Developing a Simple Abstract Data Type (ADT)

For these tasks, you will need to make use of your knowledge gathered so far in Java Programming on classes, arrays, loops and similar structures.
We will attempt to develop our first array class, a class that will contain an array to store information in and operations to make using this array as convenient to the user as possible. In other words, we will attempt to gather in a single container, everything that we will need to use arrays efficiently.

These are some of the questions that you need to think about when implementing the Array Abstract Data Type class (Array ADT class):

1. How many files will you need? Why?
2. In the data class (the ArrayADT class), what will you choose for instance variables?
3. How will you create an array-class object?
4. How about creating another one of these objects but with its array already fully populated?

Use the tasks below to answer these questions.

Tasks

Have a look at the screenshot at the end of this document if you need help to get started with these first tasks…

a) Create the ArrayADT class and declare its instance variables: data (an array to store the values), numberOfEntries (an integer to keep track of how many elements are already in the array), last (to keep track of the index of the last item added to the array), and capacity (the size of the array);
b) Provide your class with a constructor that will take a parameter size that will fix the size of the array created therein. Initialise the remaining instance variables (data members) appropriately.
c) Create another constructor that will take an already populated array (i.e. data type int[]) as a parameter; assign this to the *data* array in your class.

What sort of methods would we normally add to such a data class? What other operations could be useful to manipulate our data-structure? Add the following methods:

1) string add(int entry) add a new entry in the array (in the last available position). If the array is full, return a message saying that the array is full, otherwise add the item and return a message saying that the item has been correctly added.
2) boolean remove(position) Remove a component at a given position in the array (replace it with zero). Return true if successful, false if the position is not in the array (e.g. the position is greater than the length of the array)
3) int get(int index) Return the component at a given index in the array. Return -999 if the index is not valid (e.g. index is greater than the length of the array)
4) boolean isEmpty() Determine whether the array is empty (return true) or not (return false).
5) boolean isFull() Determine whether the array is full (return true) or not (return false).
6) int numberOfElements() Determine the number of components in the array
7) void empty() Destroy the array by setting data=null;

Testing (DriverClass)

How are we now going to use or test our new class?
What do we need to do in order to test each method already written?
What are we going to do to make sure that what we think is happening, has actually happened?

8) Create a DriverClass (i.e., the main class) to test your methods. Call the operations you implemented above and each time, to test that operation, use the print operation to reflect the change in your array ADT.

Dynamic ADT

Complete your ADT by adding the following operations\methods:

9) public void increaseSize(int newSize)
   A method that increases the array size by copying the old array into the new array of size *newSize*. *Hint: you could use Arrays.copyOf(data,newSize) to speed up the process-*– after importing java.util.Arrays*.*

10) public void showArraySummary()
   A method that prints the array size, the number of items in the array and the content of the array into a string and return the string (you can use Arrays.toString(data) – after importing java.util.Arrays).

Testing (DriverClass)

11) Use the driver class to test the three new methods.
Each time, to test that operation, use the print operation to reflect the change in your array ADT.

Challenges with Dynamic Arrays:

Implement the methods below:

12)newEntry(int entry, int index) Insert a component *entry* at a given position *index* in the array. If the array is full, use the function you created before to increase the size before adding the new element.

13)removeEntry(int entry) Remove a component at a given position in the array, making sure the array remains "packed" (with no gaps between elements)

Ask you tutor if you don't remember what "packed" means. You should also have a look at the lecture's slides (Dynamic Arrays) if you don't remember how to get started. The slides report part of the Java code you'll need for these tasks.

Answer to Tasks a and b

```
public class ArrayADT {
    //instance variables
    private int[] data;
    private int numberOfEntries;
    private int last;
    private int capacity;

    //constructor
    public ArrayADT(int size){
        this.data = new int[size];
        numberOfEntries = 0;
        last = 0;
        capacity = size;
    }

}
```

Use this example to help you getting started with today's tasks.

The *numberOfEntries* instance variable will help you in keeping track of how many elements are in your array, which is different from the array size (*capacity*).
The variable *last* will keep track of the index of the last added item.

Don't forget to update these variables every time you add or remove an item from the array.

Answer to Task c

In the ArrayADT class:

```
public ArrayADT(int[] array) {
    data = array;
    this.numberOfEntries = array.length;
}
```

In the Driver\main Class:

```
ArrayADT existingADT = new ArrayADT(new int[] { 1, 5, 9, 2, 3 });
```