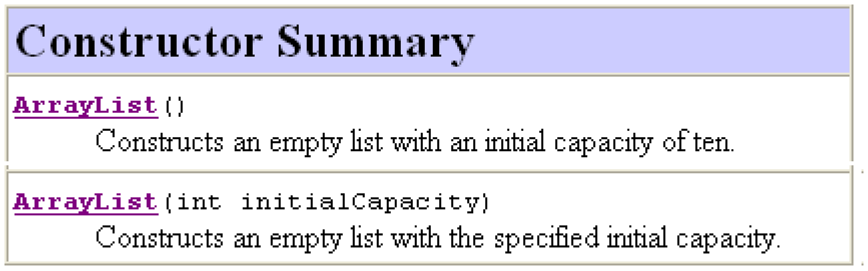
Virtual Lecture Notes (Part 1)

The **`ArrayList`** data structure solves one limitation of the array; an **`ArrayList`** can change size, an array cannot. The fixed size of an array can be circumvented by copying an array to one with more elements, and this is often done, but extra coding is required. Another benefit of an **`ArrayList`** is that methods for adding, setting, getting, and removing elements from the list are already written. To perform these actions with an array, specific methods must be written by the programmer.

To some extent, the only drawback to an **`ArrayList`** is that everything must be stored as an object. With the new release of Java 1.5, this restriction is of little consequence. When you open the Java API and examine the documentation for the **`ArrayList`** class, the first thing to notice is that the constructor is overloaded.



Constructor Summary
`ArrayList()` Constructs an empty list with an initial capacity of ten.
`ArrayList(int initialCapacity)` Constructs an empty list with the specified initial capacity.

The default constructor takes no parameters and sets the initial size to 10. The second constructor takes one **`int`** parameter, which allows the size to be intentionally set. Browse through the rest of the documentation and notice the different methods that are already available.

Open the **`IntegerArrayList`** class that you downloaded to the 08.09 Lessons project and locate the statement that invokes the **`ArrayList`** constructor.

```
ArrayList<Integer> intList = new ArrayList<Integer>();
```

It should look very familiar by now, with one minor variation. The type of an **`ArrayList`** must be included within a pair of angle brackets.

Next, examine the first loop and notice the use of the **`add()`** method to assign random numbers to index positions in the **`ArrayList`**.

```
for(int i = 0; i < 50; i ++)
{
     rndNumber = (int)(Math.random() * 100);
     intList.add(rndNumber);
}
```

In the second loop, notice the use of two other methods of the **`ArrayList`** class: **`get()`** and **`remove()`**. What numbers are being removed from the list?

```
for(int i = 0; i < intList.size(); i++)
{
     if(intList.get(i) < 50)
     {
         intList.remove(i);
         i--;
     }
}
```

In the last loop, a new set of negative random numbers are chosen and the **`set()`** method is used to place them at each index position in the **`ArrayList`**.

```
for(int i = 0; i < intList.size(); i++)
{
     rndNumber = (int)(Math.random() * -100);
     intList.set(i, rndNumber);
}
```

Finally, in the last loop, see if you can figure out what is happening with the **`add()`** method.

```
    for(int n = 0; n < 10; n?
    {
        rndNumber = (int)(Math.random() * 100) + 100;
        position = (int)((Math.random() * intList.size()));
        intList.add(position,rndNumber);
    }
```

Note:
Before moving on to Part 3 of the Lesson, modify the demo program to work with an `ArrayList` of `Doubles` instead of `Integers`.