**How ArrayList Works Internally in Java**

ArrayList arguably would be the most used collection along with the [HashMap](https://www.netjstech.com/2015/11/difference-between-hashmap-and-hashtable-java.html). Many of us programmers whip up code everyday which contains atleast one of these data structures to hold objects. I have already discussed [how HashMap works internally in Java](https://www.netjstech.com/2015/05/how-hashmap-internally-works-in-java.html), in this post I'll try to explain **how ArrayList internally works in Java**.

As most of us would already be knowing that **ArrayList is a Resizable-array implementation** of the List interface i.e. ArrayList grows dynamically as the elements are added to it. So let's try to get clear idea about the following points-

* How ArrayList is internally implemented in Java.
* What is the backing data structure for an [ArrayList](https://www.netjstech.com/2015/09/arraylist-in-java.html" \t "_blank).
* How it grows dynamically and ensures that there is always room to add elements.

Because of all these side questions it is also a very important [Java Collections interview question](https://www.netjstech.com/2015/11/java-collections-interview-questions.html).

**Note** - Code of ArrayList used here for reference is from **Java 10**.

**Table of contents**

1. [Where does ArrayList internally store elements](https://www.netjstech.com/2015/08/how-arraylist-works-internally-in-java.html#arraylistinternalstorage)
2. [What happens when ArrayList is created](https://www.netjstech.com/2015/08/how-arraylist-works-internally-in-java.html#arraylistcreation)
3. [How does ArrayList grow dynamically](https://www.netjstech.com/2015/08/how-arraylist-works-internally-in-java.html#arraylistdynamic)
4. [What happens when an element is removed from ArrayList](https://www.netjstech.com/2015/08/how-arraylist-works-internally-in-java.html#arraylistelementremoval)

**Where does ArrayList internally store elements**

Basic data structure used by Java ArrayList to store objects is an [array](https://www.netjstech.com/2017/02/array-in-java.html) of [Object class](https://www.netjstech.com/2017/06/object-class-in-java.html), which is defined as follows -

transient Object[] elementData;

I am sure many of you would be thinking why [transient](https://www.netjstech.com/2017/04/transient-in-java.html) and how about [serializing](https://www.netjstech.com/2017/04/serialization-in-java.html) an ArrayList then?  
ArrayList provides its own version of **readObject** and **writeObject** methods so no problem in serializing an ArrayList and that is the reason, I think, of making this Object array as **transient**.

**What happens when ArrayList is created**

ArrayList class in Java provides **3 constructors** to create an ArrayList.

* **public ArrayList(int initialCapacity)** - When this [constructor](https://www.netjstech.com/2015/04/constructor-in-java.html) is used we can provide some initial capacity rather than depending on the default capacity as defined in the ArrayList class.  
  **As example -**

List<String> myList = new ArrayList<String>(7);

Code in the ArrayList class is as -

public ArrayList(int initialCapacity) {

if (initialCapacity > 0) {

this.elementData = new Object[initialCapacity];

} else if (initialCapacity == 0) {

this.elementData = EMPTY\_ELEMENTDATA;

} else {

throw new IllegalArgumentException("Illegal Capacity: "+

initialCapacity);

}

}

Where **EMPTY\_ELEMENTDATA** is defined as -

private static final Object[] EMPTY\_ELEMENTDATA = {};

It is easy to see that, if provided capacity is greater than zero then the elementData array will be created with that capacity, in case provided capacity is zero then elementData array is initialized with an empty Object array. In that case ArrayList will grow when first element is added.

* **public ArrayList()** - In case **default constructor** is used i.e. ArrayList is created like -

myList = new ArrayList();

Code in the ArrayList class is as -

public ArrayList() {

this.elementData = DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA;

}

Where **DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA** is defined as

/\*\*

\* Shared empty array instance used for default sized empty instances. We

\* distinguish this from EMPTY\_ELEMENTDATA to know how much to inflate when

\* first element is added.

\*/

private static final Object[] DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA = {};

So you can see initially it will be initialized with an empty array, it will grow only when first element is added to the list.

* **public ArrayList(Collection<? extends E> c)** - If we want to construct a list containing the elements of the specified collection we can use this [constructor](https://www.netjstech.com/2015/04/constructor-chaining-in-java-calling-one-constructor-from-another.html). In this constructor implementation checks for the length of the collection passed as parameter, if length is greater than zero then **Arrays.copyOf** method is used to copy the collection to the elementData array.

elementData = Arrays.copyOf(elementData, size, Object[].class);

**How does ArrayList grow dynamically**

When we add an element to an ArrayList it first verifies whether it has that much capacity in the array to store new element or not, in case there is not then the new capacity is calculated which is 50% more than the old capacity and the array is increased by that much capacity (Actually uses Arrays.copyOf which returns the original array increased to the new length).

Code in the Java ArrayList implementation is like this-

public boolean add(E e) {

ensureCapacityInternal(size + 1); // Increments modCount!!

elementData[size++] = e;

return true;

}

private void ensureCapacityInternal(int minCapacity) {

if (elementData == DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA) {

minCapacity = Math.max(DEFAULT\_CAPACITY, minCapacity);

}

ensureExplicitCapacity(minCapacity);

}

Where DEFAULT\_CAPACITY is defined as -

private static final int DEFAULT\_CAPACITY = 10;

private void ensureExplicitCapacity(int minCapacity) {

modCount++;

// overflow-conscious code

if (minCapacity - elementData.length > 0)

grow(minCapacity);

}

You can see here it is determined if there is a need to increase the size of the array, if yes then grow method is called.

private void grow(int minCapacity) {

// overflow-conscious code

int oldCapacity = elementData.length;

int newCapacity = oldCapacity + (oldCapacity >> 1);

if (newCapacity - minCapacity < 0)

newCapacity = minCapacity;

if (newCapacity - MAX\_ARRAY\_SIZE > 0)

newCapacity = hugeCapacity(minCapacity);

// minCapacity is usually close to size, so this is a win:

elementData = Arrays.copyOf(elementData, newCapacity);

}

Note that till **Java 6** the new capacity calculation used to be like this -

int newCapacity = (oldCapacity \* 3)/2 + 1;

Which is changed in **Java 7** to use right shift operator. With right shift operator also it will grow by 50% of old capacity.  
Let's see it with the help of a small program

public class Test {

public static void main(String args[]) {

int a = 10;

System.out.println(a>>1);

}

}

**Output**

5

If the default capacity was 10 then

int newCapacity = oldCapacity + (oldCapacity >> 1);

will return 15.

**What happens when an element is removed from ArrayList**

When elements are removed from an ArrayList in Java using either **remove(int i)** (i.e using index) or **remove(Object o)**, gap created by the removal of an element has to be filled in the underlying array. That is done by Shifting any subsequent elements to the left (subtracts one from their indices). **System.arrayCopy** method is used for that.

System.arraycopy(elementData, index+1, elementData, index, numMoved);

Here index+1 is the source position and index is the destination position. Since element at the position index is removed so elements starting from index+1 are copied to destination starting from index.

**Points to note**

1. ArrayList in Java is a Resizable-array implementation of the List interface.
2. Internally ArrayList class uses an array of Object class to store its elements.
3. When initializing an ArrayList you can provide initial capacity then the array would be of the size provided as initial capacity.
4. If initial capacity is not specified then default capacity is used to create an array. Default capacity is 10.
5. When an element is added to an ArrayList it first verifies whether it can accommodate the new element or it needs to grow, in case capacity has to be increased then the new capacity is calculated which is 50% more than the old capacity and the array is increased by that much capacity.
6. When elements are removed from an ArrayList space created by the removal of an element has to be filled in the underlying array. That is done by Shifting any subsequent elements to the left.

**Recommendations for learning**

1. [Java Programming Masterclass Course](https://click.linksynergy.com/deeplink?id=*H/8FfjgiRQ&mid=39197&murl=https%3A%2F%2Fwww.udemy.com%2Fjava-the-complete-java-developer-course%2F)
2. [Java In-Depth: Become a Complete Java Engineer!](https://click.linksynergy.com/deeplink?id=*H/8FfjgiRQ&mid=39197&murl=https%3A%2F%2Fwww.udemy.com%2Fjava-in-depth-become-a-complete-java-engineer%2F)
3. [Spring Framework Master Class Course](https://click.linksynergy.com/deeplink?id=*H/8FfjgiRQ&mid=39197&murl=https%3A%2F%2Fwww.udemy.com%2Fspring-tutorial-for-beginners%2F)
4. [Complete Python Bootcamp Course](https://click.linksynergy.com/deeplink?id=*H/8FfjgiRQ&mid=39197&murl=https%3A%2F%2Fwww.udemy.com%2Fcomplete-python-bootcamp%2F)
5. [Python for Data Science and Machine Learning](https://click.linksynergy.com/deeplink?id=*H/8FfjgiRQ&mid=39197&murl=https%3A%2F%2Fwww.udemy.com%2Fpython-for-data-science-and-machine-learning-bootcamp%2F)

That's all for this topic **How ArrayList Works Internally in Java**. If you have any doubt or any suggestions to make please drop a comment. Thanks!