CSCI 2120:
Software Design & Development II

UNIT 2: Collections Framework & Generics
ArrayList class

ArrayList in Java | ArrayList Methods, Example

ArrayList in Java is a resizable array that can grow or shrink in the memory whenever needed. It is dynamically created with an initial capacity.

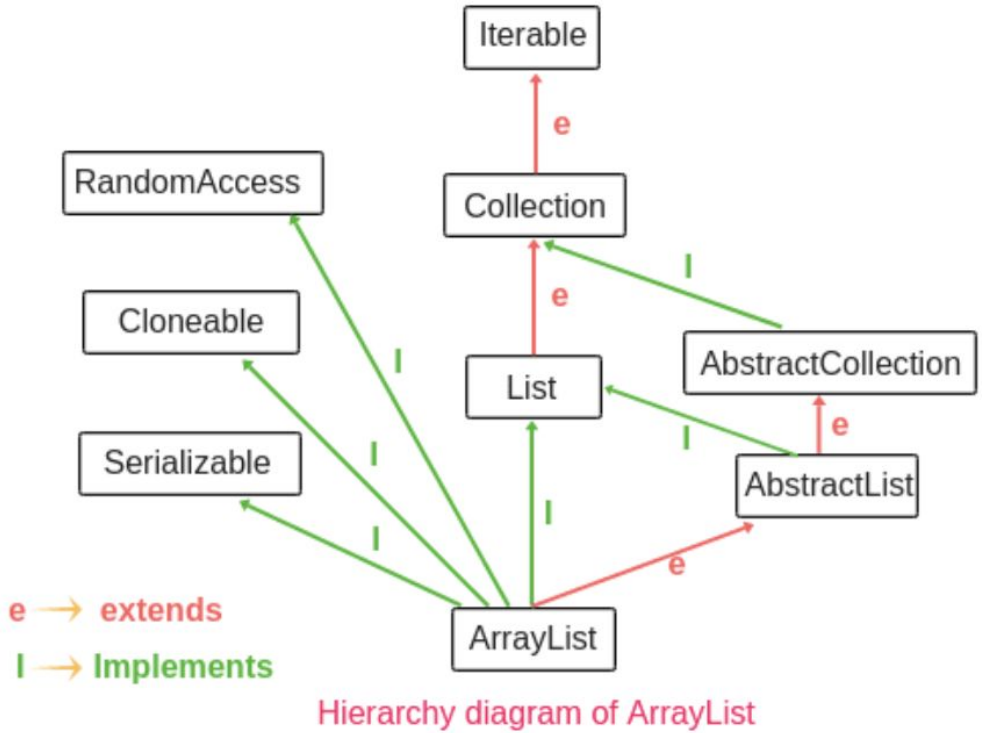
It means that if the initial capacity of the array is exceeded, a new array with larger capacity is created automatically and all the elements from the current array are copied to the new array.

Elements in ArrayList are placed according to the zero-based index. That is the first element will be placed at 0 index and the last element at index (n-1) where n is the size of ArrayList.

Java ArrayList uses a dynamic array internally for storing the group of elements or data.

The capacity of ArrayList does not shrink automatically. When elements are removed from the list, the size of array list can be shrunk automatically but not capacity.

Hierarchy Diagram of ArrayList Class in Java



Hierarchy diagram of ArrayList

Hierarchy Diagram of ArrayList Class in Java

As shown in the above hierarchy diagram, ArrayList class implements the List interface and extends AbstractList (Abstract class) which implements List interface.

Java ArrayList class also implements 3 marker interfaces: Random Access, Cloneable, and Serializable.

A marker interface is an interface that does not have any methods or any member variables. It is also called an empty interface because of no field or methods.

Random Access Interface

1. RandomAccess Interface is a marker interface that does not define any method or member. It is introduced in Java 1.4 version for optimizing the list performance.

2. RandomAccess interface is present in java.util package.

3. ArrayList class implements a random access interface so that we can access any random element at the same speed. For example, suppose there is a group of one crore objects in the array list. Assume that the first element is x, 10th element is y, and 1st crore element is z.

Now assume that first element x can be accessed within only 1 sec. Due to the implementation of random access interface, the 10th element and 1st crore element can also be accessed within 1 sec.

Thus, Any random element we can access with the same or constant speed. Therefore, if our frequent operation is retrieval operation then ArrayList is the best choice.

Serializable Interface

1. A serializable interface is a marker interface that is used to send the group of objects over the network. It is present in the java.io package.

2. It helps in sending the data from one class to another class. Usually, we use collections to hold and transfer objects from one place to another place.

To provide support for this requirement, every collections class already implements Serializable and Cloneable.

Cloneable Interface

1. A cloneable interface is present in java.lang package.

2. It is used to create exactly duplicate objects. When the data or group of objects came from the network, the receiver will create duplicate objects.

The process of creating exactly duplicate objects is known as cloning. It is a very common requirement for collection classes.

Features of ArrayList in Java

1. Resizable-array: ArrayList is a resizable array or growable array that means the size of ArrayList can increase or decrease in size at runtime. Once ArrayList is created, we can add any number of elements.

2. Index-based structure: It uses an index-based structure in java.

3. Duplicate elements: Duplicate elements are allowed in the array list.

4. Null elements: Any number of null elements can be added to ArrayList.

5. Insertion order: It maintains the insertion order in Java. That is insertion order is preserved.

6. Heterogeneous objects: Heterogeneous objects are allowed everywhere except TreeSet and TreeMap. Heterogeneous means different elements.

7. Synchronized: ArrayList is not synchronized. That means multiple threads can use the same ArrayList objects simultaneously.

8. Random Access: ArrayList implements random access because it uses an index-based structure. Therefore, we can get, set, insert, and remove elements of the array list from any arbitrary position.

9. Performance: In ArrayList, manipulation is slow because if any element is removed from ArrayList, a lot of shifting takes place. For example, if an array list has 500 elements and we remove 50th elements then the 51st element will try to acquire that 50th position, and likewise all elements. Thus, it consumes a lot of time-shifting.

Java ArrayList Constructor

Java ArrayList class provides three constructors for creating an object of ArrayList. They are as:

- ArrayList()
- ArrayList(int initialCapacity)
- ArrayList(Collection c)

Creating an object of ArrayList class in Java is very simple. First, we will declare an ArrayList variable and call the ArrayList constructor to instantiate an object of ArrayList class and then assign it to the variable.

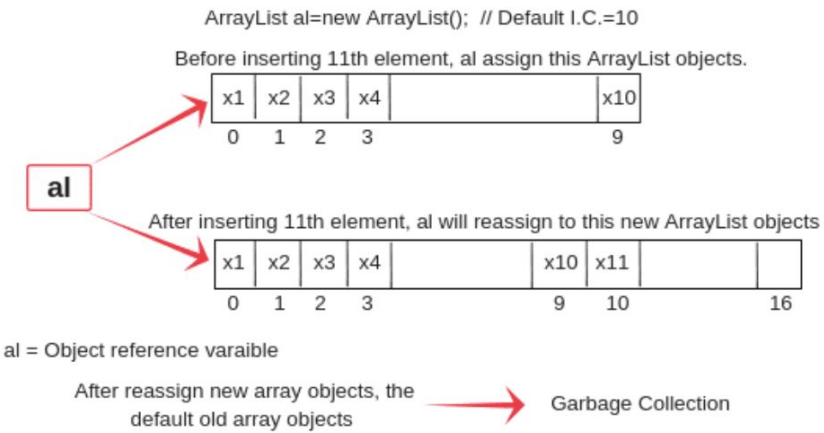
We can create an object of ArrayList class in java by using any one of three constructors. Let’s see one by one.

Java ArrayList Constructor

1. The syntax for creating an instance of ArrayList class is as:

```
ArrayList al = new ArrayList();
```

It creates an empty ArrayList with a default initial capacity of 10. In this array list, we can store only 10 elements as shown in the below diagram.



Java ArrayList Constructor

Suppose we insert the 11th element at 10th position into an array list, what will happen internally?

Once ArrayList is reached its maximum capacity, the ArrayList class automatically creates a new array with a larger capacity.

New capacity = (current capacity*3/2) + 1 = 10*3/2 + 1 = 16

After creating a new array list with a larger capacity, all the existing elements are copied into the new array list and then adds the new element into it. ArrayList class reassigns the reference to the new array objects as shown in the above figure.

The old default array list with the collection of objects is automatically gone into the garbage collection. Similarly, If we try to insert 17th element, new capacity= 16*3/2 + 1 = 25.

Java ArrayList Constructor

2. We can also initialize the capacity at the time of creating ArrayList object. The syntax for creating an instance of ArrayList class with initial capacity is as:

```
ArrayList al = new ArrayList(int initialCapacity);
```

It creates an empty ArrayList with initial capacity. If you know the initial capacity, you can directly use this method. Thus, we can improve the performance of the system by default.

For example, suppose our requirement is to add 500 elements in the array list, we will create ArrayList object like this:

```
ArrayList list2 = new ArrayList(500);
```

Java ArrayList Constructor

3. We can also create an object of ArrayList class by initializing a collection of elements into it. The syntax is as follow:

```
ArrayList al = new ArrayList(Collection c);
```

It creates an ArrayList object by initializing elements of collection c.

For example:

```
ArrayList list3 = new ArrayList(list1); // list1 is elements of collection.
```

Creating Generic ArrayList Object in Java

Java 1.5 version or later also provides us to specify the type of elements in the ArrayList object. For example, we can create a generic String ArrayList object like this:

```
ArrayList<String> al = new ArrayList<String>(); // It can be used to store only String type.

// The advantage of specifying a type is when we try to add another type, it will give compile-time error.
// or, Creating a Generic ArrayList object can also be done in separate lines like this:

ArrayList<String> arlist;
arlist = new ArrayList();
```

Note: We cannot use primitive data types as a type. For example, ArrayList<int> is illegal.

Java ArrayList Initialization

1. Using Arrays.asList: The syntax to initialize an ArrayList using asList() method is as follows:

```
ArrayList<Type> list = new ArrayList<Type>(Arrays.asList(Object o1, Object o2, .. so on));
```

For example:

```
ArrayList<String> ar = new ArrayList<String>(Arrays.asList("A", "B", "C"))
```

Java ArrayList Initialization

2: Using normal way: This is a popular way to initialize ArrayList in java program. The syntax to initialize array list is as:

```
ArrayList<Type> obj = new ArrayList<Type>();
  obj.add("Obj o1");
  obj.add("Obj o2");
  and so on.
```

ArrayList Methods in Java

1. boolean add(Object o): This method is used to add an element at the end of array list. For example, if you want to add an element at the end of the list, you simply call the add() method like this:

```
list.add("Shubh"); // This will add "Shubh" at the end of the list.
```

ArrayList Methods in Java

3. boolean addAll(int index, Collection c): This method is used to add a group of elements at a specified position in a list. For example:

```
list1.addAll(2, list2): // Adding all elements of list2 at position index 2 in list1
```

ArrayList Methods in Java

4. void add(int index, Object o): It is used to add an element at a particular position index in the list. For example:

```
list.add(3, "a"); // Adding element 'a' at position index 3 in list.
```

ArrayList Methods in Java

5. void addAll(int index, Object o): It is used to add a specific element at a particular position in the list. For example, suppose we want to a specific element “Shubh” at a position 2 in the list, we will call add(int index, Object o) method like this:

```
list.add(2,"Shubh"); // This will add "Shubh" at the second position.
```

Let’s take an example program based on the above ArrayList add() methods.

Example 1: Adding elements

```
import java.util.ArrayList;
public class ArrayListTester1 {
   public static void main(String[] args)  {
       // Create an object of the non-generic ArrayList.
       ArrayList al = new ArrayList(); // list 1 with default capacity 10.
       al.add("A");
       al.add("B");
       al.add(20);
       al.add("A");
       al.add(null);
       System.out.println(al);

       // Create an object of another non-generic ArrayList.
       ArrayList al1 = new ArrayList(); // List 2.
       al1.add("a");
       al1.add("b");
       al1.add("c");

       // Call addAll(Collection c) method using al to add all elements at the end of the list1.
       al.addAll(al1);
       System.out.println(al);

       // Call addAll(int index, Collection c) method using al1 to add all elements at specified position 2.
       al1.addAll(2, al);
       System.out.println(al1);
   }
}
```

Example 1: Adding elements

Output:

```
[A, B, 20, A, null]
[A, B, 20, A, null, a, b, c]
[a, b, A, B, 20, A, null, a, b, c, c]
```

ArrayList Methods in Java

5. boolean remove(Object o): It removes the first occurrence of the specified element from this list if it is present.

6. void remove(int index): This method removes the element from a particular position in the list. Look at the examples below.

```
list.remove("A");
```

```
list.remove(2); //  It will remove element from position 2.
```

ArrayList Methods in Java

7. void clear(): The clear() method is used to remove all elements from an array list.

ArrayList Methods in Java

8. void set(int index, Object o): The set() method replaces element at a particular position in the list with the specified element. For example, suppose we want to replace an element “A” at a position 2 with an element “a” in the list, we will have to call this method as:

```
list.set(2, "a");
```

Let’s take an example program where we will remove an element using remove() method from the list.

ArrayList Methods in Java

8. void set(int index, Object o): The set() method replaces element at a particular position in the list with the specified element. For example, suppose we want to replace an element “A” at a position 2 with an element “a” in the list, we will have to call this method as:

```
list.set(2, "a");
```

Let’s take an example program where we will remove an element using remove() method from the list

9. Object get(int index): It returns the element at the specified position in this list..

ArrayList Methods in Java

10. int size(): It returns the number of elements of the list. Size means the number of elements present in the array list. Capacity means the capability to store elements.

ArrayList Methods in Java

11. boolean contains(Object o): It returns true if a specified element is present in the list, else, returns false. Look at the examples below.

```
String str = list.get(2);
```

```
int numberOfElements = list.size();
```

```
list.contains("A");
```

ArrayList Methods in Java

12. int indexOf(Object o): It is used to get the index of the first occurrence of the specified element, if the element is not present in the list, it returns the value -1.

13. int lastIndexOf(Object o): It is used to get the index of the last occurrence of the specified element in the list. If the element is not present in the list, it returns -1. For example:

```
int pos = list.indexOf("Apple");
int lastPos = list.lastIndexOf(20)
```

How to manually inc/dec current capacity of ArrayList?

1. ensureCapacity(): This method is used to increase the current capacity of ArrayList. Since the capacity of an array list is automatically increased when we add more elements. But to increase manually, ensureCapacity() method of ArrayList class is used.

2. trimTosize(): The trimTosize() method is used to trim the capacity of ArrayList to the current size of ArrayList.

```
ArrayList<String> list = new ArrayList<String>(); // Here, list can hold 10 elements.(Default initial capacity).
  list.ensureCapacity(20); // Now it can hold 20 elements.
list.trimTosize();
```

When to use ArrayList in Java?

ArrayList can be used in an application program when

- We want to store duplicate elements.
- We want to store null elements.
- It is more preferred in Java when getting of the element is more as compared to adding and removing elements.
- We are not working in the multithreading environment in Java because ArrayList is non-synchronized.

END