An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together.

An array is a data structure, which can store a fixed-size collection of elements of the same data type.

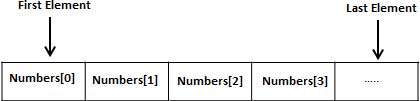
An array is an aggregate data structure that is designed to store a group of objects of the same or different types. Arrays can hold primitives as well as references. The array is the most efficient data structure for storing and accessing a sequence of objects.

An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed. You have seen an example of arrays already, in the main method of the "Hello World!" application. This section discusses arrays in greater detail.



Instead of declaring individual variables, such as number1, number2, ..., number99, you just declare one array variable **number** of integer type and use number1[0], number1[1], and ..., number1[99] to represent individual variables. Here, 0, 1, 2, .....99 are **index** associated with **var** variable and they are being used to represent individual elements available in the array.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −

int var = number[9];

The above statement will take the 10th element from the array and assign the value to **var** variable.

Here is the list of most important array features you must know (i.e. be able to program)

* copying and cloning
* insertion and deletion
* searching and sorting

An array is not a primitive data type - it has a field (and only one), called *length*. Formally speaking, an array is a reference type.

(java Specific)

The *java.util.ArrayList* class supports an idea of a *dynamic* array - an array that grows and shrinks on demand to accomodate the number of elements in the array. Below is a list of commonly used methods

* *add(object)* - adds to the end
* *add(index, object)* - inserts at the index
* *set(index, object)* - replaces at the index
* *get(index)* - returns the element at that index
* *remove(index)* - deletes the element at that index
* *size()* - returns the number of elements

**Multi-dimensional arrays**

In many practical application there is a need to use two- or multi-dimensional arrays. A two-dimensional array can be thought of as a table of rows and columns. This creates a table of 2 rows and 4 columns:

int[][] ar1 = new int[2][4];

You can create and initialize an array by using nested curcly braces. For example, this creates a table of 3 rows and 2 columns:

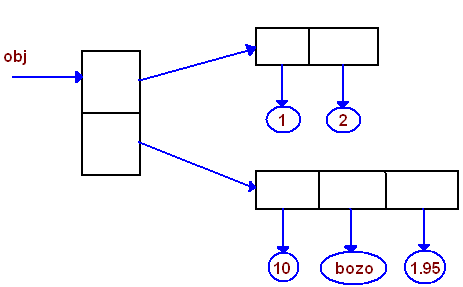
int[][] ar2 = {{1,2},{3,4},{5,6}};

Generally speaking, a two-dimensional array is not exactly a table - each row in such array can have a different length. Consider this code fragment

Object[][] obj = {{new Integer(1),new Integer(2)},

{new Integer(10), "bozo", new Double(1.95)}};

The accompanying picture sheds a bit of light on internal representation



## **Disadvantages of Arrays**

* The number of elements to be stored in an array should be known in advance.
* An array is a static structure (which means the array is of fixed size). Once declared the size of the array cannot be modified. The memory which is allocated to it cannot be increased or decreased.
* Insertion and deletion are quite difficult in an array as the elements are stored in consecutive memory locations and the shifting operation is costly.
* Allocating more memory than the requirement leads to wastage of memory space and less allocation of memory also leads to a problem.
* Complexity of the insertion or deletion: log(n)