Difference between add() and set() methods

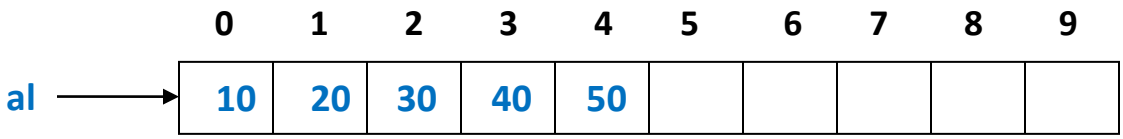
- ***add(index, object):***

If add(index, object) method is used to add an element at a specified index and if an element already exists at the specified position, then all the elements are shifted by one position to the right and then the new element will be added to the specified position. In other words, the existing element will not be replaced.

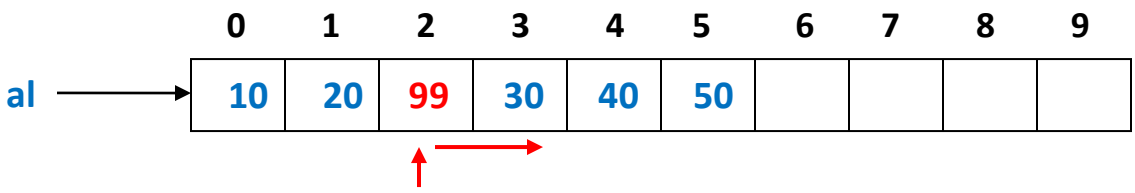
Let us understand this with the help of Code:

```
import java.util.ArrayList;
class Launch
{
    public static void main(String[] args)
    {
        ArrayList al = new ArrayList();
        al.add(10);
        al.add(20);
        al.add(30);
        al.add(40);
        al.add(50);
        System.out.println(al);

        al.add(2,99);

        System.out.println(al);
    }
}
```

Before using add() method:



After using add() method:



Output:

```
[ 10, 20, 30, 40, 50 ]
[ 10, 20, 99, 30, 40, 50 ]
Press any key to continue…
```

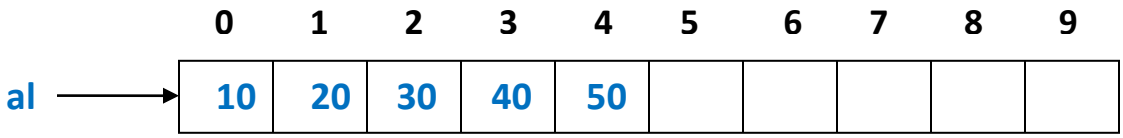
- ***set(index, object):***

If set(index, object) method is used to add an element at a specified index and if an element already exists at that specified position, then the existing element is replaced by the new element.

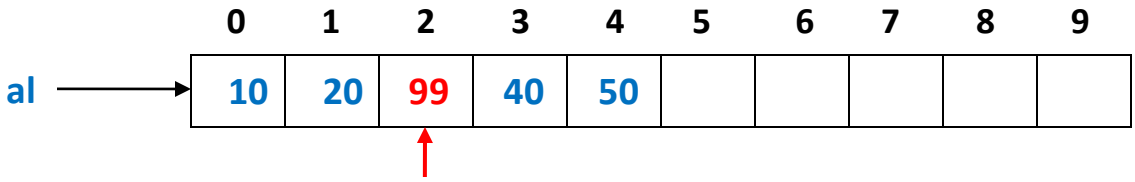
Code:

```
import java.util.ArrayList;
class Launch
{
    public static void main(String[] args)
    {
        ArrayList al = new ArrayList();
        al.add(10);
        al.add(20);
        al.add(30);
        al.add(40);
        al.add(50);
        System.out.println(al);

        al.set(2,99);

        System.out.println(al);
    }
}
```

Before using set() method:



After using set() method:



Output:

```
[ 10, 20, 30, 40, 50 ]
[ 10, 20, 99, 40, 50 ]
Press any key to continue…
```

Dynamic Array

Standard Java arrays are of fixed length. After arrays are created, they cannot grow or shrink, which means that you must know in advance how many elements an array can hold.

Array lists are created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array may be shrunk. Array List supports dynamic arrays that can grow as needed.

An array which can grow in size or shrink in size is called as ***Resizable array*** or ***Dynamic array*****.**

But how does this happen??

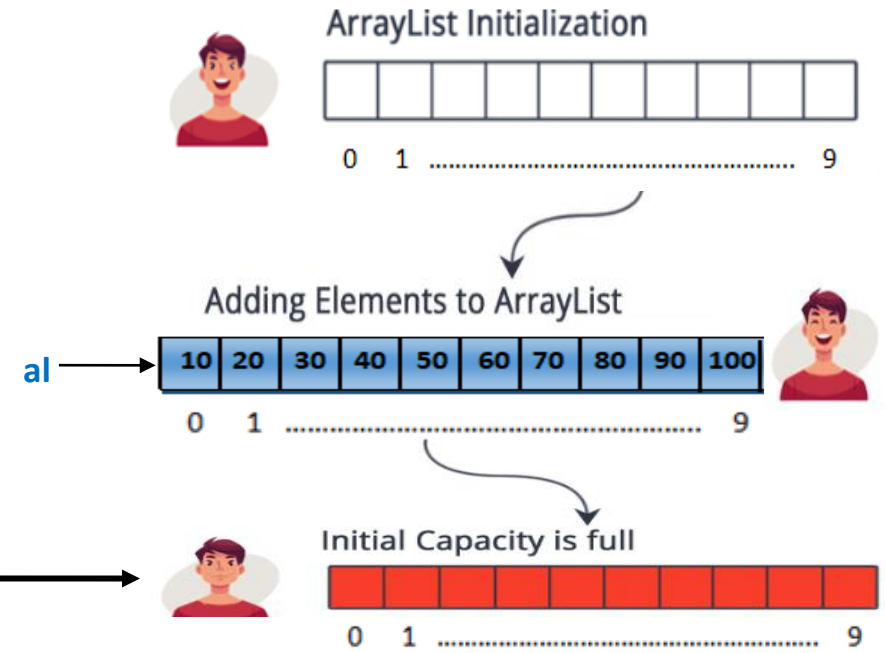
Whenever we create an ***ArrayList*** object automatically a regular array is created whose size initially will be of 10 (0 to 9).

ArrayList al = new ArrayList();

al.add(10);
al.add(20);
al.add(30);

.
.

al.add(100);
al.add(110);



Now the size of array is filled how do we add the value 110 and where do we??

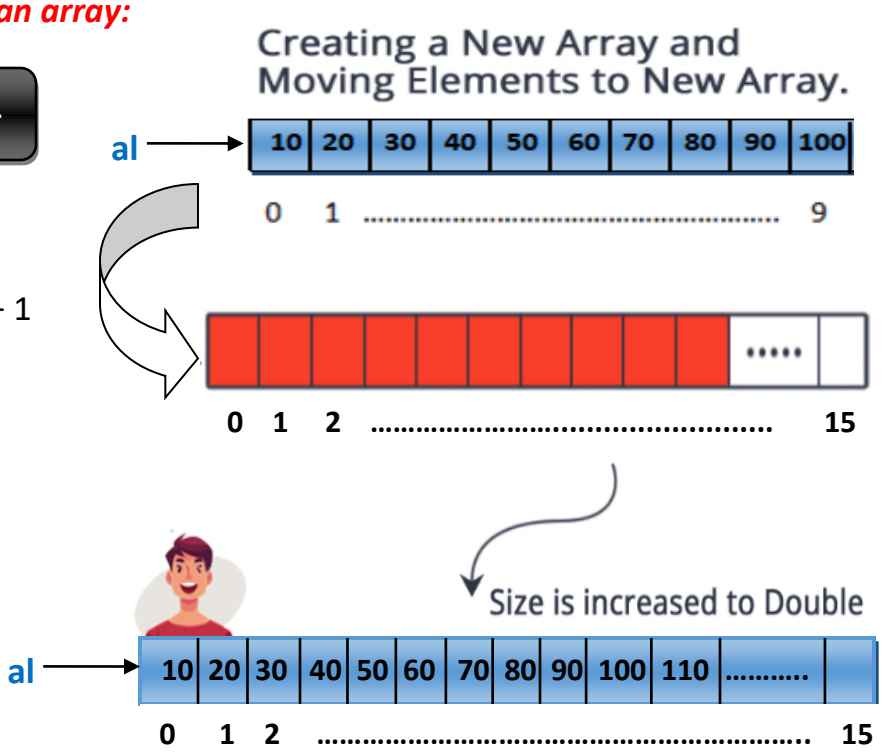
The initial capacity of the resizable array is 10. Once this capacity is filled a new array is created whose size is determined by using a formula.

Formula to find the new size of an array:

New Size = 3/2 * Old Size + 1

Therefore, **New Size** = 3/2 * 10 + 1

New Size = 16



All the elements of the previous array are copied to this new array and the reference is reassigned to this array. The old array becomes a garbage object and is deallocated.

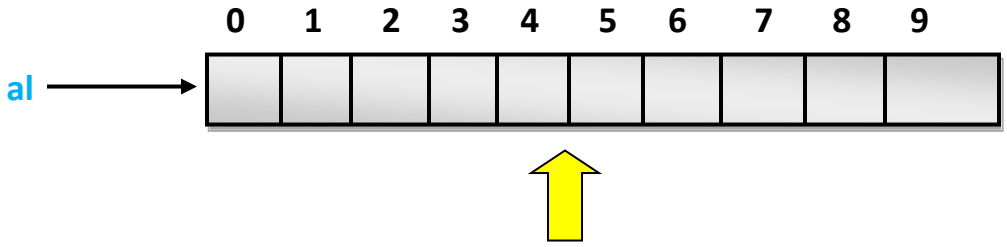
The use of size() and trimToSize() method

size(): The **size()** method of java.until.ArrayList class is used to get the number of elements in this list.
Whenever we create an array the initial capacity by default will be of 10.

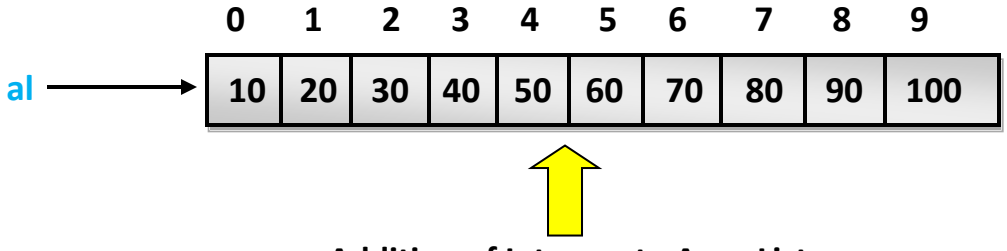
trimToSize(): This method is used to make the capacity of the arraylist equal to its size (number of elements stored). This is a useful method to avoid wastage of memory.

Example:

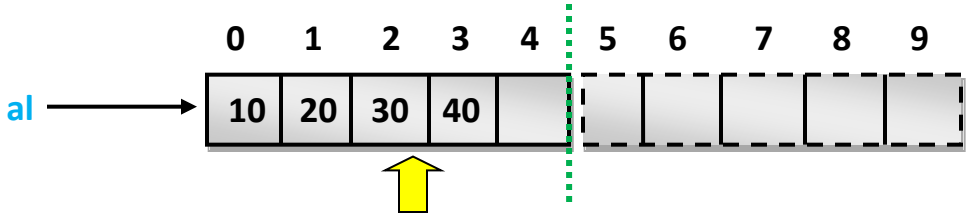
```
import java.util.ArrayList;
class Demo
{
    public static void main(String[] args)
    {
        ArrayList al = new ArrayList();
        System.out.println(al.size());
        al.add(10);
        al.add(20);
        al.add(30);
        al.add(40);
        al.add(50);
        System.out.println(al);

        System.out.println(al.size());
        al.trimToSize();
        System.out.println(al);
    }
}
```



Initial size of ArrayList = 9

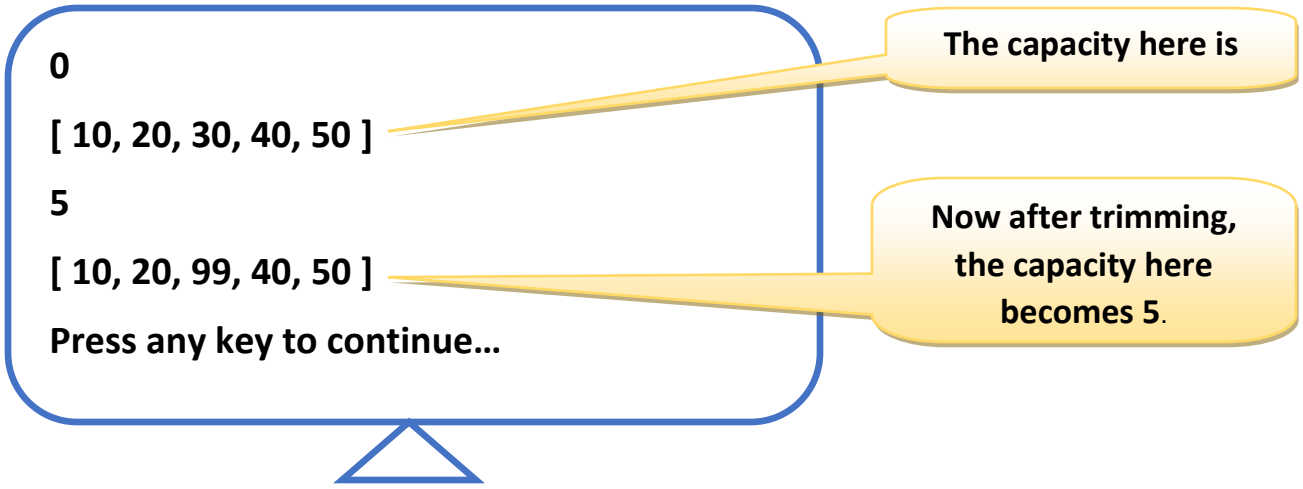


Addition of Integers to ArrayList



Trims the size of ArrayList to 5 using *trimToSize()*

Output:



When to use Array over ArrayList??

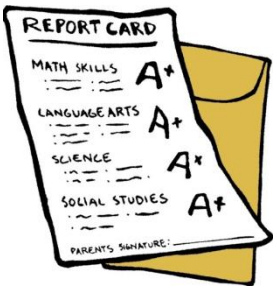
We know that ArrayList is advantageous when compared to Array. But at few conditions Array is more preferable when compared to ArrayList.
They are:

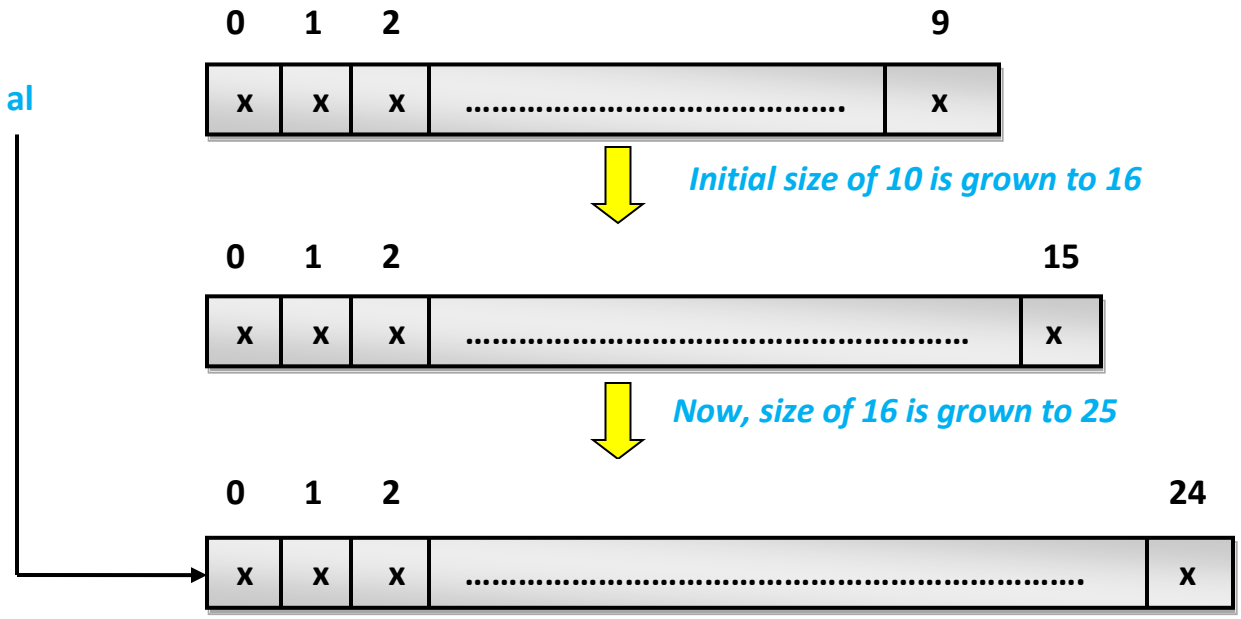
- ***When the size of the data is known before.***
- ***The data we are storing are homogeneous data.***

ArrayList Approach

For example, when we want to store marks of 25 students.

When create an AraryList object initially the size that gets allocated is 10. To store the extra data the array has to grow in size and all the elements of an array has to get copied to the new array and the reference has to get changed, all these has to happen twice.





Array Approach

Since, we know the number of students and the type of marks to be stored, directly we can create an array.

int a[ ] = new int[25] ;

Using the above statement we can directly create an array of 25 students.

Therefore, Arrays are faster in execution when compared to arraylists. Hence whenever the above two conditions are satisfied it is always better to choose an array over the arraylist to store the data.

The use of conatins(), containsAll(), getClass() methods.

The **contains()** method is used to determines whether an element exists in an ArrayList object. Returns true if this list contains the specified element.

The **containsAll()** method of List interface in Java is used to check if this List contains all of the elements in the specified Collection. So basically it is used to check if a List contains a set of elements or not.

The **getClass()** is the method of Object class. This method returns the runtime class of this object. The class object which is returned is the object that is locked by static synchronized method of the represented class.

```
import java.util.ArrayList;
class Demo
{
    public static void main(String[] args)
    {
        ArrayList al = new ArrayList();
        al.add(10);
        al.add(20);
        al.add(30);
        al.add(40);
        al.add(50);
        System.out.println(al);
        System.out.println(al.contains(50));
        System.out.println(al.contains(99));

        ArrayList b = new ArrayList();
        b.add(10);
        b.add(20);
        b.add(30);
        b.add(40);
        b.add(50);
        System.out.println(b);
        System.out.println(b.containsAll(b));
        System.out.println(b.getClass());
    }
}
```

Output

```
[ 10, 20 , 30, 40, 50 ]
true
false
```

```
[ 10, 20 , 30, 40, 50 ]
true
class java.util.Arraylist
```

When to use clone() method ?

ArrayList **clone()** method is used to create a shallow copy of the list.
In the new list, only object references are copied.

```
import java.util.ArrayList;
class Demo
{
    public static void main(String[] args)
    {
        ArrayList al = new ArrayList();
        al.add(10);
        al.add(20);
        al.add(30);
        al.add(40);
        al.add(50);
        System.out.println(al);

        ArrayList b = (ArrayList)al.clone();
        System.out.println(b);
    }
}
```

Output:

[10, 20, 30, 40, 50]

[10, 20, 30, 40, 50]

Press any key to continue…