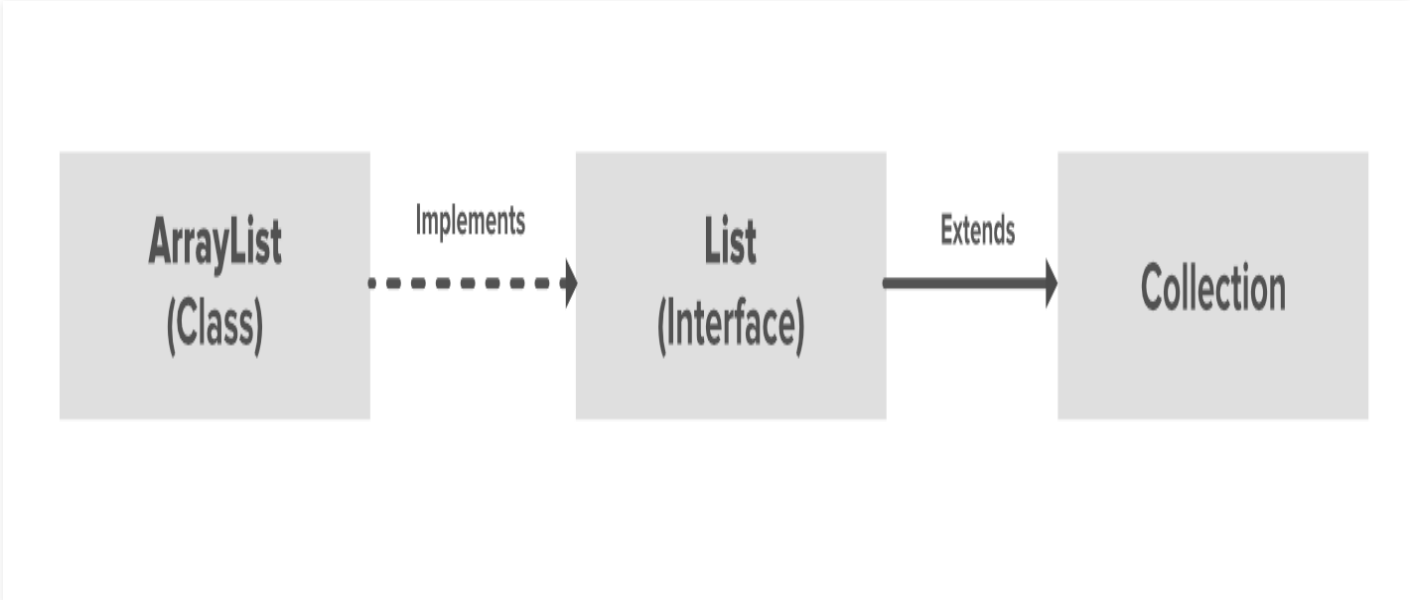
List Interface in Java with Examples

Difficulty Level : Easy Last Updated : 06 Jan, 2022

The List interface 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.



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 implements this **List** in order to create an object. And also, after the introduction of Generics in Java 1.5, it is possible to restrict the type of object that can be stored in the List. Just like several other user-defined '*interfaces' implemented by* user-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 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
        // interface implemented ArrayList class
        // Declaring object of integer type
        List<Integer> l2 = new ArrayList<Integer>();

        // 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.

Operations in a List interface

Since List is an interface, it can be used only with a class that implements this interface. Now, let's see how to perform a few frequently used operations on the List.

- **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 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
        List<String> al = new ArrayList<>();

        // Adding elements to object of List interface
        // Custom elements
        al.add("Geeks");
        al.add("Geeks");
        al.add(1, "For");

        // Print all the elements inside the
        // 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 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 theinitial elements in List
        System.out.println("Initial ArrayList " + al);

        // Setting (updating) element at 1st index
        // using set() method
        al.set(1, "For");

        // Print and display the updated List
        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 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
    public static void main(String args[])
    {

        // Creating List class object
        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]
```

Iterating over List

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 the working for a larger sampler 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 element at a specific index and the advanced for loop.

Example:

```
// 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 + " ");
    }
}
```

Output:

```
Geeks For Geeks
Geeks For Geeks
```

Methods of the List Interface

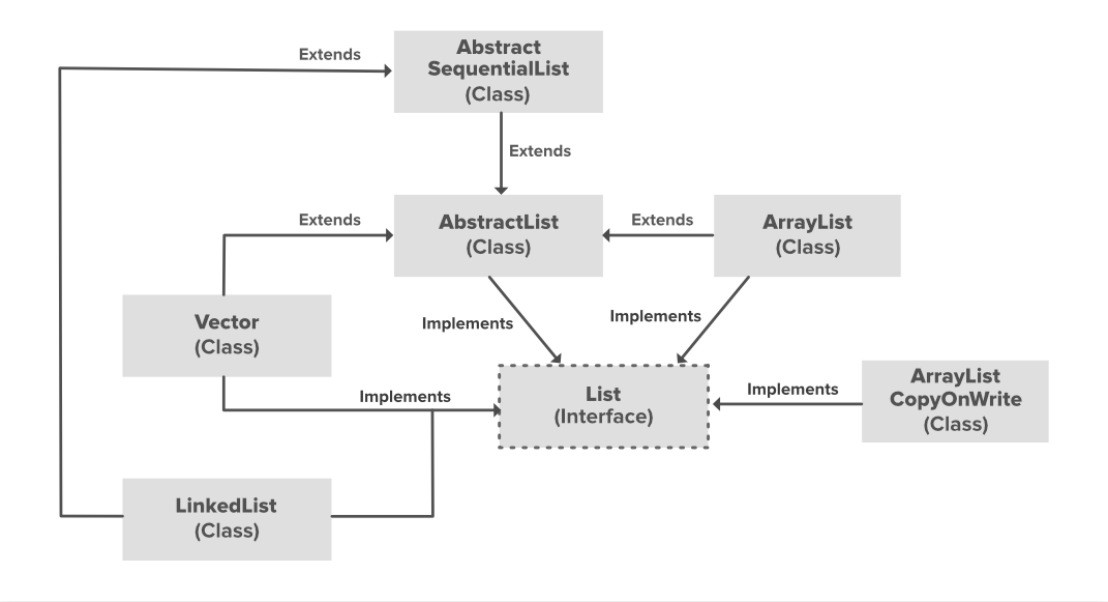
Since the main concept behind the different types of the lists is the same, the list interface contains the following methods:

Method	Description
add(int index, element)	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.
addAll(int index, Collection collection)	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.
size()	This method is used to return the size of the list.
clear()	This method is used to remove all the elements in the list. However, the reference of the list created is still stored.
remove(int index)	This method removes an element from the specified index. It shifts subsequent elements(if any) to left and decreases their indexes by 1.
remove(element)	This method is used to remove the first occurrence of the given element in the list.
get(int index)	This method returns elements at the specified index.
set(int index, element)	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.
indexOf(element)	This method returns the first occurrence of the given element or *-1* if the element is not present in the list.
lastIndexOf(element)	This method returns the last occurrence of the given element or *-1* if the element is not present in the list.
equals(element)	This method is used to compare the equality of the given element with the elements of the list.

Method	Description
hashCode()	This method is used to return the hashcode value of the given list.
isEmpty()	This method is used to check if the list is empty or not. It returns true if the list is empty, else false.
contains(element)	This method is used to check if the list contains the given element or not. It returns true if the list contains the element.
containsAll(Collection collection)	This method is used to check if the list contains all the collection of elements.
sort(Comparator comp)	This method is used to sort the elements of the list on the basis of the given comparator.

Classes Association with a List Interface

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, for which one needs to only extend this AbstractList Class and implement only the *get()* and the *size()* methods.

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.

Class 1: ArrayList

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 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);

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

        // Remove element at index 3
        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
```

Class 2: Vector

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;

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

        // Appending the new elements
        // 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
```

Class 3: Stack

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

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

class GFG {
    public static void main(String[] args)
    {
        // Size of the stack
        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
```

Class 4: LinkedList

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:

```
// Java program to demonstrate the
// 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
```