SNo	ArrayList Methods with Description
1	**void add(int index, Object element)** Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index > size()).
2	**boolean add(Object o)** Appends the specified element to the end of this list.
3	**boolean addAll(Collection c)** Appends 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. Throws NullPointerException if the specified collection is null.
4	**boolean addAll(int index, Collection c)** Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null.
5	**void clear()** Removes all of the elements from this list.
6	**Object clone()** Returns a shallow copy of this ArrayList.
7	**boolean contains(Object o)** Returns true if this list contains the specified element.
8	**void ensureCapacity(int minCapacity)** Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
9	**Object get(int index)** Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size()).
10	**int indexOf(Object o)** Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
11	**int lastIndexOf(Object o)** Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
12	**Object remove(int index)** Removes the element at the specified position in this list. Throws IndexOutOfBoundsException if index out of range (index < 0 || index >=

	size()).
13	**protected void removeRange(int fromIndex, int toIndex)** Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
14	**Object set(int index, Object element)** Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).
15	**int size()** Returns the number of elements in this list.
16	**Object[] toArray()** Returns an array containing all of the elements in this list in the correct order. Throws NullPointerException if the specified array is null.
17	**void trimToSize()** Trims the capacity of this ArrayList instance to be the list's current size.
18	**boolean isEmpty()** **isEmpty() method of Java ArrayList to check whether ArrayList is empty**. isEmpty() method returns true if this ArrayList contains no elements. You can also use size() method of List to check if List is empty

Example of ArrayList Methods

```
ArrayList<String> stringList = new ArrayList<String>();  //Generic ArrayList to Store only String objects
```

1. **Putting an Item into ArrayList**

```
stringList.add("Grapes");
stringList.add(“Apple”);
stringList.add(“Mango”);
```

2. **Checking size of ArrayList**

```
int size = stringList.size();
```

3. Checking Index of an Item in Java ArrayList

```
int index = stringList.indexOf("Apple");
```

4. **Retrieving Item from ArrayList in a loop**

```
for (int i = 0; i < stringList.size(); i++)
  String item = stringList.get(i);
  System.out.println("Item " + i + " : " + item);
}
                        [or]
for(String item: stringList){
System.out.println("retrieved element: " + item);
}                       [or]
System.out.println("The Elements are: " + item);
```

5. **Checking ArrayList for an Item**

```
boolean retval = stringList.contains(“Apple”);
```

6. **Removing an Item from ArrayList**

```
stringList.remove(0);
stringList.remove(“Grapes”);
```

7. Copying data from one ArrayList to another ArrayList in Java

```
ArrayList<String> copyOfStringList = new ArrayList<String>();
copyOfStringList.addAll(stringList);
```

8. **Replacing an element at a particular index**

```
stringList.set(1,"Papaya");
```

9. **Converting from ArrayList to Array in Java**

```
String[] itemArray = new String[stringList.size()];
String[] returnedArray = stringList.toArray(itemArray);
```

10. **Clearing all data from ArrayList**

```
stingList.clear();
```

11. clone() Method

```
ArrayList<String> arrl = new ArrayList<String>();
      //adding elements to the end
```

```
arrl.add("First");
arrl.add("Second");
arrl.add("Third");
arrl.add("Fourth");
System.out.println("Actual ArrayList:"+arrl);
ArrayList<String> copy = (ArrayList<String>) arrl.clone();
System.out.println("Cloned ArrayList:"+copy);
```

Sorting an Array

```
ArrayList<String> stringArray = new ArrayList<String>(
Arrays.asList("Hello", "Welcome", "Java", "Object", "Array", "String",
"Inheritance"));
System.out.println("****** Unsorted String Array *******");
//Sort array in ascending order
System.out.println(stringArray);
//Sort array in ascending order
System.out.println(stringArray);
System.out.println("****** Sorted String Array *******");
Collections.sort(stringArray);
System.out.println(stringArray);
//Sort array in reverse order
 Collections.sort(stringArray,Collections.reverseOrder());
 System.out.println("****** Reverse Sorted String Array *******");
 System.out.println(stringArray);
```

More Examples

```
public static void main( String[ ] args)
{
        ArrayList<Integer> myInts = new ArrayList<Integer>(25);
        System.out.println( “Size of myInts = “ + myInts.size());
        for (int k = 0; k < 10; k++)
                myInts.add( 3 * k );
        myInts.set( 6, 44 );
        myInts.add( 4, 42 );
        myInts.remove( new Integer(99) );
        System.out.println( “Size of myInts = “ + myInts.size());
        for (int k = 0; k < myInts.size(); k++)
                System.out.print( myInts.get( k ) + “, “  );
        if (myInts.contains( 57 ) ) System.out.println(“57 found”);
```

```
        System.out.println ("44 found at index " + myInts.indexOf(44));
}
```

LinkedList Methods

SN	Methods with Description
1	**void add(int index, Object element)** Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index > size()).
2	**boolean add(Object o)** Appends the specified element to the end of this list.
3	**boolean addAll(Collection c)** Appends 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. Throws NullPointerException if the specified collection is null.
4	**boolean addAll(int index, Collection c)** Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null.
5	**void addFirst(Object o)** Inserts the given element at the beginning of this list.
6	**void addLast(Object o)** Appends the given element to the end of this list.
7	**void clear()** Removes all of the elements from this list.
8	**Object clone()** Returns a shallow copy of this LinkedList.
9	**boolean contains(Object o)** Returns true if this list contains the specified element.
10	**Object get(int index)** Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).
11	**Object getFirst()** Returns the first element in this list. Throws NoSuchElementException if this list is empty.
12	**Object getLast()** Returns the last element in this list. Throws NoSuchElementException if this list is empty.
13	**int indexOf(Object o)**

	Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
14	**int lastIndexOf(Object o)** Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
15	**ListIterator listIterator(int index)** Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).
16	**Object remove(int index)** Removes the element at the specified position in this list. Throws NoSuchElementException if this list is empty.
17	**boolean remove(Object o)** Removes the first occurrence of the specified element in this list. Throws NoSuchElementException if this list is empty. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).
18	**Object removeFirst()** Removes and returns the first element from this list. Throws NoSuchElementException if this list is empty.
19	**Object removeLast()** Removes and returns the last element from this list. Throws NoSuchElementException if this list is empty.
20	**Object set(int index, Object element)** Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).
21	**int size()** Returns the number of elements in this list.
22	**Object[] toArray()** Returns an array containing all of the elements in this list in the correct order. Throws NullPointerException if the specified array is null.

Example

```
 LinkedList<String> arrl = new LinkedList<String>();
arrl.add("First");
arrl.add("Second");
```

```
arrl.add("Third");
arrl.add("Fourth");
List<String> list = new LinkedList<String>();
list.add("Second");
list.add("Fourth");
System.out.println("Does LinkedList contains all list elements?: "
                +arrl.containsAll(list));
list.add("one");
System.out.println("Does LinkedList contains all list elements?: "
                +arrl.containsAll(list));
```

Output :

```
Does LinkedList contains all list elements?: true
Does LinkedList contains all list elements?: false
```

The Comparator Interface

Comparator *can be used to compare the objects of a class that doesn't implement* **Comparable**.

The **Comparable** interface defines the **compareTo** method, which is used to compare two elements of the same class that implement the **Comparable** interface. What if the elements' classes do not implement the **Comparable** interface or the elements have different types? Can these elements be compared? You can define a *comparator* to compare the elements of different classes. To do so, define a class that implements the **java.util.Comparator<T>** interface.

The **Comparator<T>** interface has two methods,
compare and **equals**.

- **public int** compare(T element1, T element2)

Returns a negative value if **element1** is less than **element2**, a positive value if **element1** is greater than **element2**, and zero if they are equal.

- **public boolean** equals(Object element)

Returns **true** if the specified object is also a comparator and imposes the same ordering as this comparator.

The **equals** method is also defined in the **Object** class. Therefore, you will not get a compile error even if you don't implement the **equals** method in your custom comparator class. However, in some cases implementing this method may improve performance by allowing programs to determine quickly whether two distinct comparators impose the same order.