**ArrayList**

ArrayList is fundamentally an array with growing size. This means its size is not fixed.

The size of ArrayList is dynamic.

Let’s say we have an ArrayList with 5 elements.

ArrayList

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 7 | 9 | 11 | 13 |

Now if we add one more element 15 to the ArrayList then the size will increase by one.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 7 | 9 | 11 | 13 | 15 |

So, The ArrayList is simple a resizable array. ArrayList is backed by Array in memory that’s why ArrayList will have similar kind of functionalities like Array.

**Initializing ArrayList**

In Java

ArrayList<Integer> num\_arr\_list = new ArrayList<Integer> ();

While defining/initializing ArrayList we can pass the size of the ArrayList.

In Java

ArrayList<Integer> num\_arr\_list = new ArrayList<Integer> (50);

If we not specified the size by default it will take 10.

But the size will increase dynamically as we add more elements into it.

**Note:** ArrayList will not support populating elements while defining/initializing like Array.

ArrayList<Integer> num\_arr\_list = new ArrayList<Integer> (1, 2, 3);

**Most commonly used array methods**

1. Add
2. Remove
3. Get
4. Set
5. Clear
6. Toarray

Let’s create an ArrayList of size 5

ArrayList<integer> num\_arr\_list = new ArrayList<integer>(5);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Value |  |  |  |  |  |

**Add method**

Add method comes in 2 different types.

1. One type will take the object to be inserted and this will be inserted at the end.
2. Other type will take the object to be insert and index where to be insert.

So, now we add the integer 7 to the ArrayList

num\_arr\_list.add (7);

**\*NOTE: -** Basically, ArrayList will only accepts objects to add. It won’t allow primitive datatypes like int, float, etc. However, the computer will automatically convert primitive integer value to integer object. This is called “AUTO BOXING”.

Now the ArrayList will have integer 2 at index 0.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Value | 7 |  |  |  |  |

Now if we add another object 5 to ArrayList it will added at index 1 since index 0 is already fill. **“**num\_arr\_list.add (5)**”**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Value | 7 | 5 |  |  |  |

**Conclusion:** In the 1st type add method the time complexity is O (1). Since it always going to add the given element at the end of the ArrayList, no matter what the size of ArrayList.

Now we use **2nd type add** method to add object into ArrayList at specified index.

num\_arr\_list.add (11, 0)

here we are adding object 11 to ArrayList at index 0. Since index 0 is already filled it moves the objects next to one index and make index 0 empty.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Value |  | 7 | 5 |  |  |

After making specified index 0 empty it will add the given object at specified index.

**Conclusion:** For this type of add method the time complexity is O(n). since it has to shift the objects next to one index.

**Remove**

The remove method comes in 2 different types.

1. Remove(index)
2. Remove(object)

**Remove(index)**

This method will remove the object at given index location in the ArrayList and returns the removed object.

Let’s create an ArrayList of size 5

ArrayList<integer> num\_arr\_list = new ArrayList<integer>(5);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Value | 2 | 5 | 8 | 7 | 9 |

Now we want to remove the object at index location 3.

num\_arr\_list. remove (3)

this will remove the object 7 at index location 3 and return the removed object back to user.

**Remove(object)**

This method will remove the given object in the ArrayList and returns True/False.

Let’s create an ArrayList of size 5

ArrayList<integer> num\_arr\_list = new ArrayList<integer>(5);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Value | 2 | 9 | 8 | 7 | 9 |

Now we want to remove the object 9 from the ArrayList.

num\_arr\_list. remove (new Integer (9))

this will remove the 1st occurrence of the object 9 from the ArrayList.

**Get**

Get(index)

This method will get / give the object specified at given index.

Let’s create an ArrayList of size 5

ArrayList<integer> num\_arr\_list = new ArrayList<integer>(5);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Value | 2 | 9 | 8 | 7 | 9 |

Now we want to get the object at index location 2.

num\_arr\_list. Get (2)

this will give the integer object 8.

**Set**

Set (index, object)

This method will set / replace the object with given object at given index.

Let’s create an ArrayList of size 5

ArrayList<integer> num\_arr\_list = new ArrayList<integer>(5);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Value | 2 | 9 | 8 | 7 | 9 |

Now we want to set the object 6 at index location 4.

num\_arr\_list. Set (4, 6)

this will replace the object 9 at index location 4 with new object 6.

**Clear**

Clear ()

This method will clear all the objects in the ArrayList and make it empty.

Let’s create an ArrayList of size 5

ArrayList<integer> num\_arr\_list = new ArrayList<integer>(5);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Value | 2 | 9 | 8 | 7 | 9 |

Now we want to clear / empty this ArrayList.

num\_arr\_list. Clear ()

**toArray**

This method will be used to convert ArrayList to Array. The resulted Array will have integer objects instead of primitive integer.

This means the resulted Array will have datatype objects instead of primitive datatypes.

Let’s create an ArrayList of size 5

ArrayList<integer> num\_arr\_list = new ArrayList<integer>(5);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Value | 2 | 9 | 8 | 7 | 9 |

Now we want to convert this ArrayList to Array.

int num\_arr [] = num\_arr\_list. toArray ()

this will convert the ArrayList to Array. The resulted Array will have integer objects instead of primitive integer.

**\*\*\*Note:**

The most important thing in ArrayList is that how it is storing the contiguous elements or objects.

Unlike Array the ArrayList will not store the values / objects at specified index, instead it will store the memory address pointer location where the values / objects are store in the memory.

ArrayList<integer> num\_arr\_list = new ArrayList<integer>(5);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Pointer address | 81 | 70 | 101 | 53 | 62 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Address in memory | 70 | 81 | 62 | 53 | 41 | 98 | 101 |
| Stored Value | 2 | 9 | 8 | 7 | ‘abc’ | 15.7 | 9 |

Now we look at Time complexity Big-O notations for ArrayList.

We all know that ArrayList is backed by Array.

So, the Time complexity equations for ArrayList are also same as the Array.

**Report of ArrayList**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accessing | Searching | Inserting | Deleting |
| Time complexity | O (1) | O(n) | O(n) | O(n) |

**Note:** Array and ArrayList are Random access data structures. Because we can access any item randomly using indexing without depending on other items.