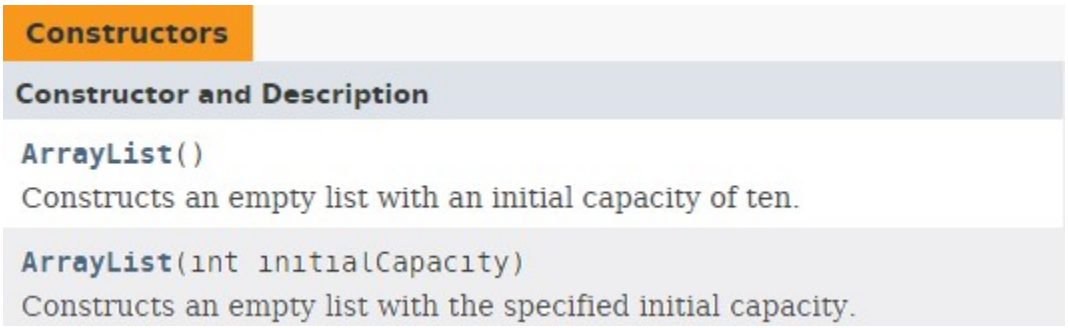
07.01 Virtual Lecture Notes

The **`ArrayList`** data structure solves one limitation of the array; an **`ArrayList`** can change size, an array cannot. 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.

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.



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 07.01 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. Notice also, the type of the ArrayList is Integer with a capital "I." ArrayLists cannot contain primitive data types so an object must be used.

In order to use the ArrayLIst class, it must be imported. Notice the import statement above the class statement. If the import statement is missing, the program will throw errors.

Next, examine the first loop and notice the use of the **`add()`** method to assign random numbers to index positions at the end of 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? Numbers greater than 25. Notice the variable for the index counter is decremented. When an element is removed from an ArrayList, the remaining elements shift down an index position. You don't want to skip any values, so you need to check the new value in the current index position.

```
for(int i = 0; i < intList.size(); i++)
{
   if(intList.get(i) > 25)
   {
      intList.remove(i);
      i--;
   }
}
```

In the next 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);
}
```

The loop iterates for a count of 10. Each time a new random number with a value of 100 to 199 is generated. Using the current size of the ArrayList, a random position is picked. The new random number is added to the randomly-picked index position. The size of the list grows by one each pass through the loop.

Note: The demo program can be modified to work with an `ArrayList` of `Doubles` instead of `Integers`. Give it a try by saving a copy of the integer program and convert to doubles.

Print