1);

// Prints element at index 3 in list 1
// using get() method
System.out.println(l1.get(3));

// Replace 0th element with 5
// in List 1
l1.set(0, 5);

// Again printing the updated List 1
System.out.println(l1);
}
```

## Output

```
[1, 2]
[1, 1, 2, 3, 2]
[1, 2, 3, 2]
2
[5, 2, 3, 2]
```

Now let us perform various operations using List Interface to have a better understanding of the same. We will be discussing the following operations listed below and later on implementing via clean java codes.

# Start Your Coding Journey Now!

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

- **Operation 1:** Adding elements to List class using add() method
- **Operation 2:** Updating elements in List class using set() method
- **Operation 3:** Removing elements using remove() method

Now let us discuss the operations individually and implement the same in the code to grasp a better grip over it.

## Operation 1: Adding elements to List class using **add()** method

In order to add an element to the list, we can use the [add\(\) method](#). This method is overloaded to perform multiple operations based on different parameters.

**Parameters:** It takes 2 parameters, namely:

- **add(Object):** This method is used to add an element at the end of the List.
- **add(int index, Object):** This method is used to add an element at a specific index in the List

## Example:

### Java

```
// Java Program to Add Elements to a List

// Importing all utility classes
import java.util.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {
        // Creating an object of List interface,
        // implemented by ArrayList class

```

# Start Your Coding Journey Now!

Read    Discuss    Courses    Practice    Video

```
// List interface object  
System.out.println(al);  
}  
}
```

## Output

[Geeks, For, Geeks]

### Operation 2: Updating elements

After adding the elements, if we wish to change the element, it can be done using the **set()** method. Since List is indexed, the element which we wish to change is referenced by the index of the element. Therefore, this method takes an index and the updated element which needs to be inserted at that index.

#### Example:

## Java

```
// Java Program to Update Elements in a List  
  
// Importing utility classes  
import java.util.*;  
  
// Main class  
class GFG {  
  
    // Main driver method  
    public static void main(String args[])  
    {  
        // Creating an object of List interface  
        List<String> al = new ArrayList<>();  
  
        // Adding elements to object of List class  
        al.add("Geeks");  
        al.add("Geeks");  
        al.add(1, "Geeks");  
  
        // Display the initial elements in List  
        System.out.println("Initial ArrayList " + al);  
  
        // Setting (updating) element at 1st index
```

# Start Your Coding Journey Now!

Read   Discuss   Courses   Practice   Video

```
        System.out.println("Updated ArrayList " + al);
    }
}
```

## Output

```
Initial ArrayList [Geeks, Geeks, Geeks]
Updated ArrayList [Geeks, For, Geeks]
```

### Operation 3: Removing Elements

In order to remove an element from a list, we can use the [remove\(\) method](#). This method is overloaded to perform multiple operations based on different parameters. They are:

#### Parameters:

- **remove(Object):** This method is used to simply remove an object from the List. If there are multiple such objects, then the first occurrence of the object is removed.
- **remove(int index):** Since a List is indexed, this method takes an integer value which simply removes the element present at that specific index in the List. After removing the element, all the elements are moved to the left to fill the space and the indices of the objects are updated.

#### Example:

## Java

```
// Java Program to Remove Elements from a List

// Importing List and ArrayList classes
// from java.util package
import java.util.ArrayList;
import java.util.List;

// Main class
class GFG {

    // Main driver method
```

# Start Your Coding Journey Now!

Read    Discuss    Courses    Practice    Video

```
List<String> al = new ArrayList<>();  
  
        // Adding elements to the object  
        // Custom inputs  
        al.add("Geeks");  
        al.add("Geeks");  
  
        // Adding For at 1st indexes  
        al.add(1, "For");  
  
        // Print the initialArrayList  
        System.out.println("Initial ArrayList " + al);  
  
        // Now remove element from the above list  
        // present at 1st index  
        al.remove(1);  
  
        // Print the List after removal of element  
        System.out.println("After the Index Removal " + al);  
  
        // Now remove the current object from the updated  
        // List  
        al.remove("Geeks");  
  
        // Finally print the updated List now  
        System.out.println("After the Object Removal "  
                           + al);  
    }  
}
```

## Output

Initial ArrayList [Geeks, For, Geeks]

After the Index Removal [Geeks, Geeks]

After the Object Removal [Geeks]

Till now we are having a very small input size and we are doing operations manually for every entity. Now let us discuss various ways by which we can iterate over the list to get them working for a larger sample set.

**Methods:** There are multiple ways to iterate through the List. The most famous ways are by using the basic [for loop](#) in combination with a [get\(\) method](#) to get the

# Start Your Coding Journey Now!

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

## Java

```
// Java program to Iterate the Elements
// in an ArrayList

// Importing java utility classes
import java.util.*;

// Main class
public class GFG {

    // main driver method
    public static void main(String args[])
    {
        // Creating an empty ArrayList of string type
        List<String> al = new ArrayList<>();

        // Adding elements to above object of ArrayList
        al.add("Geeks");
        al.add("Geeks");

        // Adding element at specified position
        // inside list object
        al.add(1, "For");

        // Using for loop for iteration
        for (int i = 0; i < al.size(); i++) {

            // Using get() method to
            // access particular element
            System.out.print(al.get(i) + " ");
        }

        // New line for better readability
        System.out.println();

        // Using for-each loop for iteration
        for (String str : al)

            // Printing all the elements
            // which was inside object
            System.out.print(str + " ");
    }
}
```

# Start Your Coding Journey Now!

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

Since the main concept behind the different types of the lists is the same, the list interface contains the following methods:

Method	Description
<a href="#"><u>add(int index, element)</u></a>	This method is used to add an element at a particular index in the list. When a single parameter is passed, it simply adds the element at the end of the list.
<a href="#"><u>addAll(int index, Collection collection)</u></a>	This method is used to add all the elements in the given collection to the list. When a single parameter is passed, it adds all the elements of the given collection at the end of the list.
<a href="#"><u>size()</u></a>	This method is used to return the size of the list.
<a href="#"><u>clear()</u></a>	This method is used to remove all the elements in the list. However, the reference of the list created is still stored.
<a href="#"><u>remove(int index)</u></a>	This method removes an element from the specified index. It shifts subsequent elements(if any) to left and decreases their indexes by 1.
<a href="#"><u>remove(element)</u></a>	This method is used to remove the first occurrence of the given element in the list.
<a href="#"><u>get(int index)</u></a>	This method returns elements at the specified index.
<a href="#"><u>set(int index, element)</u></a>	This method replaces elements at a given index with the new element. This function returns the element which was just replaced by a new element.

# Start Your Coding Journey Now!

Read   Discuss   Courses   Practice   Video

<u><a href="#">indexOf(element)</a></u>	This method returns the first occurrence of the given element or -1 if the element is not present in the list.
<u><a href="#">lastIndexOf(element)</a></u>	This method returns the last occurrence of the given element or -1 if the element is not present in the list.
<u><a href="#">equals(element)</a></u>	This method is used to compare the equality of the given element with the elements of the list.
<u><a href="#">hashCode()</a></u>	This method is used to return the hashcode value of the given list.
<u><a href="#">isEmpty()</a></u>	This method is used to check if the list is empty or not. It returns true if the list is empty, else false.
<u><a href="#">contains(element)</a></u>	This method is used to check if the list contains the given element or not. It returns true if the list contains the element.
<u><a href="#">containsAll(Collection collection)</a></u>	This method is used to check if the list contains all the collection of elements.
<u><a href="#">sort(Comparator comp)</a></u>	This method is used to sort the elements of the list on the basis of the given <u><a href="#">comparator</a></u> .

Both the List interface and the Set interface inherits the Collection interface. However, there exists some differences between them.

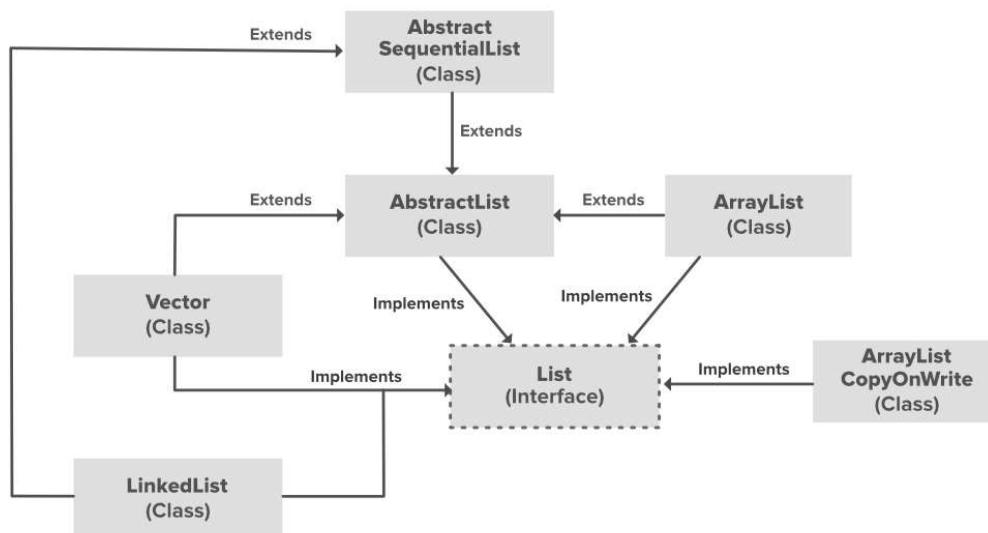
List	Set
The List is an ordered sequence.	The Set is an unordered sequence.
List allows duplicate elements	Set doesn't allow duplicate elements.

# Start Your Coding Journey Now!

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

	allowed.
Multiple null elements can be stored.	The null element can store only once.
List implementations are ArrayList, LinkedList, Vector, Stack	Set implementations are HashSet, LinkedHashSet.

Now let us discuss the classes that implement the List Interface for which first do refer to the pictorial representation below to have a better understanding of the List interface. It is as follows:



[AbstractList](#), [CopyOnWriteArrayList](#), and the [AbstractSequentialList](#) are the classes that implement the List interface. A separate functionality is implemented in each of the mentioned classes. They are as follows:

- 1. AbstractList:** This class is used to implement an unmodifiable list, for which one needs to only extend this AbstractList Class and implement only the `get()` and the `size()` methods.
- 2. CopyOnWriteArrayList:** This class implements the list interface. It is an enhanced version of [ArrayList](#) in which all the modifications (add, set, remove, etc.) are implemented by making a fresh copy of the list.
- 3. AbstractSequentialList:** This class implements the [Collection interface](#) and the AbstractCollection class. This class is used to implement an unmodifiable list,

# Start Your Coding Journey Now!

Read    Discuss    Courses    Practice    Video

We will proceed in this manner.

- ArrayList
- Vector
- Stack
- LinkedList

Let us discuss them sequentially and implement the same to figure out the working of the classes with the List interface.

An **ArrayList** class which is implemented in the collection framework provides us with dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed. Let's see how to create a list object using this class.

**Example:**

## Java

```
// Java program to demonstrate the
// creation of list object using the
// ArrayList class

import java.io.*;
import java.util.*;

class GFG {
    public static void main(String[] args)
    {
        // Size of ArrayList
        int n = 5;

        // Declaring the List with initial size n
        List<Integer> arrli
            = new ArrayList<Integer>(n);

        // Appending the new elements
        // at the end of the list
        for (int i = 1; i <= n; i++)
            arrli.add(i);
```

# Start Your Coding Journey Now!

Read    Discuss    Courses    Practice    Video

```
arrli.remove(3);

// Displaying the list after deletion
System.out.println(arrli);

// Printing elements one by one
for (int i = 0; i < arrli.size(); i++)
    System.out.print(arrli.get(i) + " ");
}

}
```

## Output

```
[1, 2, 3, 4, 5]
[1, 2, 3, 5]
1 2 3 5
```

Vector is a class that is implemented in the collection framework implements a growable array of objects. Vector implements a dynamic array that means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index. Vectors basically fall in legacy classes but now it is fully compatible with collections. Let's see how to create a list object using this class.

## Example:

## Java

```
// Java program to demonstrate the
// creation of list object using the
// Vector class

import java.io.*;
import java.util.*;

class GFG {
    public static void main(String[] args)
    {
        // Size of the vector
        int n = 5;
```

# Start Your Coding Journey Now!

Read    Discuss    Courses    Practice    Video

```
// at the end of the list
for (int i = 1; i <= n; i++)
    v.add(i);

// Printing elements
System.out.println(v);

// Remove element at index 3
v.remove(3);

// Displaying the list after deletion
System.out.println(v);

// Printing elements one by one
for (int i = 0; i < v.size(); i++)
    System.out.print(v.get(i) + " ");
}

}
```

## Output

```
[1, 2, 3, 4, 5]
[1, 2, 3, 5]
1 2 3 5
```

Stack is a class that is implemented in the collection framework and extends the vector class models and implements the [Stack data structure](#). The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search and peek. Let's see how to create a list object using this class.

## Example:

## Java

```
// Java program to demonstrate the
// creation of list object using the
// Stack class

import java.io.*;
import java.util.*;
```

# Start Your Coding Journey Now!

Read   Discuss   Courses   Practice   Video

```
int n = 5;

// Declaring the List
List<Integer> s = new Stack<Integer>();

// Appending the new elements
// at the end of the list
for (int i = 1; i <= n; i++)
    s.add(i);

// Printing elements
System.out.println(s);

// Remove element at index 3
s.remove(3);

// Displaying the list after deletion
System.out.println(s);

// Printing elements one by one
for (int i = 0; i < s.size(); i++)
    System.out.print(s.get(i) + " ");
}

}
```

## Output

```
[1, 2, 3, 4, 5]
[1, 2, 3, 5]
1 2 3 5
```

LinkedList is a class that is implemented in the collection framework which inherently implements the [linked list data structure](#). It is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node. Due to the dynamicity and ease of insertions and deletions, they are preferred over the arrays. Let's see how to create a list object using this class.

## Example:

# Start Your Coding Journey Now!

Read    Discuss    Courses    Practice    Video

```
// creation of list object using the
// LinkedList class

import java.io.*;
import java.util.*;

class GFG {
    public static void main(String[] args)
    {
        // Size of the LinkedList
        int n = 5;

        // Declaring the List with initial size n
        List<Integer> ll = new LinkedList<Integer>();

        // Appending the new elements
        // at the end of the list
        for (int i = 1; i <= n; i++)
            ll.add(i);

        // Printing elements
        System.out.println(ll);

        // Remove element at index 3
        ll.remove(3);

        // Displaying the list after deletion
        System.out.println(ll);

        // Printing elements one by one
        for (int i = 0; i < ll.size(); i++)
            System.out.print(ll.get(i) + " ");
    }
}
```

## Output

[1, 2, 3, 4, 5]

[1, 2, 3, 5]

1 2 3 5

# Start Your Coding Journey Now!

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)[Previous](#)[Next](#)[Collections Class in Java](#)[ArrayList in Java](#)

## Related Articles

1. [Why to Use Comparator Interface Rather than Comparable Interface in Java?](#)

---
2. [Java 8 | Consumer Interface in Java with Examples](#)

---
3. [Java 8 | BiConsumer Interface in Java with Examples](#)

---
4. [Java 8 | IntToDoubleFunction Interface in Java with Examples](#)

---
5. [Java 8 | DoubleToLongFunction Interface in Java with Examples](#)

---
6. [Java 8 | IntToLongFunction Interface in Java with Examples](#)

---
7. [Java.util.function.BiPredicate interface in Java with Examples](#)

---
8. [Java.util.function.DoublePredicate interface in Java with Examples](#)

---
9. [Java.util.function.LongPredicate interface in Java with Examples](#)

---
10. [Java.util.function.IntPredicate interface in Java with Examples](#)

# Start Your Coding Journey Now!

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

## Vote for difficulty

Current difficulty : [Easy](#)

[Easy](#)[Normal](#)[Medium](#)[Hard](#)[Expert](#)

**Improved By :** [Chinmoy Lenka](#), [KaashyapMSK](#), [aswinpm14](#), [solankimayank](#), [simmytarika5](#), [surindertarika1234](#), [surinderdawra388](#), [as360571](#), [casesensitiveunofficial](#), [kashishsoda](#), [gulshankumarar231](#), [akshaysingh98088](#), [nishkarshgandhi](#)

**Article Tags :** [Java - util package](#), [Java-Collections](#), [java-list](#), [Java](#)

**Practice Tags :** [Java](#), [Java-Collections](#)

[Improve Article](#)[Report Issue](#)

GeeksforGeeks

A-143, 9th Floor, Sovereign Corporate Tower,  
Sector-136, Noida, Uttar Pradesh - 201305

[feedback@geeksforgeeks.org](mailto:feedback@geeksforgeeks.org)

## Company

[About Us](#)[Careers](#)[In Media](#)[Contact Us](#)[Privacy Policy](#)

## Learn

[DSA](#)[Algorithms](#)[Data Structures](#)[SDE Cheat Sheet](#)[Machine learning](#)

# Start Your Coding Journey Now!

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

## News

[Top News](#)[Technology](#)[Work & Career](#)[Business](#)[Finance](#)[Lifestyle](#)[Knowledge](#)

## Languages

[Python](#)[Java](#)[CPP](#)[Golang](#)[C#](#)[SQL](#)[Kotlin](#)

## Web Development

[Web Tutorials](#)[Django Tutorial](#)[HTML](#)[JavaScript](#)[Bootstrap](#)[ReactJS](#)[NodeJS](#)

## Contribute

[Write an Article](#)[Improve an Article](#)[Pick Topics to Write](#)[Write Interview Experience](#)[Internships](#)[Video Internship](#)

@geeksforgeeks , Some rights reserved