ArrayList Class

ArrayList class is an improvement over arrays. Arrays are fixed size data structures. This means once created we cannot change their length again, which in turn requires the size to be calculated upfront. ArrayList on the other hand are dynamic in size. This enables us to add or remove elements from the ArrayList easily and it grows and shrinks in size accordingly.

We use arrays where we know the size required upfront and no changes will be done in size later. ArrayList due to its dynamic size property are mostly used where a variable length data store is needed.

Use ArrayList in following situations:

- Size cannot be determined upfront
- Calculating size upfront take a lot of effort
- Size can change – Elements can be added or removed after

Syntax:

```
ArrayList<ClassName> list_name = new ArrayList<ClassName>();
```

ArrayList can only hold objects. So if you want to store any primitive data type then you have to use its corresponding wrapper class. Java automatically handles the conversion.

Class Name for different data types:

- **Integer** class for int
- **String** class for String
- **Character** class for char

Example 1: Creating Simple ArrayList

```
1   ArrayList<Integer> arrList1 = new ArrayList<Integer>();
1   ArrayList<String> arrList2 = new ArrayList<String>();
```

Basic Methods: add() and size()

Some basic methods:

- add(element) : appends given "element" at the end of the given ArrayList
- size() : returns the size of ArrayList (total elements present)

Example 2: Using simple methods

```
1   ArrayList<String> countries = new ArrayList<String>();
2
3   countries.add("India");                    //add element at the end of ArrayList
4   countries.add("China");
5
6   int len = countries.size();                //get current size of our ArrayList
7
```

```
8  System.out.println(len);
9  countries.add("Pakistan");
10 len = countries.size();                    //size automatically increases
11 System.out.println(len);
12
13 System.out.println(countries);             //printing our ArrayList
```

Output:

```
2
3
[India, China, Pakistan]
```

Problem 1	**Give output of following code.**

```
1  ArrayList<String> fruits = new ArrayList<String>();
2  fruits.add("Banana");
3  fruits.add("Apple");
4  int len = fruits.size();
5  System.out.println(len);
6  fruits.add("Orange");
7  len = fruits.size();
8  System.out.println(len);
9  System.out.println(fruits);
```

Getting and Setting elements at specific index.

Accessing and modifying an element at specific index is very easy.

- set(index, new_element) : changes the "element" at specific "index" of ArrayList
- get(index) : returns the element at given "index" of ArrayList. Index ranges from 0 to size() – 1

Example 3.1: Demonstrating Operations – Getting and Setting elements at specific index

```
1  //Assume ArrayList : marks<Integer> contains [70, 65, 33, 94]
2
3  int len = marks.size();
4  System.out.println(len);
5
6  System.out.println(marks);
7                                    //getting Integer at 1st index (2nd element).
8  int m1 = marks.get(1);            //Notice use of int. Java automatically
9                                    //converts Integer to int.
10 System.out.println(m1);
11 marks.set(2, 40);                 //Setting 40 at 2nd index.
12
13 System.out.println(marks);
```

Output:

```
4
[70, 65, 33, 94]
65
[70, 65, 40, 94]
```

Explanation:

ArrayList follows the same index structure like arrays. Indexes start from 0 and run till size() -1. So first index value means second element. If you try to access or modify an index value which is outside the range then you'll get IndexOutOfBoundsException. Following example will make this clear.

Example 3.2: Example to Demonstrate IndexOutOfBoundsException

```
1  // Let's create an ArrayList of characters. We'll use Character class.
2
3  ArrayList<Character> arrList = new ArrayList<Character>();
4
5  arrList.add('T');                          //adding our first Character
6  arrList.add('R');
7
8  int len = arrList.size();
9  System.out.println(len);
10
11 arrList.set(-1, 40);                       //Setting 40 at index less than 0
12 arrList.get(4);                            //get element at 4th index
13
14 System.out.println(arrList);
```

Output:

```
IndexOutOfBoundsException
```

Explanation:

Both the lines at 11 and 12 in above code are trying to access or modify the index values outside the valid range of 0 – (size() – 1). Just like arrays we get an IndexOutOfBoundsException here.

Problem 2	**Give output of following code.**

```
1  ArrayList<Integer> arrList = new ArrayList<Integer>();
2  arrList.add(30);
3  arrList.add(40);
4  arrList.add(10);
5  arrList.add(50);
6  int len = arrList.size();
7  System.out.println(len);
8  System.out.println(arrList);
9  int num = arrList.get(0);
10 System.out.println(num);
11 arrList.set(1, num + 5);
12 arrList.set(2, num + 10);
13 arrList.set(3, num + 15);
14 System.out.println(arrList);
```

Problem 3	Give output of following code.

```
1  ArrayList<String> whitelist = new ArrayList<String>();
2  whitelist.add("Rohan");
3  whitelist.add("Neha");
4  int len = whitelist.size();
5  System.out.println(len);
6  whitelist.set(0, "Mohan");
7  whitelist.set(2, "Shivani");
8  System.out.println(whitelist);
```

Traversing an ArrayList: Index based approach

- We can use standard for loop to traverse an ArrayList.
- This method is preferred if you want to keep track of index. It helps to change the value if needed. Also it enables us to start or stop our loop at specific index.

Example 4: Index based ArrayList traversal

```
1  //Assume ArrayList : numbers<Integer> contains [12, 13, 9, 10]
2
3  int len = numbers.size();
4
5  for(int i = 0; i < len; i++)
6  {
7     int num = numbers.get(i);
8     System.out.println("Number at index " + i + " is " + num);
9
10    num = num + 5;
11    numbers.set(i, num);                    //adding 5 to elements at same index
12 }
13 System.out.println(numbers);
```

Output:

```
Number at index 0 is 12
Number at index 1 is 13
Number at index 2 is 9
Number at index 3 is 10
[17, 18, 14, 15]
```

Problem 4	Give output of following code.

```
1  ArrayList<Integer> arrList = new ArrayList<Integer>();
2  int arr[] = {3, 9, 6, 7};
3
4  int len = arr.length;
5
6  for(int i = 0; i < len; i++)
7  {
8     int num = arr[i];
9     arrList.add(num);
10    num = num * 2;
```

```
11     arrList.add(num);
12  }
13
14  len = arrList.size();
15  System.out.println("Size: " + len);
16
17  for(int i = 0; i < len; i++)
18  {
19     int ele = arrList.get(i);
20     System.out.println("#" + i + " : " + ele);
21  }
```

Traversing an ArrayList: Using Iterator

- Iterator skips the indexes.
- Preferred if you don't need indexes. But, still want to remove elements.
- It also provides ability to move backward and forward using next() and previous();
- Important methods of Iterator:
    - next() : moves forward and returns the next element
    - hasNext(): returns false if we are at the last element of given ArrayList

Example 5: ArrayList traversal : Iterator based approach

```
1  //Assume ArrayList : numbers<Integer> contains [12, 13, 9, 10]
2
3  Iterator itr = numbers.iterator();         //getting ArrayList iterator
4
5  while(itr.hasNext())                       //checking if we are at end
6  {
7     int n = itr.next();                     //getting next element & moving forward
8     System.out.println(n);
9  }
```

Output:

```
12
13
9
10
```

Explanation:

The iterator() method returns the object of Iterator class which points at the start of our ArrayList. The hasNext() method returns true if there is next element in the iteration. The next() method, moving forward, gives us the next element.

Problem 5	**Give output of following code.**

```
1  ArrayList<Character> alphabets = new ArrayList<Character>();
2  alphabets.add('A');
3  alphabets.add('C');
4  alphabets.add('D');
```

```
5  alphabets.add('E');
6  alphabets.set(0, 'B');
7
8  Iterator itr = alphabets.iterator();
9
10 while(itr.hasNext())
11 {
12    char ch = itr.next();
13    System.out.println(ch);
14 }
```

Traversing with For-each loop

- For-each loop skips the indexes.
- Preferred if you only want access to values. Loop will automatically start from very first element and keep going on till we reach the end. No need to calculate or maintain index. Howeve, changing element is not possible.
- Can work with both arrays as well as ArrayList.

Example 6.1: For-each loop to traverse Arrays

```
1  int arr = {12, 9, 16, 13}
2  for(int item : arr)                //Read: for every "item" in "arr"
3  {                                  //variable name "item" can be changed
4     System.out.println(item);
5  }
```

Output:

```
12
9
16
13
```

Example 6.2: For-each loop to traverse ArrayList

```
1  //Assume ArrayList : names<String> contains [Salman, Anushka, Randeep]
2  for(String str : names)                          //for every element str in names
3  {
4     System.out.println("Star Cast: " + str);
5  }
```

Output:

```
Star Cast: Salman
Star Cast: Anushka
Star Cast: Randeep
```

Explanation:

The for-each loop is very simple in its operation. You basically define a new variable that will hold one element from the array/ArrayList at a time. When the loop starts it holds the very first element. At each iteration the this variable will receive next value. The loop automatically terminates when we have traversed all the elements. Inside loop we have no idea of our position (index).

Problem 6	Give output of following code.

```
1  //Assume ArrayList : numbers<Integer> contains [2, 7, 4, 1, 5]
2  for(int num : numbers)
3  {
4     if(num < 5)
5     {
6           System.out.println(num);
7     }
8  }
```

Collection Framework

Collection Framework reduces the programming effort by providing useful data structure and algorithms. You don't need to create these common classes/algorithms from scratch, instead you can concentrate on other important things in your program.

Collection Framework provides:

- Useful classes like ArrayList, HashSet, HashMap, etc.,
- Set of interfaces like List, Set, etc.,
- Common algorithms like sorting and searching

ArrayList is part of Java Collection Framework. Some of the key points about ArrayList are:

- ArrayList class implements the List interface. List interface in Collections provides basic methods like add(), set(), etc.
- Unlike arrays that are fixed in size ArrayList can change their size. Hence, we use ArrayList wherever we need a variable length structure that can grow and shrink according to our needs.

Key Differences between Array and ArrayList:

Parameters	Array	ArrayList
Resizable	No. Arrays are fixed size structure. We cannot change the length after creation of array object after creation.	Yes. ArraList are dynamic in size. We can add new elements to ArraList. It grows accordingly.
Primitives	Arrays can hold both primitive data types (like int, char, double) as well as objects.	ArrayList can only store objects. However, Java automatically handles the conversion of primitive data type to its corresponding class (called wrapper classes).
Iterating/ Traversal	Only through for and for-each loops.	For, for-each and iterators can be used for traversal.

Problem 7 Num10to20	Create an ArrayList that contains numbers from 10 – 20. Use for loop to add numbers into this ArrayList. Print the ArrayList after adding numbers.	
Test Cases	Input	Output
	-	[10,11,12,13,14,15,16,17,18,19,20]

Problem 8 AllEvens	Complete the function given below. The function takes an integer array as input. Add all the even elements from this array to a new ArrayList. Print this ArrayList.	

```
public void allEvens(int[] arr)
{
   //create and print an ArrayList that contains only
   //the even numbers from input array
}
```

Test Cases	Input	Output
	allEvens({1,2,3,4,5,6})	[2,4,6]

Problem 9 ModifyTwice	Complete the function given below. The function takes an Integer ArrayList as input. Multiply every element in this ArrayList by 2. Print the modified ArrayList.	

```
public void modifyTwice(ArrayList<Integer> arrList)
{
   //multiply every element in input ArrayList by 2
   //print the modified ArrayList
}
```

Test Cases	Input	Output
	modifyTwice({1,2,3,4,5,6})	[2,4,6,8,10,12]