**Array Capacity VS Length**

If somebody asks you how long the DVD Array is, what what would your answer be?

There are two different answers you might have given.

1. The number of DVDs the box could hold, if it was full, or
2. The number of DVDs currently in the box.

Both answers are correct, and both have very different meanings! It's important to understand the difference between them, and use them correctly. We call the first one the **capacity** of the Array, and the second one the **length** of the Array.

Array Capacity

Let's say we've created a new Array like this.

DVD[] array = new DVD[6]

Is it a valid operation to insert an element at array[6]? What about at array[10]?

Nope, neither of these are valid. When we created the Array, we specified that it can hold up to 6 DVD's. This is the Array's **capacity**.

Remembering that indexing starts at 0, we can only insert items at array[0], array[1], array[2], array[3], array[4], and array[5]. Trying to put an element anywhere else, such as array[-3], array[6], or array[100] will cause your code to *crash* with an ArrayIndexOutOfBoundsException!

The Array's capacity must be decided when the Array is created. *The capacity cannot be changed later*. Going back to our DVD's-in-a-cardboard-box-analogy, changing the capacity of an Array would be akin to trying to make a cardboard box bigger. Trying to make a fixed-size cardboard box bigger is impractical, and it's the same as an Array on a computer!

So, what do we do if we get a 7th DVD and we'd like all our DVD's in the same Array? Well, unfortunately it's the same as it is with our cardboard box. We'll need to go get a larger one, and then move all the existing DVD's into it, along with the new one.

The **capacity** of an Array in Java can be checked by looking at the value of its length attribute. This is done using the code arr.length, where arr is the name of the Array. Different programming languages have different ways of checking the length of an Array.

int capacity = array.length;

System.out.println("The Array has a capacity of " + capacity);

Running this code will give the following output:

The Array has a capacity of 6

Yup, it's a bit confusing that you need to access the capacity of an Array by using .length. Unfortunately, this is just something you'll need to get used to.

Array Length

The other definition of **length** is the number of DVDs, or other items, currently in the Array. This is something you'll need to keep track of yourself, and you won't get any errors if you overwrite an existing DVD, or if you leave a gap in the Array.

You might have noticed that we've been using a length variable in our previous examples, to keep track of the next empty index.

|  |
| --- |
| // Create a new array with a capacity of 6.  int[] array = new int[6];  // Current length is 0, because it has 0 elements.  int length = 0;  // Add 3 items into it.  for (int i = 0; i < 3; i++) {  array[i] = i \* i;  // Each time we add an element, the length goes up by one.  length++;  }  System.out.println("The Array has a capacity of " + array.length);  System.out.println("The Array has a length of " + length); |

Running this code will give the following output:

The Array has a capacity of 6

The Array has a length of 3

Handling Array Parameters

Most Array questions on LeetCode have an Array passed in as a parameter, with no "length" or "capacity" parameter. What do we mean by this? Well, let's look at an example. Here is the description for the first problem you'll be asked to solve.

Given a binary array, find the maximum number of consecutive 1s in this array.

And here is the code template you're given.

The only parameter is nums; an Array. You couldn't possibly solve this question without knowing how long nums is. Well, luckily it's straightforward. When an Array is given as a parameter, without any additional information, you can safely assume that **length == capacity**. That is, the Array is the exact right size to hold all of it's data. We can, therefore, use .length.

Be careful though, Array's are 0-indexed. The capacity/ length is a number of items, *not a highest index*. The highest index is .length - 1. Therefore, to iterate over all items in the Array, we can do the following.

|  |
| --- |
| class Solution {  public int findMaxConsecutiveOnes(int[] nums) {  // Hint: Initialise and declare a variable here to  // keep track of how many 1's you've seen in a row.  for (int i = 0; i < nums.length; i++) {  // Do something with element nums[i].  }  }  } |

And that is the basics of Arrays that you'll need to get started! In the next chapter, we'll look at some of the fundamental techniques we use to work with Arrays.

Before that though, we have a few introductory Array problems for you to play around with, starting with the one we briefly looked at above. Enjoy!