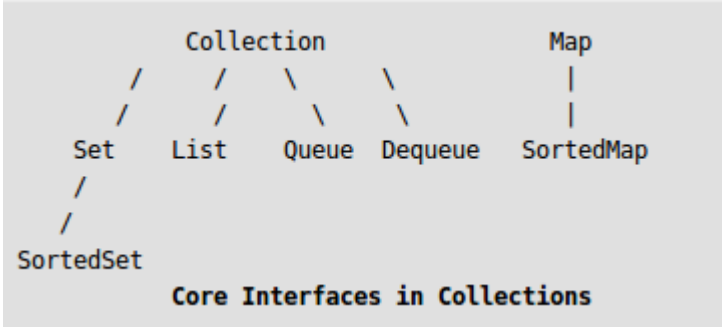
List Interface in Java with Examples

Java.util.List is a child interface of Collection.



Core Interfaces in Collections

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

Creating List Objects:

List is an interface, we can create instance of List in following ways:

```
List a = new ArrayList();
List b = new LinkedList();
List c = new Vector();
List d = new Stack();
```

Generic List Object:

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. We can declare type safe List in following way:

```
// Obj is type of object to be stored in List.
List<Obj> list = new List<Obj> ();
```

Operations on List:

List Interface extends Collection, hence it supports all the operations of Collection Interface and along

with following operations:

1. **Positional Access:**
   List allows add, remove, get and set operations based on numerical positions of elements in List. List provides following methods for these operations:

   - **void add(int index,Object O):** This method adds given element at specified index.
   - **boolean addAll(int index, Collection c):** This method adds all elements from specified collection to list. First element gets inserted at given index. If there is already an element at that position, that element and other subsequent elements(if any) are shifted to the right by increasing their index.
   - **Object 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.
   - **Object get(int index):** This method returns element at the specified index.
   - **Object set(int index, Object new):** This method replaces element at given index with new element. This function returns the element which was just replaced by new element.

```
// Java program to demonstrate positional access
// operations on List interface
import java.util.*;

public class ListDemo
{
    public static void main (String[] args)
    {
        // Let us create a list
        List l1 = new ArrayList();
        l1.add(0, 1);  // adds 1 at 0 index
        l1.add(1, 2);  // adds 2 at 1 index
        System.out.println(l1);  // [1, 2]

        // Let us create another list
        List l2 = new ArrayList();
        l2.add(1);
        l2.add(2);
        l2.add(3);

        // will add list l2 from 1 index
        l1.addAll(1, l2);
        System.out.println(l1);

        l1.remove(1);     // remove element from index 1
        System.out.println(l1); // [1, 2, 3, 2]

        // prints element at index 3
        System.out.println(l1.get(3));

        l1.set(0, 5);   // replace 0th element with 5
        System.out.println(l1);  // [5, 2, 3, 2]
    }
}
```

Run on IDE

Output :

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

2. **Search:**

List provides methods to search element and returns its numeric position. Following two methods are supported by List for this operation:

- **int indexOf(Object o):** This method returns first occurrence of given element or -1 if element is not present in list.
- **int lastIndexOf(Object o):** This method returns the last occurrence of given element or -1 if element is not present in list.

```
// Java program to demonstrate search
// operations on List interface
import java.util.*;

public class ListDemo
{
    public static void main(String[] args)
    {
        // type safe array list, stores only string
        List<String> l = new ArrayList<String>(5);
        l.add("Geeks");
        l.add("for");
        l.add("Geeks");

        // Using indexOf() and lastIndexOf()
        System.out.println("first index of Geeks:" +
                                l.indexOf("Geeks"));
        System.out.println("last index of Geeks:" +
                               l.lastIndexOf("Geeks"));
        System.out.println("Index of element not present : " +
                                l.indexOf("Hello"));
    }
}
```

Run on IDE

Output :

```
first index of Geeks:0
last index of Geeks:2
Index of element not present : -1
```

3. **Iteration:**
   ListIterator(extends Iterator) is used to iterate over List element. List iterator is bidirectional iterator. For more details about ListIterator refer Iterators in Java.

4. **Range-view:**
   List Interface provides method to get List view of the portion of given List between two indices. Following is the method supported by List for range view operation.

   - **List subList(int fromIndex,int toIndex):**This method returns List view of specified List between fromIndex(inclusive) and toIndex(exclusive).

```
// Java program to demonstrate subList operation
// on List interface.
import java.util.*;
public class ListDemo
{
    public static void main (String[] args)
    {
        // Type safe array list, stores only string
        List<String> l = new ArrayList<String>(5);

        l.add("GeeksforGeeks");
        l.add("Practice");
        l.add("GeeksQuiz");
        l.add("IDE");
        l.add("Courses");

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

        // return List between 2nd(including)
        // and 4th element(excluding)
        range = l.subList(2, 4);

        System.out.println(range);  // [GeeksQuiz, IDE]
    }
}
```

Run on IDE

Output :

```
[GeeksQuiz, IDE]
```

This article is contributed by **Dharmesh Singh**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

GATE CS Corner Company Wise Coding Practice