**Number List**

Normal arrays are objects, and ***immutable*** – their size can't be changed once instantiated. The java.util.ArrayList class is a data structure that implements a ***mutable*** list, one that can grow and shrink when necessary.

So how does an ArrayList work, given arrays are immutable? In this lab, you will implement a class **NumberList.java**, which is essentially an ArrayList<Integer> (a mutable list that contains Integer objects). This lab will serve as your introduction to data structures for those going to CS 3 next year!

**The NumberList class**

This class will have the following methods (which should function the same as ArrayList<Integer> methods).

**Read on for instructions, the following are just descriptions of the methods!**

**For now, use whatever IDE you are comfortable with. I will introduce Eclipse in the next lab.**

|  |  |
| --- | --- |
| NumberList() | Construct an empty NumberList with an initial *capacity* of 2. |
| int size() | Returns the number of elements in this list (NOT the length of the backing array). |
| boolean isEmpty() | Returns true iff (if and only if) this list contains no elements. |
| String toString() | Returns a string representation of this number list in a specified format. |
| void add(int index, Integer val) | Inserts the specified element at the specified position in this number list. |
| boolean add(Integer element) | Appends the specified element to the ***end*** ofthisnumber list. Returns true (check the **FAQ** if you're not sure why). |
| Integer get(int index) | Returns the element at the specified position in this number list. |
| Integer set(int index, Integer val) | Replaces the element at the specified position in this number list with the specified element. Returns the replaced element. |
| Integer remove(int index) | Removes the element at the specified position in the list. |

**Implement the NumberList class as follows:**

1. Write the class heading along with an empty class body. NumberList does not extend another class.
2. Add the two private instance variables for NumberList:
   * private Integer[] list - the "backing" array that will store the list's Integer objects.
   * private int size - A NumberList also needs to remember the list's size.
     + **Note:** size **(number of elements currently in the list) and** list.length **(the *capacity* of the backing array) are NOT the same thing!**
     + size will be maintained dynamically (as elements are added / removed from the list).

**Do not initialize these variables in their declaration statements (this should be done in the constructor(s)).**

1. Implement the zero-parameter constructor. The array should have a default capacity (length) of two. **Note:** A NumberList object begins empty – your size variable should be initialized accordingly.
2. Write the size accessor (getter) method. This method should NOT be counting the number of non-nulls in the backing array.
3. Write the isEmpty method which returns true iff (if and only if) the list is empty (has no elements). Test this method in a Runner class, with an empty list. It should return true.
4. Write the toString method, which returns a String representation of the number list. The String representation consists of the NumberList's elements in order, enclosed in square brackets ("[]"). Adjacent elements are separated by the characters ", " (comma and space). Some examples are below.

|  |  |
| --- | --- |
| **The number list elements** | toString **return value** |
| empty list | "[]" |
| 5 | "[5]" |
| 7 -3 | "[7, -3]" |
| 0 15 100 | "[0, 15, 100]" |

When iterating over the backing array, use the size instance variable (rather than checking for non-null elements). There is no need to traverse the entire array when you know exactly how many elements are currently in the list.

1. The "backing" array often contains extra "slots" that are not being used to store Integer objects. For example, when a NumberList object is initially created, the array has length 2, but there are no Integer objects stored in it. We say that it has ***capacity*** of 2 and ***size*** of 0.  
     
   The first two numbers that you add to a number list will use this excess capacity. After these adds, the size will have increased to 2, but the capacity will still be 2. A third add would require additional capacity. We will increase the capacity of a number list with the method private void doubleCapacity().  
     
   Implement the doubleCapacity method. It must do the following:
   * Create a new Integer array that is twice the length of your original array (list).
   * Copy all the elements of your original array into the corresponding elements of the new array.
   * Change the value of your array instance variable to reference the new array.
2. Now implement the **two-parameter** add method which inserts an Integer object into your list. Write your method as follows:
   * It's legal to insert at the beginning of your list (index == 0), at the end of the list object (index == size), or anywhere in between. If index is not in this range, this method throws an IndexOutOfBoundsException. Do this by adding the following statement:

throw new IndexOutOfBoundsException();

* + If you have no excess capacity in your list, you need to increase it. Call your doubleCapacity helper method if necessary.
  + Move all the elements in the array with an index greater than or equal to the given index to "make room" for the new given element. Your loop should be using the current size of the list, NOT the *capacity* of the backing array.
  + Store the element in the array.
  + Increment the instance variable that you are using to keep track of the size.

Compile your code. Make sure that all the methods you have written so far work for empty number lists as well as lists with one, two, and three elements. Comment out calls to methods you haven't written.

1. Implement and test the **one-parameter** add method that will "append" an Integer to the end of the list. Take advantage of a method you have already written. The body of this add method should consist of exactly two statements (one of which should be return true). If you're curious why the add method returns true, check the ArrayList powerpoint.
2. Implement and test the get method, which returns the element at the given index. If the given index is invalid, get must throw a new IndexOutOfBoundsException. Again, whether or not an index is invalid depends on the size of the list, not the capacity of the backing array.
3. Implement and test the set method which replaces the Integer object at the given index with the given element. It then returns the replaced object. If the given index is invalid, it must throw a new IndexOutOfBoundsException.
4. Implement and test the remove method, which removes and returns the element at the given index. Don't forget to decrement the size, "slide down" the remaining elements, or to throw an IndexOutOfBoundsException when appropriate.
5. Test your NumberList class by running the main method in the **NumberListRunner.java** class (provided, copy/paste or import into your project). Done properly, you should see a "Pass: ..." output on the console from each test method. If not, look at the test that failed, look at what the test code did, and figure out what went wrong. Oh, and fix it.

***Number List*** *lab by Robert Glen Martin, Dallas School for the Talented and Gifted*

*(with modifications by me)*

**(Advanced) Use your NumberList**

*Problems tagged (Advanced) are completely optional! These problems are interesting and challenging problems that are* ***not required*** *as part of the project.*

Using objects of your NumberList class (essentially an ArrayList<Integer> object), solve the following problems:

1. The prime number 41 can be written as the sum of six ***consecutive*** primes: 2 + 3 + 5 + 7 + 11 + 13.

This is the longest sum of consecutive primes that adds to a prime below one hundred.

The longest sum of consecutive primes below one thousand that adds to a prime contains 21 terms, the sum of which is equal to 953.

**Which prime, below one million, can be written as the sum of the most consecutive primes? Note:** the "chain" of primes does not have to start with 2 (the first prime).

1. A perfect number is a number for which the sum of its proper divisors is exactly equal to the number. For example, the sum of the proper divisors of 28 would be 1 + 2 + 4 + 7 + 14 == 28, which means that 28 is a perfect number.

A number n is called "deficient" if the sum of its proper divisors is less than n and it is called "abundant" if this sum exceeds n.

As 12 is the smallest abundant number (1 + 2 + 3 + 4 + 6 == 16) the smallest number that can be written as the sum of two abundant numbers is 24.

By mathematical analysis, it can be shown that all integers greater than 28123 can be written as the sum of two abundant numbers. However, this upper limit cannot be reduced any further by analysis even though it is known that the greatest number that cannot be expressed as the sum of two abundant numbers is less than this limit.

**Find the sum of all the positive integers which cannot be written as the sum of two abundant numbers.**