# ARRAYLIST IN JAVA

ArrayList is part of the collections framework. ArrayList is a List and implements the java.util.list interface.

ArrayList is a better alternative to Arrays, especially if you are not sure about the array size. Unlike array which have a fixed size, ArrayList can grow in size when needed.

Internally ArrayList also uses arrays to store data. When it reaches the current capacity and needs to grow, a new array is created and elements are copied from the old array to the new array. In the newer code, ArrayList is used along with generics.

ArrayList has many advantages over arrays such as:

- It dynamically resizes based on the number of elements in the list.
- It reduces the memory footprint. In case of arrays, we need to allocate memory upfront as we cannot change later. So extra memory is allocated even if we don't use it.

If you are sure about the number of elements or if primitive data types are the elements we can use Arrays instead.

The ArrayList class is similar to a Vector as both can grow in size as needed. But unlike Vector, ArrayList is not synchronized. ArrayList is always preferred over a Vector. There are many ways through which we can make an ArrayList thread safe like Collections.synchronizedList(theArrayList) or we can use a use a CopyOnWriteArrayList.

If simultaneous overwrite occurs in an ArrayList, a ConcurrentModificationException exception is thrown. Synchronizing a class when not required can affect the performance.

## Creating ArrayList

There are three constructors for creating an ArrayList.

When you create an ArrayList using the constructor **ArrayList()**, the internal array for storage is created with a size 10.
The constructor **ArrayList(int initialCapacity)** allows us to set the initial capacity.
The constructor **ArrayList( Collection)** constructs a list containing the elements of the specified collection.
In the newer code, ArrayList is used along with generics:

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

```
ArrayList<String> list = new ArrayList<String>(20);
```

```
ArrayList<String> list = new ArrayList<String>(myCollection);
```

## Adding elements

We can **add elements** to an array list using the **add or addAll** methods that appends one or more elements to the end of the list, or an **overloaded add or addAll** with an index argument that inserts one or more elements at a position within the list.

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

names.add("Heartin");

names.add("Sneha");

names.add(1, "Jacob");

System.out.println(names);

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

namesNew.addAll(names);

System.out.println(namesNew);

namesNew.addAll(2,names);

System.out.println(namesNew);
```

This will print:

```
[Heartin, Jacob, Sneha]

[Heartin, Jacob, Sneha]

[Heartin, Jacob, Heartin, Jacob, Sneha, Sneha]
```

## Changing elements

We can **modify an element** of a list using the set method.

```
names.set(1, "June");
```

## Retrieving elements and index

To **retrieve an element** at a given position, use the **get method**.

System.out.println(**names.get(2)**);

The index of the first occurrence of an element can be obtained using the **indexOf method**.
System.out.println(**names.indexOf("Jacob")**);
-1 is returned if the element is not present. The **index of the last occurrence of an element** can be obtained using **lastIndexOf method**.

## Traversing

We can **traverse an ArrayList** using:

- Simple for statement
- for-each statement
- Iterator
- ListIterator

**Example - for each loop**

```
for(String str : names)
{
  System.out.println(str);
}
```

**Example – for loop**

```
for(int i=0; i<names.size();i++)
{
  System.out.println(names.get(i));
}
```

**Example – Iterator**

```
Iterator<String> iterator = names.iterator();
while (iterator.hasNext()) {
```

```
  System.out.println(iterator.next());
}
```

**Example – ListIterator**

ListIterator can be used to traverse the list in both directions.

```
ListIterator<String> listIterator = names.listIterator();
while (listIterator.hasNext()) {
  System.out.println(listIterator.next());
}
while(listIterator.hasPrevious()) {
  System.out.println(listIterator.previous());
}
```

This will print:

```
Heartin
June
Sneha
Sneha
June
Heartin
```

First while loop will traverse in the forward direction from first reaching the last and then the second while loop will traverse in the reverse direction.

Note that if we put the hasPrevious while loop before the next while loop, output will be only:

```
Heartin
June
Sneha
```

## Sorting

**Collections.sort** method can be used to sort an ArrayList:

```
Collections.sort(names);
```

## Removing elements

The **clear method** will remove all elements. The **remove method** removes a single element and the**removeAll** method removes all values in a given collection from the list. The **retainAll** method retains all values in a given collection from the list.

```
namesNew.remove(0);

namesNew.remove("Jacob");

namesNew.retainAll(names);

namesNew.removeAll(names);

namesNew.clear();
```

Source : http://javajee.com/arraylist-in-java