Java List

List in Java provides the facility to maintain the *ordered collection*. It contains the index-based methods to insert, update, delete and search the elements. It can have the duplicate elements also. We can also store the null elements in the list.

The List interface is found in the 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 List interface are ArrayList, LinkedList, Stack and Vector. The ArrayList and LinkedList are widely used in Java programming. The Vector class is deprecated since Java 5.

List Interface declaration

1. **public interface** List<E> **extends** Collection<E>

Java List Methods

Method	Description
void add(int index, E element)	It is used to insert the specified element at the specified position in a list.
boolean add(E e)	It is used to append the specified element at the end of a list.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of a list.
boolean addAll(int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void clear()	It is used to remove all of the elements from this list.
boolean equals(Object o)	It is used to compare the specified object with the elements of a list.
int hashcode()	It is used to return the hash code value for a list.
E get(int index)	It is used to fetch the element from the particular position of the list.
boolean isEmpty()	It returns true if the list is empty, otherwise false.
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.

Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
<T> T[] toArray(T[] a)	It is used to return an array containing all of the elements in this list in the correct order.
boolean contains(Object o)	It returns true if the list contains the specified element
boolean containsAll(Collection<?> c)	It returns true if the list contains all the specified element
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
E remove(int index)	It is used to remove the element present at the specified position in the list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element.
boolean removeAll(Collection<?> c)	It is used to remove all the elements from the list.
void replaceAll(UnaryOperator<E> operator)	It is used to replace all the elements from the list with the specified element.
void retainAll(Collection<?> c)	It is used to retain all the elements in the list that are present in the specified collection.
E set(int index, E element)	It is used to replace the specified element in the list, present at the specified position.
void sort(Comparator<? super E> c)	It is used to sort the elements of the list on the basis of specified comparator.
Spliterator<E> spliterator()	It is used to create spliterator over the elements in a list.
List<E> subList(int fromIndex, int toIndex)	It is used to fetch all the elements lies within the given range.
int size()	It is used to return the number of elements present in the list.

Java List vs ArrayList

List is an interface whereas ArrayList is the implementation class of List.

How to create List

The ArrayList and LinkedList classes provide the implementation of List interface. Let's see the examples to create the List:

```
1. //Creating a List of type String using ArrayList
2. List<String> list=new ArrayList<String>();
3.
4. //Creating a List of type Integer using ArrayList
5. List<Integer> list=new ArrayList<Integer>();
6.
7. //Creating a List of type Book using ArrayList
8. List<Book> list=new ArrayList<Book>();
9.
10. //Creating a List of type String using LinkedList
11. List<String> list=new LinkedList<String>();
```

In short, you can create the List of any type. The ArrayList<T> and LinkedList<T> classes are used to specify the type. Here, T denotes the type.

Java List Example

Let's see a simple example of List where we are using the ArrayList class as the implementation.

```
1. import java.util.*;
2. public class ListExample1{
3. public static void main(String args[]){
4.  //Creating a List
5.  List<String> list=new ArrayList<String>();
6.  //Adding elements in the List
7.  list.add("Mango");
8.  list.add("Apple");
9.  list.add("Banana");
10. list.add("Grapes");
11. //Iterating the List element using for-each loop
12. for(String fruit:list)
13.  System.out.println(fruit);
14.
15. }
16. }
```

Test it Now

Output:

```
Mango
Apple
Banana
```

```
Grapes
```

How to convert Array to List

We can convert the Array to List by traversing the array and adding the element in list one by one using list.add() method. Let's see a simple example to convert array elements into List.

```
1. import java.util.*;
2. public class ArrayToListExample{
3. public static void main(String args[]){
4. //Creating Array
5. String[] array={"Java","Python","PHP","C++"};
6. System.out.println("Printing Array: "+Arrays.toString(array));
7. //Converting Array to List
8. List<String> list=new ArrayList<String>();
9. for(String lang:array){
10. list.add(lang);
11. }
12. System.out.println("Printing List: "+list);
13.
14. }
15. }
```

Test it Now

Output:

```
Printing Array: [Java, Python, PHP, C++]
Printing List: [Java, Python, PHP, C++]
```

How to convert List to Array

We can convert the List to Array by calling the list.toArray() method. Let's see a simple example to convert list elements into array.

```
1. import java.util.*;
2. public class ListToArrayExample{
3. public static void main(String args[]){
4.  List<String> fruitList = new ArrayList<>();
5.  fruitList.add("Mango");
6.  fruitList.add("Banana");
7.  fruitList.add("Apple");
8.  fruitList.add("Strawberry");
9.  //Converting ArrayList to Array
10. String[] array = fruitList.toArray(new String[fruitList.size()]);
11. System.out.println("Printing Array: "+Arrays.toString(array));
```

```
12.  System.out.println("Printing List: "+fruitList);
13. }
14. }
```

Test it Now

Output:

```
Printing Array: [Mango, Banana, Apple, Strawberry]
Printing List: [Mango, Banana, Apple, Strawberry]
```

Get and Set Element in List

The *get() method* returns the element at the given index, whereas the *set() method* changes or replaces the element.

```
1.  import java.util.*;
2.  public class ListExample2{
3.   public static void main(String args[]){
4.   //Creating a List
5.   List<String> list=new ArrayList<String>();
6.   //Adding elements in the List
7.   list.add("Mango");
8.   list.add("Apple");
9.   list.add("Banana");
10.  list.add("Grapes");
11.  //accessing the element
12.  System.out.println("Returning element: "+list.get(1));//it will return the 2nd ele
     ment, because index starts from 0
13.  //changing the element
14.  list.set(1,"Dates");
15.  //Iterating the List element using for-each loop
16.  for(String fruit:list)
17.   System.out.println(fruit);
18.
19.  }
20. }
```

Test it Now

Output:

```
Returning element: Apple
Mango
Dates
Banana
Grapes
```

How to Sort List

There are various ways to sort the List, here we are going to use Collections.sort() method to sort the list element. The *java.util* package provides a utility class **Collections** which has the static method sort(). Using the **Collections.sort()** method, we can easily sort any List.

```
1. import java.util.*;
2. class SortArrayList{
3.  public static void main(String args[]){
4.   //Creating a list of fruits
5.   List<String> list1=new ArrayList<String>();
6.   list1.add("Mango");
7.   list1.add("Apple");
8.   list1.add("Banana");
9.   list1.add("Grapes");
10.  //Sorting the list
11.  Collections.sort(list1);
12.   //Traversing list through the for-each loop
13.  for(String fruit:list1)
14.    System.out.println(fruit);
15.
16. System.out.println("Sorting numbers...");
17.  //Creating a list of numbers
18.  List<Integer> list2=new ArrayList<Integer>();
19.  list2.add(21);
20.  list2.add(11);
21.  list2.add(51);
22.  list2.add(1);
23.  //Sorting the list
24.  Collections.sort(list2);
25.   //Traversing list through the for-each loop
26.  for(Integer number:list2)
27.    System.out.println(number);
28. }
29.
30.}
```

Output:

```
Apple
Banana
Grapes
Mango
Sorting numbers...
1
11
21
51
```

Java ListIterator Interface

ListIterator Interface is used to traverse the element in a backward and forward direction.

ListIterator Interface declaration

```
1. public interface ListIterator<E> extends Iterator<E>
```

Methods of Java ListIterator Interface:

Method	Description
void add(E e)	This method inserts the specified element into the list.
boolean hasNext()	This method returns true if the list iterator has more elements while traversing the list in the forward direction.
E next()	This method returns the next element in the list and advances the cursor position.
int nextIndex()	This method returns the index of the element that would be returned by a subsequent call to next()
boolean hasPrevious()	This method returns true if this list iterator has more elements while traversing the list in the reverse direction.
E previous()	This method returns the previous element in the list and moves the cursor position backward.
E previousIndex()	This method returns the index of the element that would be returned by a subsequent call to previous().
void remove()	This method removes the last element from the list that was returned by next() or previous() methods
void set(E e)	This method replaces the last element returned by next() or previous() methods with the specified element.

Example of ListIterator Interface

```
1. import java.util.*;
2. public class ListIteratorExample1{
3. public static void main(String args[]){
4. List<String> al=new ArrayList<String>();
5.         al.add("Amit");
6.         al.add("Vijay");
7.         al.add("Kumar");
8.         al.add(1,"Sachin");
```

```
9.        ListIterator<String> itr=al.listIterator();
10.       System.out.println("Traversing elements in forward direction");
11.       while(itr.hasNext()){
12.
13.       System.out.println("index:"+itr.nextIndex()+" value:"+itr.next());
14.       }
15.       System.out.println("Traversing elements in backward direction");
16.       while(itr.hasPrevious()){
17.
18.       System.out.println("index:"+itr.previousIndex()+" value:"+itr.previous());
19.       }
20.}
21.}
```

Output:

```
Traversing elements in forward direction
index:0 value:Amit
index:1 value:Sachin
index:2 value:Vijay
index:3 value:Kumar
Traversing elements in backward direction
index:3 value:Kumar
index:2 value:Vijay
index:1 value:Sachin
index:0 value:Amit
```

Example of List: Book

Let's see an example of List where we are adding the Books.

```
1.  import java.util.*;
2.  class Book {
3.  int id;
4.  String name,author,publisher;
5.  int quantity;
6.  public Book(int id, String name, String author, String publisher, int quantity) {

7.      this.id = id;
8.      this.name = name;
9.      this.author = author;
10.     this.publisher = publisher;
11.     this.quantity = quantity;
12.}
13.}
14. public class ListExample5 {
15. public static void main(String[] args) {
```

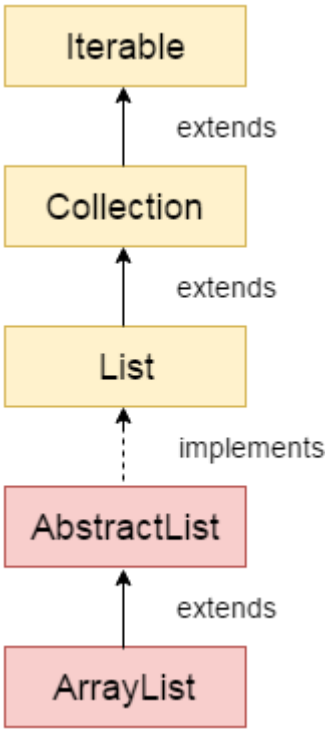
```
16.    //Creating list of Books
17.    List<Book> list=new ArrayList<Book>();
18.    //Creating Books
19.    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
20.    Book b2=new Book(102,"Data Communications and Networking","Forouza
       n","Mc Graw Hill",4);
21.    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
22.    //Adding Books to list
23.    list.add(b1);
24.    list.add(b2);
25.    list.add(b3);
26.    //Traversing list
27.    for(Book b:list){
28.    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.qu
       antity);
29.    }
30.}
31.}
```

Test it Now

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications and Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

Java ArrayList



Java **ArrayList** class uses a *dynamic array* for storing the elements. It is like an array, but there is *no size limit*. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the *java.util* package. It is like the Vector in C++.

The ArrayList in Java can have the duplicate elements also. It implements the List interface so we can use all the methods of the List interface here. The ArrayList maintains the insertion order internally.

It inherits the AbstractList class and implements List interface.

The important points about the Java ArrayList class are:

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.

Method	Description
void add(int index, E element)	It is used to insert the specified element at the specified position in a list.
boolean add(E e)	It is used to append the specified element at the end of a list.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean addAll(int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void clear()	It is used to remove all of the elements from this list.
void ensureCapacity(int requiredCapacity)	It is used to enhance the capacity of an ArrayList instance.
E get(int index)	It is used to fetch the element from the particular position of the list.
boolean isEmpty()	It returns true if the list is empty, otherwise false.
Iterator()	
listIterator()	
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
<T> T[] toArray(T[] a)	It is used to return an array containing all of the elements in this list in the correct order.
Object clone()	It is used to return a shallow copy of an ArrayList.
boolean contains(Object o)	It returns true if the list contains the specified element.
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
E remove(int index)	It is used to remove the element present at the specified position in the list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element.

boolean removeAll(Collection<?> c)	It is used to remove all the elements from the list.
boolean removeIf(Predicate<? super E> filter)	It is used to remove all the elements from the list that satisfies the given predicate.
protected void removeRange(int fromIndex, int toIndex)	It is used to remove all the elements lies within the given range.
void replaceAll(UnaryOperator<E> operator)	It is used to replace all the elements from the list with the specified element.
void retainAll(Collection<?> c)	It is used to retain all the elements in the list that are present in the specified collection.
E set(int index, E element)	It is used to replace the specified element in the list, present at the specified position.
void sort(Comparator<? super E> c)	It is used to sort the elements of the list on the basis of the specified comparator.
Spliterator<E> spliterator()	It is used to create a spliterator over the elements in a list.
List<E> subList(int fromIndex, int toIndex)	It is used to fetch all the elements that lies within the given range.
int size()	It is used to return the number of elements present in the list.
void trimToSize()	It is used to trim the capacity of this ArrayList instance to be the list's current size.

- Java ArrayList allows random access because the array works on an index basis.
- In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.
- We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases. For example:

```
1. ArrayList<int> al = ArrayList<int>(); // does not work
2. ArrayList<Integer> al = new ArrayList<Integer>(); // works fine
```

- Java ArrayList gets initialized by the size. The size is dynamic in the array list, which varies according to the elements getting added or removed from the list.

Hierarchy of ArrayList class

As shown in the above diagram, the Java ArrayList class extends AbstractList class which implements the List interface. The List interface extends the Collection and Iterable interfaces in hierarchical order.

ArrayList class declaration

Let's see the declaration for java.util.ArrayList class.

```
1. public class ArrayList<E> extends AbstractList<E> implements List<E>, Ran
   domAccess, Cloneable, Serializable
```

Constructors of ArrayList

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection<? extends E> c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Methods of ArrayList

Java Non-generic Vs. Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.

Java new generic collection allows you to have only one type of object in a collection. Now it is type-safe, so typecasting is not required at runtime.

Let's see the old non-generic example of creating a Java collection.

```
1. ArrayList list=new ArrayList();//creating old non-generic arraylist
```

Let's see the new generic example of creating java collection.

```
1. ArrayList<String> list=new ArrayList<String>();//creating new generic arraylist
```

In a generic collection, we specify the type in angular braces. Now ArrayList is forced to have the only specified type of object in it. If you try to add another type of object, it gives a *compile-time error*.

For more information on Java generics, click here Java Generics Tutorial.

Java ArrayList Example

FileName: ArrayListExample1.java

```
1. import java.util.*;
2.  public class ArrayListExample1{
```

```
3.   public static void main(String args[]){
4.    ArrayList<String> list=new ArrayList<String>();//Creating arraylist
5.       list.add("Mango");//Adding object in arraylist
6.       list.add("Apple");
7.       list.add("Banana");
8.       list.add("Grapes");
9.       //Printing the arraylist object
10.      System.out.println(list);
11. }
12.}
```

Test it Now

Output:

```
[Mango, Apple, Banana, Grapes]
```

Iterating ArrayList using Iterator

Let's see an example to traverse ArrayList elements using the Iterator interface.

FileName: ArrayListExample2.java

```
1.  import java.util.*;
2.  public class ArrayListExample2{
3.   public static void main(String args[]){
4.    ArrayList<String> list=new ArrayList<String>();//Creating arraylist
5.    list.add("Mango");//Adding object in arraylist
6.    list.add("Apple");
7.    list.add("Banana");
8.    list.add("Grapes");
9.    //Traversing list through Iterator
10.   Iterator itr=list.iterator();//getting the Iterator
11.   while(itr.hasNext()){//check if iterator has the elements
12.    System.out.println(itr.next());//printing the element and move to next
13.   }
14. }
15.}
```

Test it Now

Output:

```
Mango
Apple
Banana
Grapes
```

Iterating ArrayList using For-each loop

Let's see an example to traverse the ArrayList elements using the for-each loop

FileName: ArrayListExample3.java

```
1. import java.util.*;
2. public class ArrayListExample3{
3.  public static void main(String args[]){
4.   ArrayList<String> list=new ArrayList<String>();//Creating arraylist
5.   list.add("Mango");//Adding object in arraylist
6.   list.add("Apple");
7.   list.add("Banana");
8.   list.add("Grapes");
9.   //Traversing list through for-each loop
10.  for(String fruit:list)
11.    System.out.println(fruit);
12.
13. }
14.}
```

Output:

Test it Now

```
Mango
Apple
Banana
Grapes
```

Get and Set ArrayList

The *get() method* returns the element at the specified index, whereas the *set() method* changes the element.

FileName: ArrayListExample4.java

```
1. import java.util.*;
2. public class ArrayListExample4{
3.  public static void main(String args[]){
4.   ArrayList<String> al=new ArrayList<String>();
5.   al.add("Mango");
6.   al.add("Apple");
7.   al.add("Banana");
8.   al.add("Grapes");
9.   //accessing the element
10.  System.out.println("Returning element: "+al.get(1));//it will return the 2nd ele
     ment, because index starts from 0
11.  //changing the element
12.  al.set(1,"Dates");
```

```
13.  //Traversing list
14.  for(String fruit:al)
15.    System.out.println(fruit);
16.
17. }
18.}
```

Test it Now

Output:

```
Returning element: Apple
Mango
Dates
Banana
Grapes
```

How to Sort ArrayList

The *java.util* package provides a utility class **Collections**, which has the static method sort(). Using the **Collections.sort()** method, we can easily sort the ArrayList.

FileName: SortArrayList.java

```
1.  import java.util.*;
2.  class SortArrayList{
3.   public static void main(String args[]){
4.    //Creating a list of fruits
5.    List<String> list1=new ArrayList<String>();
6.    list1.add("Mango");
7.    list1.add("Apple");
8.    list1.add("Banana");
9.    list1.add("Grapes");
10.  //Sorting the list
11.  Collections.sort(list1);
12.   //Traversing list through the for-each loop
13.  for(String fruit:list1)
14.    System.out.println(fruit);
15.
16. System.out.println("Sorting numbers...");
17.  //Creating a list of numbers
18.  List<Integer> list2=new ArrayList<Integer>();
19.  list2.add(21);
20.  list2.add(11);
21.  list2.add(51);
22.  list2.add(1);
23.  //Sorting the list
```

```
24.  Collections.sort(list2);
25.   //Traversing list through the for-each loop
26.  for(Integer number:list2)
27.    System.out.println(number);
28. }
29.
30.}
```

Output:

```
Apple
Banana
Grapes
Mango
Sorting numbers...
1
11
21
51
```

Ways to iterate the elements of the collection in Java

There are various ways to traverse the collection elements:

1. By Iterator interface.
2. By for-each loop.
3. By ListIterator interface.
4. By for loop.
5. By forEach() method.
6. By forEachRemaining() method.

Iterating Collection through remaining ways

Let's see an example to traverse the ArrayList elements through other ways

FileName: ArrayList4.java

```
1.  import java.util.*;
2.  class ArrayList4{
3.   public static void main(String args[]){
4.      ArrayList<String> list=new ArrayList<String>();//Creating arraylist
5.            list.add("Ravi");//Adding object in arraylist
6.            list.add("Vijay");
7.            list.add("Ravi");
8.            list.add("Ajay");
9.
10.           System.out.println("Traversing list through List Iterator:");
```

```
11.         //Here, element iterates in reverse order
12.           ListIterator<String> list1=list.listIterator(list.size());
13.           while(list1.hasPrevious())
14.           {
15.               String str=list1.previous();
16.               System.out.println(str);
17.           }
18.       System.out.println("Traversing list through for loop:");
19.         for(int i=0;i<list.size();i++)
20.         {
21.          System.out.println(list.get(i));
22.         }
23.
24.       System.out.println("Traversing list through forEach() method:");
25.       //The forEach() method is a new feature, introduced in Java 8.
26.          list.forEach(a->{ //Here, we are using lambda expression
27.             System.out.println(a);
28.           });
29.
30.          System.out.println("Traversing list through forEachRemaining() method
    :");
31.           Iterator<String> itr=list.iterator();
32.           itr.forEachRemaining(a-> //Here, we are using lambda expression
33.           {
34.          System.out.println(a);
35.           });
36. }
37.}
```

Output:

```
Traversing list through List Iterator:
Ajay
Ravi
Vijay
Ravi
Traversing list through for loop:
Ravi
Vijay
Ravi
Ajay
Traversing list through forEach() method:
Ravi
Vijay
Ravi
Ajay
Traversing list through forEachRemaining() method:
Ravi
Vijay
Ravi
```

```
Ajay
```

User-defined class objects in Java ArrayList

Let's see an example where we are storing Student class object in an array list.

FileName: ArrayList5.java

```
1.  class Student{
2.    int rollno;
3.    String name;
4.    int age;
5.    Student(int rollno,String name,int age){
6.     this.rollno=rollno;
7.     this.name=name;
8.     this.age=age;
9.    }
10. }
```

```
1.  import java.util.*;
2.   class ArrayList5{
3.   public static void main(String args[]){
4.    //Creating user-defined class objects
5.    Student s1=new Student(101,"Sonoo",23);
6.    Student s2=new Student(102,"Ravi",21);
7.    Student s2=new Student(103,"Hanumat",25);
8.    //creating arraylist
9.    ArrayList<Student> al=new ArrayList<Student>();
10.   al.add(s1);//adding Student class object
11.   al.add(s2);
12.   al.add(s3);
13.   //Getting Iterator
14.   Iterator itr=al.iterator();
15.   //traversing elements of ArrayList object
16.   while(itr.hasNext()){
17.     Student st=(Student)itr.next();
18.     System.out.println(st.rollno+" "+st.name+" "+st.age);
19.   }
20.  }
21. }
```

Output:

```
        101 Sonoo 23
```

```
102 Ravi 21
103 Hanumat 25
```

Java ArrayList Serialization and Deserialization Example

Let's see an example to serialize an ArrayList object and then deserialize it.

FileName: ArrayList6.java

```
1.  import java.io.*;
2.  import java.util.*;
3.   class ArrayList6 {
4.
5.        public static void main(String [] args)
6.        {
7.          ArrayList<String> al=new ArrayList<String>();
8.          al.add("Ravi");
9.          al.add("Vijay");
10.         al.add("Ajay");
11.
12.         try
13.         {
14.            //Serialization
15.            FileOutputStream fos=new FileOutputStream("file");
16.            ObjectOutputStream oos=new ObjectOutputStream(fos);
17.            oos.writeObject(al);
18.            fos.close();
19.            oos.close();
20.            //Deserialization
21.            FileInputStream fis=new FileInputStream("file");
22.            ObjectInputStream ois=new ObjectInputStream(fis);
23.          ArrayList  list=(ArrayList)ois.readObject();
24.          System.out.println(list);
25.         }catch(Exception e)
26.         {
27.            System.out.println(e);
28.         }
29.      }
30.   }
```

Output:

```
[Ravi, Vijay, Ajay]
```

Java ArrayList example to add elements

Here, we see different ways to add an element.

FileName: ArrayList7.java

```
1. import java.util.*;
2. class ArrayList7{
3. public static void main(String args[]){
4. ArrayList<String> al=new ArrayList<String>();
5.         System.out.println("Initial list of elements: "+al);
6.         //Adding elements to the end of the list
7.         al.add("Ravi");
8.         al.add("Vijay");
9.         al.add("Ajay");
10.        System.out.println("After invoking add(E e) method: "+al);
11.        //Adding an element at the specific position
12.        al.add(1, "Gaurav");
13.        System.out.println("After invoking add(int index, E element) method: "+
    al);
14.        ArrayList<String> al2=new ArrayList<String>();
15.        al2.add("Sonoo");
16.        al2.add("Hanumat");
17.        //Adding second list elements to the first list
18.        al.addAll(al2);
19.        System.out.println("After invoking addAll(Collection<? extends E> c) m
    ethod: "+al);
20.        ArrayList<String> al3=new ArrayList<String>();
21.        al3.add("John");
22.        al3.add("Rahul");
23.        //Adding second list elements to the first list at specific position
24.        al.addAll(1, al3);
25.        System.out.println("After invoking addAll(int index, Collection<? extend
    s E> c) method: "+al);
26.
27. }
28.}
```

Output:

```
Initial list of elements: []
After invoking add(E e) method: [Ravi, Vijay, Ajay]
After invoking add(int index, E element) method: [Ravi, Gaurav, Vijay, Ajay]
After invoking addAll(Collection<? extends E> c) method:
[Ravi, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addAll(int index, Collection<? extends E> c) method:
[Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
```

Java ArrayList example to remove elements

Here, we see different ways to remove an element.

FileName: ArrayList8.java

```
1.  import java.util.*;
2.   class ArrayList8 {
3.
4.       public static void main(String [] args)
5.       {
6.         ArrayList<String> al=new ArrayList<String>();
7.         al.add("Ravi");
8.         al.add("Vijay");
9.         al.add("Ajay");
10.        al.add("Anuj");
11.        al.add("Gaurav");
12.        System.out.println("An initial list of elements: "+al);
13.        //Removing specific element from arraylist
14.        al.remove("Vijay");
15.        System.out.println("After invoking remove(object) method: "+al);
16.        //Removing element on the basis of specific position
17.        al.remove(0);
18.        System.out.println("After invoking remove(index) method: "+al);
19.
20.        //Creating another arraylist
21.        ArrayList<String> al2=new ArrayList<String>();
22.        al2.add("Ravi");
23.        al2.add("Hanumat");
24.        //Adding new elements to arraylist
25.        al.addAll(al2);
26.        System.out.println("Updated list : "+al);
27.        //Removing all the new elements from arraylist
28.        al.removeAll(al2);
29.        System.out.println("After invoking removeAll() method: "+al);
30.        //Removing elements on the basis of specified condition
31.        al.removeIf(str -
    > str.contains("Ajay"));   //Here, we are using Lambda expression
32.        System.out.println("After invoking removeIf() method: "+al);
33.        //Removing all the elements available in the list
34.        al.clear();
35.        System.out.println("After invoking clear() method: "+al);
36.     }
```

```
37.    }
```

Output:

```
An initial list of elements: [Ravi, Vijay, Ajay, Anuj, Gaurav]
After invoking remove(object) method: [Ravi, Ajay, Anuj, Gaurav]
After invoking remove(index) method: [Ajay, Anuj, Gaurav]
Updated list : [Ajay, Anuj, Gaurav, Ravi, Hanumat]
After invoking removeAll() method: [Ajay, Anuj, Gaurav]
After invoking removeIf() method: [Anuj, Gaurav]
After invoking clear() method: []
```

Java ArrayList example of retainAll() method

FileName: ArrayList9.java

```
1.  import java.util.*;
2.  class ArrayList9{
3.   public static void main(String args[]){
4.    ArrayList<String> al=new ArrayList<String>();
5.    al.add("Ravi");
6.    al.add("Vijay");
7.    al.add("Ajay");
8.    ArrayList<String> al2=new ArrayList<String>();
9.    al2.add("Ravi");
10.   al2.add("Hanumat");
11.   al.retainAll(al2);
12.   System.out.println("iterating the elements after retaining the elements of al2"
      );
13.   Iterator itr=al.iterator();
14.   while(itr.hasNext()){
15.    System.out.println(itr.next());
16.   }
17. }
18.}
```

Output:

```
       iterating the elements after retaining the elements of al2
       Ravi
```

Java ArrayList example of isEmpty() method

FileName: ArrayList4.java

```
1.  import java.util.*;
2.   class ArrayList10{
3.
```

```
4.        public static void main(String [] args)
5.        {
6.          ArrayList<String> al=new ArrayList<String>();
7.          System.out.println("Is ArrayList Empty: "+al.isEmpty());
8.          al.add("Ravi");
9.          al.add("Vijay");
10.         al.add("Ajay");
11.         System.out.println("After Insertion");
12.         System.out.println("Is ArrayList Empty: "+al.isEmpty());
13.      }
14.   }
```

Output:

```
Is ArrayList Empty: true
After Insertion
Is ArrayList Empty: false
```

Java ArrayList Example: Book

Let's see an ArrayList example where we are adding books to the list and printing all the books.

FileName: ArrayListExample20.java

```
1. import java.util.*;
2. class Book {
3. int id;
4. String name,author,publisher;
5. int quantity;
6. public Book(int id, String name, String author, String publisher, int quantity) {

7.     this.id = id;
8.     this.name = name;
9.     this.author = author;
10.    this.publisher = publisher;
11.    this.quantity = quantity;
12.}
13.}
14.public class ArrayListExample20 {
15.public static void main(String[] args) {
16.    //Creating list of Books
17.    List<Book> list=new ArrayList<Book>();
18.    //Creating Books
19.    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
```

```
20.     Book b2=new Book(102,"Data Communications and Networking","Forouza
    n","Mc Graw Hill",4);
21.     Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
22.     //Adding Books to list
23.     list.add(b1);
24.     list.add(b2);
25.     list.add(b3);
26.     //Traversing list
27.     for(Book b:list){
28.         System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.
    quantity);
29.     }
30. }
31. }
```

Test it Now

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications and Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

Size and Capacity of an ArrayList

Size and capacity of an array list are the two terms that beginners find confusing. Let's understand it in this section with the help of some examples. Consider the following code snippet.

FileName: SizeCapacity.java

```
1.  import java.util.*;
2.
3.  public class SizeCapacity
4.  {
5.
6.  public static void main(String[] args) throws Exception
7.  {
8.
9.      ArrayList<Integer> al = new ArrayList<Integer>();
10.
11.     System.out.println("The size of the array is: " + al.size());
12. }
13. }
```

Output:

```
The size of the array is: 0
```

Explanation: The output makes sense as we have not done anything with the array list. Now observe the following program.

FileName: SizeCapacity1.java

```
1. import java.util.*;
2.
3. public class SizeCapacity1
4. {
5.
6. public static void main(String[] args) throws Exception
7. {
8.
9.     ArrayList<Integer> al = new ArrayList<Integer>(10);
10.
11.    System.out.println("The size of the array is: " + al.size());
12.}
13.}
```

Output:

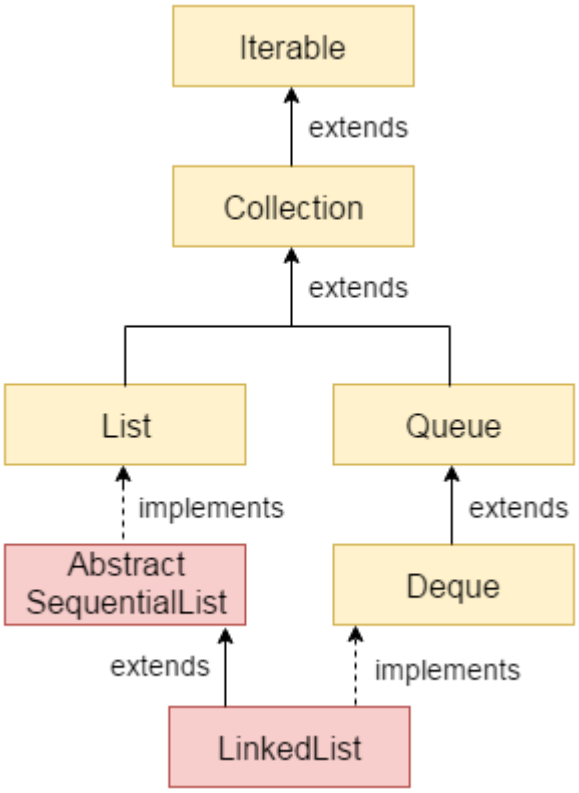
```
The size of the array is: 0
```

Explanation: We see that the size is still 0, and the reason behind this is the number 10 represents the capacity no the size. In fact, the size represents the total number of elements present in the array. As we have not added any element, therefore, the size of the array list is zero in both programs.

Capacity represents the total number of elements the array list can contain. Therefore, the capacity of an array list is always greater than or equal to the size of the array list. When we add an element to the array list, it checks whether the size of the array list has become equal to the capacity or not. If yes, then the capacity of the array list increases. So, in the above example, the capacity will be 10 till 10 elements are added to the list. When we add the 11^{th} element, the capacity increases. Note that in both examples, the capacity of the array list is 10. In the first case, the capacity is 10 because the default capacity of the array list is 10. In the second case, we have explicitly mentioned that the capacity of the array list is 10.

Note: There is no any standard method to tell how the capacity increases in the array list. In fact, the way the capacity increases vary from one GDK version to the other version. Therefore, it is required to check the way capacity increases code is implemented in the GDK. There is no any pre-defined method in the ArrayList class that returns the capacity of the array list. Therefore, for better understanding, use the capacity() method of the Vector class. The logic of the size and the capacity is the same in the ArrayList class and the Vector class.

Java LinkedList class



Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to occur.
- Java LinkedList class can be used as a list, stack or queue.

Hierarchy of LinkedList class

As shown in the above diagram, Java LinkedList class extends AbstractSequentialList class and implements List and Deque interfaces.

Doubly Linked List

In the case of a doubly linked list, we can add or remove elements from both sides.

Method	Description
boolean add(E e)	It is used to append the specified element to the end of a list.
void add(int index, E element)	It is used to insert the specified element at the specified position index in a list.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean addAll(int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void addFirst(E e)	It is used to insert the given element at the beginning of a list.
void addLast(E e)	It is used to append the given element to the end of a list.
void clear()	It is used to remove all the elements from a list.
Object clone()	It is used to return a shallow copy of an ArrayList.
boolean contains(Object o)	It is used to return true if a list contains a specified element.
Iterator<E> descendingIterator()	It is used to return an iterator over the elements in a deque in reverse sequential order.
E element()	It is used to retrieve the first element of a list.
E get(int index)	It is used to return the element at the specified position in a list.
E getFirst()	It is used to return the first element in a list.
E getLast()	It is used to return the last element in a list.
int indexOf(Object o)	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
int lastIndexOf(Object o)	It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element.
ListIterator<E> listIterator(int index)	It is used to return a list-iterator of the elements in proper sequence, starting at the specified position in the list.
boolean offer(E e)	It adds the specified element as the last element of a list.
boolean offerFirst(E e)	It inserts the specified element at the front of a list.

boolean offerLast(E e)	It inserts the specified element at the end of a list.
E peek()	It retrieves the first element of a list
E peekFirst()	It retrieves the first element of a list or returns null if a list is empty.
E peekLast()	It retrieves the last element of a list or returns null if a list is empty.
E poll()	It retrieves and removes the first element of a list.
E pollFirst()	It retrieves and removes the first element of a list, or returns null if a list is empty.
E pollLast()	It retrieves and removes the last element of a list, or returns null if a list is empty.
E pop()	It pops an element from the stack represented by a list.
void push(E e)	It pushes an element onto the stack represented by a list.
E remove()	It is used to retrieve and removes the first element of a list.
E remove(int index)	It is used to remove the element at the specified position in a list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element in a list.
E removeFirst()	It removes and returns the first element from a list.
boolean removeFirstOccurrence(Object o)	It is used to remove the first occurrence of the specified element in a list (when traversing the list from head to tail).
E removeLast()	It removes and returns the last element from a list.
boolean removeLastOccurrence(Object o)	It removes the last occurrence of the specified element in a list (when traversing the list from head to tail).
E set(int index, E element)	It replaces the element at the specified position in a list with the specified element.
Object[] toArray()	It is used to return an array containing all the elements in a list in proper sequence (from first to the last element).
<T> T[] toArray(T[] a)	It returns an array containing all the elements in the proper sequence (from first to the last element); the runtime type of the returned array is that of the specified array.
int size()	It is used to return the number of elements in a list.

fig- doubly linked list

LinkedList class declaration

Let's see the declaration for java.util.LinkedList class.

1. **public class** LinkedList<E> **extends** AbstractSequentialList<E> **implements** List<E>, Deque<E>, Cloneable, Serializable

Constructors of Java LinkedList

Constructor	Description
LinkedList()	It is used to construct an empty list.
LinkedList(Collection<? extends E> c)	It is used to construct a list containing the elements of the spec collection, in the order, they are returned by the collection's iterat

Methods of Java LinkedList

Java LinkedList Example

```
1. import java.util.*;
2. public class LinkedList1{
3.  public static void main(String args[]){
4.
5.   LinkedList<String> al=new LinkedList<String>();
6.   al.add("Ravi");
7.   al.add("Vijay");
8.   al.add("Ravi");
9.   al.add("Ajay");
10.
11.  Iterator<String> itr=al.iterator();
12.  while(itr.hasNext()){
13.   System.out.println(itr.next());
14.  }
15. }
16.}
```

```
Output: Ravi
        Vijay
        Ravi
```

Java LinkedList example to add elements

Here, we see different ways to add elements.

```
1. import java.util.*;
2. public class LinkedList2{
3.  public static void main(String args[]){
4.  LinkedList<String> ll=new LinkedList<String>();
5.          System.out.println("Initial list of elements: "+ll);
6.          ll.add("Ravi");
7.          ll.add("Vijay");
8.          ll.add("Ajay");
9.          System.out.println("After invoking add(E e) method: "+ll);
10.         //Adding an element at the specific position
11.         ll.add(1, "Gaurav");
12.         System.out.println("After invoking add(int index, E element) method: "+
    ll);
13.         LinkedList<String> ll2=new LinkedList<String>();
14.         ll2.add("Sonoo");
15.         ll2.add("Hanumat");
16.         //Adding second list elements to the first list
17.         ll.addAll(ll2);
18.         System.out.println("After invoking addAll(Collection<? extends E> c) m
    ethod: "+ll);
19.         LinkedList<String> ll3=new LinkedList<String>();
20.         ll3.add("John");
21.         ll3.add("Rahul");
22.         //Adding second list elements to the first list at specific position
23.         ll.addAll(1, ll3);
24.         System.out.println("After invoking addAll(int index, Collection<? extend
    s E> c) method: "+ll);
25.         //Adding an element at the first position
26.         ll.addFirst("Lokesh");
27.         System.out.println("After invoking addFirst(E e) method: "+ll);
28.         //Adding an element at the last position
29.         ll.addLast("Harsh");
30.         System.out.println("After invoking addLast(E e) method: "+ll);
31.
32.  }
33. }
```

```
Initial list of elements: []
After invoking add(E e) method: [Ravi, Vijay, Ajay]
After invoking add(int index, E element) method: [Ravi, Gaurav, Vijay, Ajay]
After invoking addAll(Collection<? extends E> c) method:
[Ravi, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addAll(int index, Collection<? extends E> c) method:
[Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addFirst(E e) method:
[Lokesh, Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addLast(E e) method:
[Lokesh, Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat, Harsh]
```

Java LinkedList example to remove elements

Here, we see different ways to remove an element.

```
1.  import java.util.*;
2.  public class LinkedList3 {
3.
4.      public static void main(String [] args)
5.      {
6.        LinkedList<String> ll=new LinkedList<String>();
7.        ll.add("Ravi");
8.        ll.add("Vijay");
9.        ll.add("Ajay");
10.       ll.add("Anuj");
11.       ll.add("Gaurav");
12.       ll.add("Harsh");
13.       ll.add("Virat");
14.       ll.add("Gaurav");
15.       ll.add("Harsh");
16.       ll.add("Amit");
17.       System.out.println("Initial list of elements: "+ll);
18.     //Removing specific element from arraylist
19.         ll.remove("Vijay");
20.         System.out.println("After invoking remove(object) method: "+ll);
21.     //Removing element on the basis of specific position
22.         ll.remove(0);
23.         System.out.println("After invoking remove(index) method: "+ll);
24.         LinkedList<String> ll2=new LinkedList<String>();
25.         ll2.add("Ravi");
26.         ll2.add("Hanumat");
27.     // Adding new elements to arraylist
28.         ll.addAll(ll2);
29.         System.out.println("Updated list : "+ll);
30.     //Removing all the new elements from arraylist
```

```
31.             ll.removeAll(ll2);
32.             System.out.println("After invoking removeAll() method: "+ll);
33.         //Removing first element from the list
34.             ll.removeFirst();
35.             System.out.println("After invoking removeFirst() method: "+ll);
36.          //Removing first element from the list
37.             ll.removeLast();
38.             System.out.println("After invoking removeLast() method: "+ll);
39.          //Removing first occurrence of element from the list
40.             ll.removeFirstOccurrence("Gaurav");
41.             System.out.println("After invoking removeFirstOccurrence() method: "
    +ll);
42.          //Removing last occurrence of element from the list
43.             ll.removeLastOccurrence("Harsh");
44.             System.out.println("After invoking removeLastOccurrence() method: "
    +ll);
45.
46.             //Removing all the elements available in the list
47.             ll.clear();
48.             System.out.println("After invoking clear() method: "+ll);
49.       }
50.    }
```

```
Initial list of elements: [Ravi, Vijay, Ajay, Anuj, Gaurav, Harsh, Virat,
Gaurav, Harsh, Amit]
After invoking remove(object) method: [Ravi, Ajay, Anuj, Gaurav, Harsh,
Virat, Gaurav, Harsh, Amit]
After invoking remove(index) method: [Ajay, Anuj, Gaurav, Harsh, Virat,
Gaurav, Harsh, Amit]
Updated list : [Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav, Harsh, Amit, Ravi,
Hanumat]
After invoking removeAll() method: [Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav,
Harsh, Amit]
After invoking removeFirst() method: [Gaurav, Harsh, Virat, Gaurav, Harsh,
Amit]
After invoking removeLast() method: [Gaurav, Harsh, Virat, Gaurav, Harsh]
After invoking removeFirstOccurrence() method: [Harsh, Virat, Gaurav, Harsh]
After invoking removeLastOccurrence() method: [Harsh, Virat, Gaurav]
After invoking clear() method: []
```

Java LinkedList Example to reverse a list of elements

```
1.  import java.util.*;
2.  public class LinkedList4{
3.   public static void main(String args[]){
4.
5.    LinkedList<String> ll=new LinkedList<String>();
6.             ll.add("Ravi");
7.             ll.add("Vijay");
```

```
        ll.add("Ajay");
        //Traversing the list of elements in reverse order
        Iterator i=ll.descendingIterator();
        while(i.hasNext())
        {
            System.out.println(i.next());
        }

 }
}
```

```
Output: Ajay
Vijay
Ravi
```

Java LinkedList Example: Book

```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {

    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class LinkedListExample {
public static void main(String[] args) {
    //Creating list of Books
    List<Book> list=new LinkedList<Book>();
    //Creating Books
    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
    Book b2=new Book(102,"Data Communications & Networking","Forouzan",
"Mc Graw Hill",4);
    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
    //Adding Books to list
    list.add(b1);
    list.add(b2);
```

```
25.     list.add(b3);
26.     //Traversing list
27.     for(Book b:list){
28.     System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.qu
        antity);
29.     }
30. }
31. }
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

Difference Between ArrayList and LinkedList

ArrayList and LinkedList both implement the List interface and maintain insertion order. Both are non-synchronized classes.

However, there are many differences between the ArrayList and LinkedList classes that are given below.

ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) An ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.
5) The memory location for the elements of an ArrayList is contiguous.	The location for the elements of a linked list is not contagious.
6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList.	There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized.
7) To be precise, an ArrayList is a resizable array.	LinkedList implements the doubly linked list of the list interface.

Example of ArrayList and LinkedList in Java

Let's see a simple example where we are using ArrayList and LinkedList both.

FileName: TestArrayLinked.java

```
1. import java.util.*;
2. class TestArrayLinked{
3.  public static void main(String args[]){
4.
5.   List<String> al=new ArrayList<String>();//creating arraylist
6.   al.add("Ravi");//adding object in arraylist
7.   al.add("Vijay");
8.   al.add("Ravi");
```

```
9.    al.add("Ajay");
10.
11.   List<String> al2=new LinkedList<String>();//creating linkedlist
12.   al2.add("James");//adding object in linkedlist
13.   al2.add("Serena");
14.   al2.add("Swati");
15.   al2.add("Junaid");
16.
17.   System.out.println("arraylist: "+al);
18.   System.out.println("linkedlist: "+al2);
19.  }
20.}
```

Test it Now

Output:

```
arraylist: [Ravi,Vijay,Ravi,Ajay]
linkedlist: [James,Serena,Swati,Junaid]
```

Points to Remember

The following are some important points to remember regarding an ArrayList and LinkedList.

- When the rate of addition or removal rate is more than the read scenarios, then go for the LinkedList. On the other hand, when the frequency of the read scenarios is more than the addition or removal rate, then ArrayList takes precedence over LinkedList.
- Since the elements of an ArrayList are stored more compact as compared to a LinkedList; therefore, the ArrayList is more cache-friendly as compared to the LinkedList. Thus, chances for the cache miss are less in an ArrayList as compared to a LinkedList. Generally, it is considered that a LinkedList is poor in cache-locality.
- Memory overhead in the LinkedList is more as compared to the ArrayList. It is because, in a LinkedList, we have two extra links (next and previous) as it is required to store the address of the previous and the next nodes, and these links consume extra space. Such links are not present in an ArrayList.