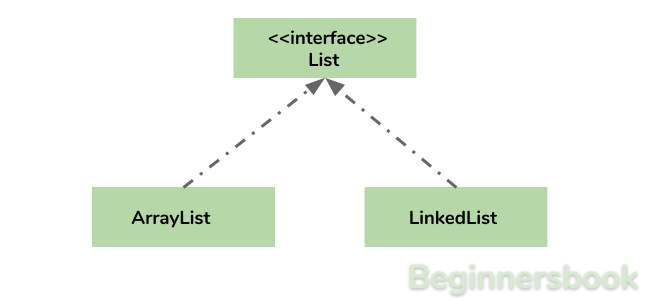
**Java Collections – List**

-based index.

A List is an ordered Collection (sometimes called a sequence). Lists may contain duplicate elements. Elements can be inserted or accessed by their position in the list, using a index.

**Arraylist** class implements List interface and it is based on an Array data structure. It is widely used because of the functionality and flexibility it offers. Most of the developers **choose Arraylist over Array** as it’s a very good alternative of traditional java arrays. ArrayList is a resizable-array implementation of the List interface. It implements all optional list operations, and permits all elements, including null.

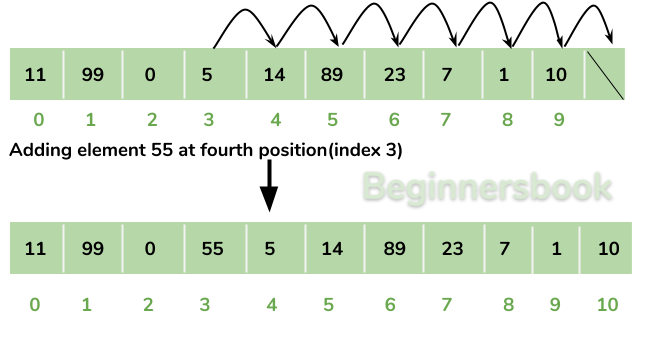


**Why ArrayList is better than Array?**

The limitation with array is that it has a fixed length so if it is full you cannot add any more elements to it, likewise if there are number of elements gets removed from it the memory consumption would be the same as it doesn’t shrink.

On the other ArrayList can dynamically grow and shrink after addition and removal of elements. Apart from these benefits ArrayList class enables us to use predefined methods of it which makes our task easy.

**Adding Element in ArrayList at specified position:**



**Removing Element from ArrayList:**



**How to create an ArrayList?**

This statement creates an ArrayList with the name alist with type “String”. The type determines which type of elements the list will have. Since this list is of “String” type, the elements that are going to be added to this list will be of type “String”.

ArrayList<String> alist=new ArrayList<String>();

Similarly we can create ArrayList that accepts int elements.

ArrayList<Integer> list=new ArrayList<Integer>();

**How to add elements to an ArrayList?**

We add elements to an ArrayList by using add() method, this method has couple of variations, which we can use based on the requirement. For example: If we want to add the element at the end of the List then simply do it like this:

alist.add("Steve"); //This will add "Steve" at the end of List

To add the element at the specified location in ArrayList, we can specify the index in the add method like this:

alist.add(3, "Steve"); //This will add "Steve" at the fourth position

Lets write the complete code:

import java.util.\*;

class JavaExample{

public static void main(String args[]){

ArrayList<String> alist=new ArrayList<String>();

alist.add("Steve");

alist.add("Tim");

alist.add("Lucy");

alist.add("Pat");

alist.add("Angela");

alist.add("Tom");

//displaying elements

System.out.println(alist);

//Adding "Steve" at the fourth position

alist.add(3, "Steve");

//displaying elements

System.out.println(alist);

}

}

**Output:**

[Steve, Tim, Lucy, Pat, Angela, Tom]

[Steve, Tim, Lucy, Steve, Pat, Angela, Tom]

**Note:** Since the index starts with 0, index 3 would represent fourth position not 3.

**How to remove elements from ArrayList?**

We use remove() method to remove elements from an ArrayList, Same as add() method, this method also has few variations.

For example:

import java.util.\*;

class JavaExample{

public static void main(String args[]){

ArrayList<String> alist=new ArrayList<String>();

alist.add("Steve");

alist.add("Tim");

alist.add("Lucy");

alist.add("Pat");

alist.add("Angela");

alist.add("Tom");

//displaying elements

System.out.println(alist);

//Removing "Steve" and "Angela"

alist.remove("Steve");

alist.remove("Angela");

//displaying elements

System.out.println(alist);

//Removing 3rd element

alist.remove(2);

//displaying elements

System.out.println(alist);

}

}

**Output:**

[Steve, Tim, Lucy, Pat, Angela, Tom]

[Tim, Lucy, Pat, Tom]

[Tim, Lucy, Tom]

**Iterating ArrayList**

In the above examples, we have displayed the ArrayList elements just by referring the ArrayList instance, which is definitely not the right way to displays the elements. The correct way of displaying the elements is by using an advanced for loop like this.

import java.util.\*;

class JavaExample{

public static void main(String args[]){

ArrayList<String> alist=new ArrayList<String>();

alist.add("Gregor Clegane");

alist.add("Khal Drogo");

alist.add("Cersei Lannister");

alist.add("Sandor Clegane");

alist.add("Tyrion Lannister");

//iterating ArrayList

for(String str:alist)

System.out.println(str);

}

}

**Output:**

Gregor Clegane

Khal Drogo

Cersei Lannister

Sandor Clegane

Tyrion Lannister

**ArrayList Example in Java**

This example demonstrates how to create, initialize, add and remove elements from ArrayList. In this example we have an ArrayList of type “String”. We have added 5 String element in the ArrayList using the method add(String E), this method adds the element at the end of the ArrayList.

We are then adding two more elements in the ArrayList using method add(int index, String E), this method adds the specified element at the specified index, index 0 indicates first position and 1 indicates second position.

We are then removing the elements “Chaitanya” and “Harry” from the ArrayList and then we are removing the second element of the ArrayList using method remove(int index). Since we have specified the index as 1 (remove(1)), it would remove the second element.

import java.util.\*;

public class JavaExample {

public static void main(String args[]) {

/\* Creating ArrayList of type "String" which means

\* we can only add "String" elements

\*/

ArrayList<String> obj = new ArrayList<String>();

/\*This is how we add elements to an ArrayList\*/

obj.add("Ajeet");

obj.add("Harry");

obj.add("Chaitanya");

obj.add("Steve");

obj.add("Anuj");

// Displaying elements

System.out.println("Original ArrayList:");

for(String str:obj)

System.out.println(str);

/\* Add element at the given index

\* obj.add(0, "Rahul") - Adding element "Rahul" at first position

\* obj.add(1, "Justin") - Adding element "Justin" at second position

\*/

obj.add(0, "Rahul");

obj.add(1, "Justin");

// Displaying elements

System.out.println("ArrayList after add operation:");

for(String str:obj)

System.out.println(str);

//Remove elements from ArrayList like this

obj.remove("Chaitanya"); //Removes "Chaitanya" from ArrayList

obj.remove("Harry"); //Removes "Harry" from ArrayList

// Displaying elements

System.out.println("ArrayList after remove operation:");

for(String str:obj)

System.out.println(str);

//Remove element from the specified index

obj.remove(1); //Removes Second element from the List

// Displaying elements

System.out.println("Final ArrayList:");

for(String str:obj)

System.out.println(str);

}

}

**Output:**

Original ArrayList:

Ajeet

Harry

Chaitanya

Steve

Anuj

ArrayList after add operation:

Rahul

Justin

Ajeet

Harry

Chaitanya

Steve

Anuj

ArrayList after remove operation:

Rahul

Justin

Ajeet

Steve

Anuj

Final ArrayList:

Rahul

Ajeet

Steve

Anuj

**Methods of ArrayList class**

In the above example we have used methods such as add() and remove(). However there are number of methods available which can be used directly using object of ArrayList class. Let’s discuss few important methods of ArrayList class.

1)  **add(Object o)**: This method adds an object o to the arraylist.

obj.add("hello");

This statement would add a string hello in the arraylist at last position.

2) **add(int index, Object o)**: It adds the object o to the array list at the given index.

obj.add(2, **"bye");**

It will add the string bye to the 2nd index (3rd position as the array list starts with index 0) of array list.

3) **remove(Object o)**: Removes the object o from the ArrayList.

obj.remove("Chaitanya");

This statement will remove the string “Chaitanya” from the ArrayList.

4) **remove(int index)**: Removes element from a given index.

obj.remove(3);

It would remove the element of index 3 (4th element of the list – List starts with o).

5) **set(int index, Object o)**: Used for updating an element. It replaces the element present at the specified index with the object o.

obj.set(2, "Tom");

It would replace the 3rd element (index =2 is 3rd element) with the value Tom.

6)**int indexOf(Object o)**: Gives the index of the object o. If the element is not found in the list then this method returns the value -1.

int pos = obj.indexOf("Tom");

This would give the index (position) of the string Tom in the list.

7) **Object get(int index)**: It returns the object of list which is present at the specified index.

String str= obj.get(2);

Function get would return the string stored at 3rd position (index 2) and would be assigned to the string “str”. We have stored the returned value in string variable because in our example we have defined the ArrayList is of String type. If you are having integer array list then the returned value should be stored in an integer variable.

8) **int size()**: It gives the size of the ArrayList – Number of elements of the list.

int numberofitems = obj.size();

9) **boolean contains(Object o)**: It checks whether the given object o is present in the array list if its there then it returns true else it returns false.

Boolean a=Obj.contains(“abc”)

It would return true if the string “Steve” is present in the list else we would get false.

10) **clear():** It is used for removing all the elements of the array list in one go. The below code will remove all the elements of ArrayList whose object is obj.

obj.clear();

**How to loop ArrayList in Java**

Earlier we shared ArrayList example and how to initialize ArrayList in Java. In this post we are sharing how to**iterate (loop) ArrayList in Java**.

There are four ways to loop ArrayList:

1. For Loop
2. Advanced for loop
3. While Loop
4. Iterator

Lets have a look at the below example

import java.util.\*;

public class LoopExample {

public static void main(String[] args) {

ArrayList<Integer> arrlist = new ArrayList<Integer>();

arrlist.add(14);

arrlist.add(7);

arrlist.add(39);

arrlist.add(40);

/\* For Loop for iterating ArrayList \*/

System.out.println("For Loop");

for (int counter = 0; counter < arrlist.size(); counter++) {

System.out.println(arrlist.get(counter));

}

/\* Advanced For Loop\*/

System.out.println("Advanced For Loop");

for (Integer num : arrlist) {

System.out.println(num);

}

/\* While Loop for iterating ArrayList\*/

System.out.println("While Loop");

int count = 0;

while (arrlist.size() > count) {

System.out.println(arrlist.get(count));

count++;

}

/\*Looping Array List using Iterator\*/

System.out.println("Iterator");

Iterator iter = arrlist.iterator();

while (iter.hasNext()) {

System.out.println(iter.next());

}

}

}

**Output:**

**For Loop**

14

7

39

40

**Advanced For Loop**

14

7

39

40

**While Loop**

14

7

39

40

**Iterator**

14

7

39

40

**How to find length of ArrayList in Java**

By using  size() method of ArrayList class we can easily **determine the size of the ArrayList**. This method returns the number of elements of ArrayList.

public int size()

**Example:**

package beginnersbook.com;

import java.util.ArrayList;

public class Details

{

public static void main(String [] args)

{

ArrayList<Integer> al=new ArrayList<Integer>();

System.out.println("Initial size: "+al.size());

al.add(1);

al.add(13);

al.add(45);

al.add(44);

al.add(99);

System.out.println("Size after few additions: "+al.size());

al.remove(1);

al.remove(2);

System.out.println("Size after remove operations: "+al.size());

System.out.println("Final ArrayList: ");

for(int num: al){

System.out.println(num);

}

}

}

**Output:**

Initial size: 0

Size after few additions: 5

Size after remove operations: 3

Final ArrayList:

1

45

99

**SORTING**

**How to sort ArrayList in Java**

**Example 1: Sorting of ArrayList<String>**

Here we are sorting the ArrayList of String type. We are doing it by simply calling the Collections.sort(arraylist) method. The output List will be sorted alphabetically.

import java.util.\*;

public class Details {

public static void main(String args[]){

ArrayList<String> listofcountries = new ArrayList<String>();

listofcountries.add("India");

listofcountries.add("US");

listofcountries.add("China");

listofcountries.add("Denmark");

/\*Unsorted List\*/

System.out.println("Before Sorting:");

for(String counter: listofcountries){

System.out.println(counter);

}

/\* Sort statement\*/

Collections.sort(listofcountries);

/\* Sorted List\*/

System.out.println("After Sorting:");

for(String counter: listofcountries){

System.out.println(counter);

}

}

}

**Output:**

Before Sorting:

India

US

China

Denmark

After Sorting:

China

Denmark

India

US

**Example 2: Sorting of ArrayList<Integer>**

The same Collections.sort() method can be used for sorting the Integer ArrayList as well.

import java.util.\*;

public class ArrayListOfInteger {

public static void main(String args[]){

ArrayList<Integer> arraylist = new ArrayList<Integer>();

arraylist.add(11);

arraylist.add(2);

arraylist.add(7);

arraylist.add(3);

/\* ArrayList before the sorting\*/

System.out.println("Before Sorting:");

for(int counter: arraylist){

System.out.println(counter);

}

/\* Sorting of arraylist using Collections.sort\*/

Collections.sort(arraylist);

/\* ArrayList after sorting\*/

System.out.println("After Sorting:");

for(int counter: arraylist){

System.out.println(counter);

}

}

}

**Output:**

Before Sorting:

11

2

7

3

After Sorting:

2

3

7

11

**How to sort ArrayList in descending order in Java**

**Example: Sorting in Descending order**

We are using Collections.reverseOrder() method along with Collections.sort() in order to sort the list in decreasing order. In the below example we have used the following statement for sorting in reverse order.  
Collections.sort(arraylist, Collections.reverseOrder());

However the reverse order sorting can also be done as following – This way the list will be sorted in ascending order first and then it will be reversed.  
Collections.sort(list);  
Collections.reverse(list);

**Complete example:**

import java.util.\*;

public class Details {

public static void main(String args[]){

ArrayList<String> arraylist = new ArrayList<String>();

arraylist.add("AA");

arraylist.add("ZZ");

arraylist.add("CC");

arraylist.add("FF");

/\*Unsorted List: ArrayList content before sorting\*/

System.out.println("Before Sorting:");

for(String str: arraylist){

System.out.println(str);

}

/\* Sorting in decreasing order\*/

Collections.sort(arraylist, Collections.reverseOrder());

/\* Sorted List in reverse order\*/

System.out.println("ArrayList in descending order:");

for(String str: arraylist){

System.out.println(str);

}

}

}

**Output:**

Before Sorting:

AA

ZZ

CC

FF

ArrayList in descending order:

ZZ

FF

CC

AA

**LinkedList in Java with Example**

LinkedList is a doubly-linked list implementation of the List and Deque interfaces. LinkedList allows for constant-time insertions or removals using iterators, but only sequential access of elements. In other words, LinkedList can be searched forward and backward but the time it takes to traverse the list is directly proportional to the size of the list.

**Example of LinkedList in Java**

import java.util.\*;

public class LinkedListExample {

public static void main(String args[]) {

/\* Linked List Declaration \*/

LinkedList<String> linkedlist = new LinkedList<String>();

/\*add(String Element) is used for adding

\* the elements to the linked list\*/

linkedlist.add("Item1");

linkedlist.add("Item5");

linkedlist.add("Item3");

linkedlist.add("Item6");

linkedlist.add("Item2");

/\*Display Linked List Content\*/

System.out.println("Linked List Content: " +linkedlist);

/\*Add First and Last Element\*/

linkedlist.addFirst("First Item");

linkedlist.addLast("Last Item");

System.out.println("LinkedList Content after addition: " +linkedlist);

/\*This is how to get and set Values\*/

Object firstvar = linkedlist.get(0);

System.out.println("First element: " +firstvar);

linkedlist.set(0, "Changed first item");

Object firstvar2 = linkedlist.get(0);

System.out.println("First element after update by set method: " +firstvar2);

/\*Remove first and last element\*/

linkedlist.removeFirst();

linkedlist.removeLast();

System.out.println("LinkedList after deletion of first and last element: " +linkedlist);

/\* Add to a Position and remove from a position\*/

linkedlist.add(0, "Newly added item");

linkedlist.remove(2);

System.out.println("Final Content: " +linkedlist);

}

}

**Output:**

Linked List Content: [Item1, Item5, Item3, Item6, Item2]

LinkedList Content after addition: [First Item, Item1, Item5, Item3, Item6, Item2, Last Item]

First element: First Item

First element after update by set method: Changed first item

LinkedList after deletion of first and last element: [Item1, Item5, Item3, Item6, Item2]

Final Content: [Newly added item, Item1, Item3, Item6, Item2]

**Methods of LinkedList class:**

For all the examples in the below methods, consider llistobj as a reference for LinkedList<String>.

LinkedList<String> llistobj  = new LinkedList<String>();

1) **boolean add(Object item)**: It adds the item at the end of the list.

llistobj.add("Hello");

It would add the string “Hello” at the end of the linked list.

2) **void add(int index, Object item)**: It adds an item at the given index of the the list.

llistobj.add(2, "bye");

This will add the string “bye” at the 3rd position( 2 index is 3rd position as index starts with 0).

3) **boolean addAll(Collection c)**: It adds all the elements of the specified collection c to the list. It throws NullPointerException if the specified collection is null. Consider the below example –

LinkedList<String> llistobj = new LinkedList<String>();

ArrayList<String> arraylist= new ArrayList<String>();

arraylist.add("String1");

arraylist.add("String2");

llistobj.addAll(arraylist);

This piece of code would add all the elements of ArrayList to the LinkedList.

4) **boolean addAll(int index, Collection c)**: It adds all the elements of collection c to the list starting from a give index in the list. It throws NullPointerException if the collection c is null and IndexOutOfBoundsException when the specified index is out of the range.

llistobj.add(5, arraylist);

It would add all the elements of the ArrayList to the LinkedList starting from position 6 (index 5).

5) **void addFirst(Object item)**: It adds the item (or element) at the first position in the list.

llistobj.addFirst("text");

It would add the string “text” at the beginning of the list.

6) **void addLast(Object item)**: It inserts the specified item at the end of the list.

llistobj.addLast("Chaitanya");

This statement will add a string “Chaitanya” at the end position of the linked list.

7) **void clear()**: It removes all the elements of a list.

llistobj.clear();

8) **Object clone()**: It returns the copy of the list.

For e.g. My linkedList has four items: text1, text2, text3 and text4.

Object str= llistobj.clone();

System.out.println(str);

Output: The output of above code would be:

[text1, text2, text3, text4]

9) **boolean contains(Object item)**: It checks whether the given item is present in the list or not. If the item is present then it returns true else false.

boolean var = llistobj.contains("TestString");

It will check whether the string “TestString” exist in the list or not.

10) **Object get(int index)**: It returns the item of the specified index from the list.

Object var = llistobj.get(2);

It will fetch the 3rd item from the list.

11) **Object getFirst()**: It fetches the first item from the list.

Object var = llistobj.getFirst();

12) **Object getLast()**: It fetches the last item from the list.

Object var= llistobj.getLast();

13) **int indexOf(Object item)**: It returns the index of the specified item.

llistobj.indexOf("bye");

14) **int lastIndexOf(Object item)**: It returns the index of last occurrence of the specified element.

int pos = llistobj.lastIndexOf("hello);

integer variable pos will be having the index of last occurrence of string “hello”.

15) **Object poll()**: It returns and removes the first item of the list.

Object o = llistobj.poll();

16) **Object pollFirst()**: same as poll() method. Removes the first item of the list.

Object o = llistobj.pollFirst();

17) **Object pollLast()**: It returns and removes the last element of the list.

Object o = llistobj.pollLast();

18) **Object remove()**: It removes the first element of the list.

llistobj.remove();

19) **Object remove(int index)**: It removes the item from the list which is present at the specified index.

llistobj.remove(4);

It will remove the 5th element from the list.

20) **Object remove(Object obj)**: It removes the specified object from the list.

llistobj.remove("Test Item");

21) **Object removeFirst()**: It removes the first item from the list.

llistobj.removeFirst();

22) **Object removeLast()**: It removes the last item of the list.

llistobj.removeLast();

23) **Object removeFirstOccurrence(Object item)**: It removes the first occurrence of the specified item.

llistobj.removeFirstOccurrence("text");

It will remove the first occurrence of the string “text” from the list.

24) **Object removeLastOccurrence(Object item)**: It removes the last occurrence of the given element.

llistobj.removeLastOccurrence("String1);

It will remove the last occurrence of string “String1”.

25) **Object set(int index, Object item)**: It updates the item of specified index with the give value.

llistobj.set(2, "Test");

It will update the 3rd element with the string “Test”.

26)**int size()**: It returns the number of elements of the list.

llistobj.size();

**How to loop LinkedList in Java**

There are four ways in which a LinkedList can be iterated –

1. For loop
2. Advanced For loop
3. Iterator
4. While Loop

**Example:**

In this example we have a LinkedList of String Type and we are looping through it using all the four mentioned methods.

import java.util.\*;

public class LinkedListExample {

public static void main(String args[]) {

/\*LinkedList declaration\*/

LinkedList<String> linkedlist=new LinkedList<String>();

linkedlist.add("Apple");

linkedlist.add("Orange");

linkedlist.add("Mango");

/\*for loop\*/

System.out.println("\*\*For loop\*\*");

for(int num=0; num<linkedlist.size(); num++)

{

System.out.println(linkedlist.get(num));

}

/\*Advanced for loop\*/

System.out.println("\*\*Advanced For loop\*\*");

for(String str: linkedlist)

{

System.out.println(str);

}

/\*Using Iterator\*/

System.out.println("\*\*Iterator\*\*");

Iterator i = linkedlist.iterator();

while (i.hasNext()) {

System.out.println(i.next());

}

/\* Using While Loop\*/

System.out.println("\*\*While Loop\*\*");

int num = 0;

while (linkedlist.size() > num) {

System.out.println(linkedlist.get(num));

num++;

}

}

}

**Output:**

\*\*For loop\*\*

Apple

Orange

Mango

\*\*Advanced For loop\*\*

Apple

Orange

Mango

\*\*Iterator\*\*

Apple

Orange

Mango

\*\*While Loop\*\*

Apple

Orange

Mango

**Vector in Java**

Vector implements List Interface. Like ArrayList it also maintains insertion order but it is rarely used in non-thread environment as it is synchronized and due to which it gives poor performance in searching, adding, delete and update of its elements.

**Three ways to create vector class object:**

**Method 1:**

Vector vec = new Vector();

It creates an empty Vector with the default initial capacity of 10. It means the Vector will be re-sized when the 11th elements needs to be inserted into the Vector. Note: By default vector doubles its size. i.e. In this case the Vector size would remain 10 till 10 insertions and once we try to insert the 11th element It would become 20 (double of default capacity 10).

**Method 2:**  
Syntax: Vector object= new Vector(int initialCapacity)

Vector vec = new Vector(3);

It will create a Vector of initial capacity of 3.

**Method 3:**  
Syntax:

Vector object= new vector(int initialcapacity, capacityIncrement)

Example:

Vector vec= new Vector(4, 6)

Here we have provided two arguments. The initial capacity is 4 and capacityIncrement is 6. It means upon insertion of 5th element the size would be 10 (4+6) and on 11th insertion it would be 16(10+6).

**Complete Example of Vector in Java:**

import java.util.\*;

public class VectorExample {

public static void main(String args[]) {

/\* Vector of initial capacity(size) of 2 \*/

Vector<String> vec = new Vector<String>(2);

/\* Adding elements to a vector\*/

vec.addElement("Apple");

vec.addElement("Orange");

vec.addElement("Mango");

vec.addElement("Fig");

/\* check size and capacityIncrement\*/

System.out.println("Size is: "+vec.size());

System.out.println("Default capacity increment is: "+vec.capacity());

vec.addElement("fruit1");

vec.addElement("fruit2");

vec.addElement("fruit3");

/\*size and capacityIncrement after two insertions\*/

System.out.println("Size after addition: "+vec.size());

System.out.println("Capacity after increment is: "+vec.capacity());

/\*Display Vector elements\*/

Enumeration en = vec.elements();

System.out.println("\nElements are:");

while(en.hasMoreElements())

System.out.print(en.nextElement() + " ");

}

}

**Output:**

Size is: 4

Default capacity increment is: 4

Size after addition: 7

Capacity after increment is: 8

Elements are:

Apple Orange Mango Fig fruit1 fruit2 fruit3

**Commonly used methods of Vector Class:**

1. **void addElement(Object element):** It inserts the element at the end of the Vector.
2. **int capacity():** This method returns the current capacity of the vector.
3. **int size():** It returns the current size of the vector.
4. **void setSize(int size):** It changes the existing size with the specified size.
5. **boolean contains(Object element):** This method checks whether the specified element is present in the Vector. If the element is been found it returns true else false.
6. **boolean containsAll(Collection c):** It returns true if all the elements of collection c are present in the Vector.
7. **Object elementAt(int index):** It returns the element present at the specified location in Vector.
8. **Object firstElement():** It is used for getting the first element of the vector.
9. **Object lastElement():** Returns the last element of the array.
10. **Object get(int index):** Returns the element at the specified index.
11. **boolean isEmpty():** This method returns true if Vector doesn’t have any element.
12. **boolean removeElement(Object element):** Removes the specifed element from vector.
13. **boolean removeAll(Collection c):** It Removes all those elements from vector which are present in the Collection c.
14. **void setElementAt(Object element, int index):** It updates the element of specifed index with the given element.