**Main Concept: One Dimensional Arrays**

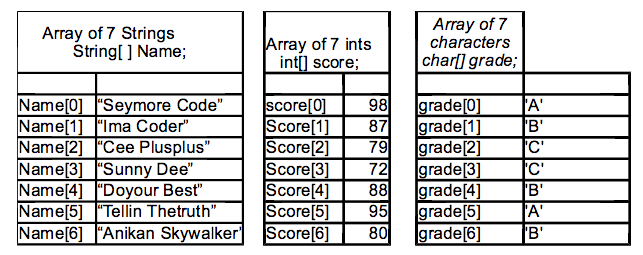
An Array is a list in Java (or any other programming language). The syntax to specify that a variable (or object) is an array is denoted by brackets [ ].

For example*: int number* is a declaration of a variable, *number*, that will store an integer value.

But *int[ ] number* is a declaration of a variable, *number*, that will store a whole bunch of integer values.

**String** ***name*** is a declaration of an object, name, that will store a single String value.

**String[ ]** ***name*** is a declaration of an object, name, that will store many String objects!



Each item in an array is an ***element*** of the array. An elements position in the array is called the ***index***. For example, the number 80 is the ***element*** of the score array at ***index*** 6, even though it is really the 7th item in the list.

You can't access an item in array that is out of the bounds of the array. This can be a little tricky in java as, if you are using arrays like this you have to **know ahead of time how many elements your array will have** (or how many it can have). It also arrays are not intended for lists that will be changing sizes, or lists where you want to insert items. Later I will show you a way of using the ArrayList class, which will help with arrays where you don't know ahead of time how many elements are in the array, or you want to easily add/delete items from an array. But for now we are just going to get the basics down.

**Main Concept: Declaring an Array**

In general, an array is declared as follows:

<variable type>[ ] <variable name> = new <variable type>[MAX\_NUM]

So, for example, this is how you would declare an array used to store information as mentioned above:

<variable type>[ ] <variable name> = new <variable type>[MAX\_NUM]

String[ ] Nam = new String[ 20];

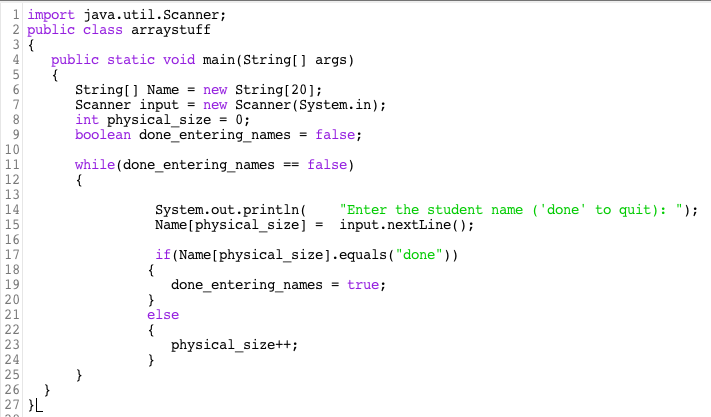
This will actually allocate 20 spots for student names to be stored – so you will have to resize later when you know how many students there are. **It is always better to overshoot the max amount than to undershoot it.**

Right now the array is not full, let’s say we only have 7 items stored in the array (as defined above).

**The physical size** of this array is 20, but the **logical size** is 7. You will junk (whatever happens to be in the particular memory spot) if you try to access StudentName[19]. You will crash your whole program (out of bounds error) if you try to access StudentName[21]. Remember to use the out of bounds errors exceptions in your code.

**Main Concept: Traversing An Array**

When reading in (obtaining) values for an array **for loops** are a great bet. For example:



**For Each Element, Do Something**

Arrays greatly expand your programming options because you are no longer limited to variables that can hold only one piece of information at a time. Since all of the elements are stored in consecutive memory locations, the array can be treated efficiently as a single unit.

Loops are indispensable tools for handling arrays because algorithms can be used to process the entire data structure. For example, finding the sum of 100 integers. Essentially, what we want to say when adding the elements in an array is "for each integer in the array, add it to the sum." The for loop is very handy for this task, but it can be improved upon. In programming (as you are discovering), it is often true that "for each" solution, there is often an even simpler solution! The for-each loop will be our next focus.

for(int index = 0; index < 10; index++)  
{  
     System.out.println(names[index]);  
}

The block of code inside the opening and closing curly braces is executed until the terminal condition is encountered. In this example, the loop counts from 0 to 9 and prints the information contained at the position in the **names** array indicated by the **index** variable. This **for** loop couldn't be any simpler, unless there was a shortcut.

The traditional **for** loop can very efficiently iterate through the index positions of an array; however, Java introduced an even more specialized **for** loop, specifically to deal with indexed data collections like arrays. This new loop is referred to as an enhanced for loop, or simply a **for-each** loop. The structure and syntax of a **for-each** loop is shown below.

for(String person : names)  
{  
     System.out.println(person);  
}

What happened to the index initialization, condition test, and incrementation? The strategy behind the **for-each** loop is that the index always starts at zero and increments by one, and that all elements in the array will be processed. The IMACS lesson will fully explain the details of the **for-each** loop; it is very useful, but there will still be certain situations in which the traditional **for** loop must be used with arrays.