**Arrays**

Declaring Arrays

* An *array* is a simple data structure to hold a series of data elements of the same type.
* Declare an array variable in one of two ways:
  + With [] after the variable type: int[] values;
  + With [] after the variable name: int values[];
* Arrays can be single- or multi-dimensional.
  + A two dimensional array could be declared with: double values[][];
* Array elements are integer indexed.
  + Use *arrayName*.length to get the array length.
  + Elements are indexed from 0 to *arrayName*.length - 1
  + Access individual elements with *arrayName*[*index*]

Examples of array declarations:

String[] args; // single-dimensional array of String objects

int[] numbers; // single-dimensional array of ints

byte[] buffer; // single-dimensional array of bytes

short[][] shorts; // double-dimensional array of shorts

To get the array length:

int arrayLength = myNums.length; // 5

int nameLength = name.length; // 3

int bufLength = buf[2].length; // 1024

Creating and Initializing Array Objects

* Arrays are implemented as objects.
* You can use the new operator to create an array of an indicated type and length, as in:

int[] values = new int[10];

* + Elements are initialized automatically to type-appropriate default values.
* Alternatively, you can use an *array initializer* to create an array with a set of initial values.
  + The array initializer consists of a sequence of comma-separated expressions of an appropriate type, enclosed within braces. For example:

int[] values = {5, 4, 3, 2, 1};

* + The array object is sized automatically based on the number of initial values.

Examples of array initializations:

int[] myNums = { 1, 3, 5, 7, 9 }; // 5 elements

char[] name = new char[3];

name[0] = 'T';

name[1] = 'o';

name[2] = 'm';

short[][] matrix = { {1,4,7}, {5,6,3}, {2,8,9} };

byte[][] bufs = new byte[10][1024];

To access an element of an array:

int myNum = myNums[3]; // 7

char firstLetter = name[0]; // 'T'

matrix[1][2] = 0; // used to be 3

3.6. Modifying Array Size

* Once created, the size of an array cannot change.
  + If you need to grow an array, you must create a larger array object of the same type, and then copy the elements from the old array to the new array.
  + The System.arraycopy() method is an efficient way to copy the existing elements to the new array. For example:
  + int[] values = {5, 4, 3, 2, 1}; // A 5-element int array
  + int[] newValues = new int[10]; // A 10-element int array
  + // Copy all elements from values to newValues
  + System.arraycopy(values, 0, newValues, 0, values.length);
  + // Assign the array back to values

values = newValues;

In the example above, the values and newValues variables do not actually contain the arrays, they*refer* to the array objects. We’ll examine this concept more in the [Object Oriented](https://thenewcircle.com/bookshelf/java_fundamentals_tutorial/object_oriented.html) module.

3.7. Strings

**Strings**

* Strings are objects.
  + In Java, strings are instances of the String class.
  + Unlike C/C++, you can’t treat a String directly as an array of chars.
  + There are methods that allow you to create a String from an array of chars, and vice versa.
* Enclosing a sequence of literal characters within double quotes automatically creates a Stringobject: "this is my string"
* Strings are *immutable*
  + Once created, a String cannot be modified.
  + But a String has methods to perform various transformations, each returning a newString object.

Immutable means you can't not change the value of the same reference. Every time you required to create new reference means new memory location.

ex:

String str="abc";

str="bcd";

Here, in the above code, in the memory there are 2 blocks for storing the value. The first for value "abc" and second for "bcd". The second value is not replace to first value.

This is call the immutable.

* The String class has many methods, including: length, replace, substring, indexOf, equals,trim, split, toUpperCase, endsWith, etc.