ArrayList

1. The Underlying Data Structure for ArrayList is Resizable Array OR Growable Array.
2. Duplicates are allowed.
3. Insertion Order is Preserved.
4. Heterogeneous Objects are allowed (Except TreeSet and TreeMap Everywhere Heterogeneous Objects are allowed).
5. null Insertion is Possible.

Constructor

ArrayList l = new ArrayList();

1. Creates an Empty ArrayList Object with Default Initial Capacity 10
2. If ArrayList Reaches its Max Capacity then a New ArrayList Object will be Created with

- **Java 8 formula**

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

  1. E.g., if the Array size is 10 and it is fully filled with the elements, while adding new elements, array capacity will increase as 10+ (10>>1) => 10+ 5 => 15.
  2. I.e., Size of the array increase by 50%
  3. To increase the array size by 50%, we use the right shift operator

- **Java 6 formula**

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

While in **Java 6** it's totally different from the Java 8 calculation on increasing the size of the Array, in java 6 the capacity increases by the amount to **1.5X**

```
int newCapacity = (30 * 3)/2 +1;
                = 90/2 + 1
                =  45 + 1
                =    46
```

Overview of Arraylist

1. **ArrayList** is a **class** in java.util package
2. implements **dynamic-sized arrays** in Java

3. ArrayList **dynamically grows and shrinks** in size on the addition and removal of elements respectively.
4. ArrayList **inherits the AbstractList class** and **implements** the **List, RandomAccess and Serializable interfaces**

Important Features of ArrayList Java

1. **Dynamic Resizing**
2. Ordered - ArrayList **preserves the order** of the elements.
3. Index based - ArrayList in java **supports Random Access**. I.e., can directly access elements using index numbers.
4. Object based **- stores Object** type only (not primitive type like int, long,etc)
5. Not Synchronized - ArrayList in Java is **not synchronized**. So multiple threads can access the same time.

Time Complexity

1. Random Access - O(1)
2. Adding Element - O(1)
3. Inserting Element at location - O(n)
4. Deleting Element - O(n)
5. Searching- O(n)

How does ArrayList() work?

1. When ArrayList is created, its **default capacity is 10** if the user doesn't specify the capacity.
2. The Size of ArrayList **grows based on load factor and current capacity.**
    a. **Load Factor**
       It is a **measure to decide when to increase ArrayList capacity**.
    b. The **default value** of load factor is **0.75f**
3. ArrayList capacity expands its capacity after each threshold which is calculated as
   Threshold = (Load Factor) * (Current Capacity)
   = 0.75 * 10
   = 7
4. This means when we add the 7th element to the list, the **size will increase by 50% in Java 8.**
5. Internally, a new ArrayList is created with new capacity and the elements present in the old ArrayList are copied to the new ArrayList.
6. Old ArrayList will be removed by the Garbage Collector.

7. Java 8 formula

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

   - E.g., if the Array size is 10 and it reaches the threshold value, while adding new elements, array capacity will increase as 10+ (10>>1) => 10+ 5 => 15.
   - I.e., Size of the array increase by 50%
   - To increase the array size by 50%, we use the right shift operator

Constructors in ArrayList

1. **ArrayList<>()**
   his constructor is the default one, it creates an empty ArrayList instance with default initial capacity i.e. 10.
   E.g., ArrayList<String> a = new ArrayList<>();

2. **ArrayList(int capacity)**
   This constructor creates an empty ArrayList with initial capacity as mentioned by the user.
   E.g., ArrayList<String> a = new ArrayList(50);

3. **ArrayList(Collection c)**
   This constructor creates an ArrayList and stores the elements that are present in the collection list.
   E.g., List<String> **list** = Arrays.asList(new String[]{"foo", "bar"});
   List<String> arr = new ArrayList<>(**list**);

Note:
If you want to create an ArrayList instance which can store Objects of multiple types, don't parameterize the instance.
E.g.,

```
ArrayList a = new ArrayList<>();

a.add(Integer.valueOf(1));

a.add("Test");
```

Methods in ArrayList

Method	Description
add(Object o)	Adds a specific element at the end of ArrayList.
add(int index, Object o)	Inserts a specific element into the ArrayList at the specified

	index.
addAll(Collection c)	This method is used to add all the elements present in a specified collection at the end of the ArrayList
addAll(int index, Collection c)	Inserts all the elements present in a specified collection into the ArrayList starting with the specified index.
remove(Object o)	Removes the first occurrence of the specified element from the ArrayList if present.
remove(int index)	Removes the element present at specified index from the ArrayList.
removeAll(Collection c)	Removes all the elements present in a specified collection from the ArrayList.
clear()	This method is used to remove all the elements from the ArrayList.
contains(Object o)	Returns true if the ArrayList contains the specified element.
get(int index)	Returns the element at the specified index in the ArrayList.
indexOf(Object o)	Returns the index of the first occurrence of the specified element in the ArrayList, or -1 if the list does not contain the element.
isEmpty()	Returns true if the ArrayList contains no elements.
lastIndexOf(Object o)	Returns the index of the last occurrence of the specified element in the ArrayList, or -1 if the list does not contain the element.
listIterator()	Returns a list iterator over the elements in the ArrayList in proper sequence.
listIterator(int index)	Returns a list iterator over the elements in the ArrayList in proper sequence, starting at the specified index in the list.
set(int index, String e)	used to replace the element at the specified position in the ArrayList with the specified element.
toArray()	returns an array containing all the elements present in the ArrayList in

	proper sequence.
subList(int fromIndex, int toIndex)	Returns a view of the portion of this list between the specified fromIndex (inclusive) and toIndex (exclusive)
trimToSize()	Trims the capacity of the ArrayList instance to the list's current size
size()	Returns the number of elements present in the ArrayList.

Note:
We can only add objects in the ArrayList but if we want to add primitive data types such as int, float, etc., we can use wrapper class for such cases.

How to Sort ArrayList in Java?

We can sort an ArrayList using the **sort() method** of the Collection framework in Java

Syntax:

Collections.sort(ArrayList)

- it takes the object of an ArrayList and sorts the ArrayList in the ascending order according to the natural ordering of its elements.
- e.g.,

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

names.add("Raj");

names.add("Priya");

names.add("Shashank");

names.add("Ansh");

System.out.println("Before sorting, names : " + names);

Collections.sort(names);

System.out.println("Before sorting, names : " + names);
```

Output:

```
Before sorting, names : [Raj, Priya, Shashank, Ansh]

Before sorting, names : [Ansh, Priya, Raj, Shashank]
```

Change an element in ArrayList

- In order to change an element in the ArrayList, we can use the **set()** method.
- Syntax

  set(int index, String e)
- we provide the index and the new element as arguments which in return replaces the element present at the specified index with the new element.
- e.g.,

```
ArrayList<String> names = new ArrayList<String>();
names.add("Raj");
names.add("Priya");
names.add("Shashank");
names.add("Ansh");
System.out.println("Before set, names : " + names);
names.set(0, "Rahil");
System.out.println("After set element at 0 "+ names);
```

Output:

```
Before set, names : [Raj, Priya, Shashank, Ansh]
After set element at 0 [Rahil, Priya, Raj, Shashank]
```

How to Remove Elements from ArrayList?

- We can remove elements from an ArrayList with the help of remove() method.
- Syntax
  - remove(Object o)
  - remove(index i)
- e.g.,

```
ArrayList<String> colors = new ArrayList<String>();
colors.add("red");
colors.add("orange");
colors.add("blue");
colors.add("pink");
colors.add("black");
colors.add("green");
System.out.println("ArrayList colors : " + colors);

// removing element pink from the ArrayList
colors.remove("pink");
System.out.println("ArrayList colors : " + colors);

// removing 3rd element from the ArrayList
colors.remove(2);
System.out.println("ArrayList colors : " + colors);
```

Output:

```
ArrayList colors : [red, orange, blue, pink, black, green]
ArrayList colors : [red, orange, blue, black, green]
ArrayList colors : [red, orange, black, green]
```

Iterating ArrayList

- Printing elements **using a simple loop**

  ```
  for (int i = 0; i < colors.size(); i++) {
      System.out.print(colors.get(i) + " ");
  }
  ```

  Passing index to get() method to get an elements

- Printing elements **using for each loop**

  ```
    for (String color : colors)
      System.out.print(color + " ");
  ```

- **Using listIterator()**
  - An iterator is an object that enables a programmer to traverse through collections such as ArrayList, HashSet, etc
  - Syntax:
    public ListIterator<E> listIterator()
  - This method **returns a listIterator** over the elements in the ArrayList in proper sequence.
  - The **returned listIterator is fail-fast.**
- e.g.,

  ```
  ListIterator<String> iterator = colors.listIterator();

  while(iterator.hasNext())
  {
          System.out.println(iterator.next());
  }
  ```

Note:

- **Fail-Fast iterators** immediately **throw ConcurrentModificationException** if there is **structural modification of the collection.**
- Structural modification means adding, removing any element from a collection while a thread is iterating over that collection.

Finding the length of the ArrayList

- In order to find the length of ArrayList, we can **use** the **size() method.**
- **size()**: It **returns the number of elements** present in an ArrayList.
- e.g.,

```
ArrayList<String> colors = new ArrayList<String>();
colors.add("red");
colors.add("orange");
colors.size();
```

How to check if ArrayList is Empty in Java?

- **isEmpty()** - we can use the isEmpty() method to find whether an ArrayList is empty or not.
- Syntax:
  public boolean isEmpty()
  This method returns true if an ArrayList is empty.
- e.g.,

```
ArrayList<String> colors = new ArrayList<String>();
colors.add("red");
colors.add("orange");
// removing all elements
colors.clear();
System.out.println("ArrayList is empty "+ colors.isEmpty());
```

How to synchronize ArrayList in Java?

- ArrayList is not synchronized.
- i.e., multiple threads can access ArrayList at same time.
- We **can synchronize the ArrayList** using below method
  - **Collections.synchronizedList()** method
  - Using **CopyOnWriteArrayList (it is thread-safe variant of ArrayList)**

1. Using **Collections.synchronizedList()** method
   a. To synchronize the ArrayList, we can use **Collections.synchronizedList()** method.
   b. Syntax

```
List<String> animal = new ArrayList<String>();
animal = Collections.synchronizedList(animal);
```

   c. If we make ArrayList synchronized, then **iterator() must be declared in a synchronized block for synchronized ArrayList**.

d. e.g.,

```
List<String> animal = new ArrayList<String>();
animal = Collections.synchronizedList(animal);

synchronized (animal)
{
        ListIterator<String> listIterator = animal.listIterator();
        while(listIterator.hasNext())
        {
                System.out.println(listIterator.next());
        }
}

synchronized (animal)
{
        Iterator iterator = animal.iterator();
        while(iterator.hasNext())
        {
                System.out.println(iterator.next());
        }
}
```

2. Using **CopyOnWriteArrayList**
   a. Using CopyOnWriteList, we can **create a synchronized ArrayList**.
   b. Syntax:

```
CopyOnWriteArrayList<String> arr = new CopyOnWriteArrayList<>();
```

   c. Example

```
CopyOnWriteArrayList<String> arr = new CopyOnWriteArrayList<>();

arr.add("I");
arr.add("am");
arr.add("a");
arr.add("Synchronized");
arr.add("ArrayList");

Iterator itr = arr.iterator();

while(itr.hasNext())
{
        System.out.println();
}
```