Arrays

There are times when it is convenient to store a number of data items in memory to be used (and reused) in the computations of an application. The data items may be input from a file or they may be created by a program.

One-dimensional Arrays

One-dimensional arrays give a variable name to a number of data elements. We number the data elements from 0 to one less than the number of elements in the array; each of these numbers is an index, sometimes called a subscript, of the array. Each data element must have the same type.

Array variables are used to store and access data elements one at a time. Individual elements are referred to by their subscript, or index, number.

Declaration, allocation and initialization of one-dimensional arrays:

The declaration of an array is indicated by putting square brackets after the type. The declaration gives the name of the array and the type of the elements.

<type> [ ] <arrayname> ;

In addition, the array must be allocated, which means that you must say how many elements there will be. This can be done on the same line, or separately. Here it is shown on the same line.

<type> [ ] <arrayname> = new <type> [<numberofelements>];

Examples:

int [ ] scores = new int[25]; (25 elements in the array scores)
double [ ] prices = new double[50];
String [ ] studentnames = new String[12];

An array could also be initialized to hold a number of values. (This is usually useful only for small arrays.) The initialization will take the place of allocation because the number of elements is obvious from the initialization list.

<type> [ ] <arrayname> = <initialization list>

Example:

int [ ] codes = { 4011, 4080, 4519, 4708}

This array will be allocated with 4 elements.

Array indexing; Setting and using elements of an array:

The array elements are numbered starting from 0 and going to 1 less than the number of elements. For example, if the array has 10 elements, the subscript numbers range from 0 to 9.

Individual data elements are referred to by putting a valid subscript number in square brackets:

The fourth element of the prices array:

```
prices [ 3 ]
```

This can be used in expressions or in assignment like a variable.

Assigning to the tenth element of the scores array:

```
scores [ 9 ] = 59
```

A standard for loop to use each element of an array would have the form:

```
String [ ] students = new String[25]

for (int index = 0; index < 25; index++) {
// do whatever processing needs to be done to each element students[index] here
        labelstring = labelstring + students [ index ] + " "
}
```

For each array, there is also a property called "length" that will give the number of elements in the array. For example, the number of elements in the students array would be given by "students.length". Therefore, the standard array for loop is quite often given as:

```
for (int index = 0; index < students.length; index++) {
// do whatever processing needs to be done to each element students[index] here
        labelstring = labelstring + students [ index ] + " ";
}
```

You should not try to subscript an array element that has not been defined (this is also true for regular variables) and you should not try to use subscript numbers that are not between 0 and the number of elements -1. If this happens, the system will give an "Array Index Out of Bounds" error.

Note: There are variations on the array declaration syntax not covered here. The Java Tutorial page on arrays is at
http://download.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Arrays as Parameters and Result Values from methods

Individual array values can be passed as parameters to methods and returned as results in the same way as any other individual value. For example, if we have a method to find the maximum of two integers, it's method header could look like:

```
private int max ( int x, int y)
```

Then if we also have an array declared:

```
int [ ] scores = new int[25];
```

And if that array has values defined for scores[4] and scores[5], which will be of type int, we can pass those as values to a method:

```
int result = max ( scores[4], scores[5] )
```

But *entire arrays* can also be passed as values to a method and returned as a result. In this case, the type of the array is given, for example, as

int [ ] , for an array of integers (of any size)
double [ ] , for an array of doubles (of any size)
String [ ] , for an array of Strings (of any size)

Then we can use this to define methods. For example, if we want to write a method to find the average of the values of an array of type int, we could write the method header:

```
private double getAverage (int [ ] numbers )
```

And to write a method that calls the random number generator and returns an array of random numbers of a particular size, we could write the method header:

```
private int [] getRandomNumbers (int sizearray)
```

Two-dimensional Arrays (and beyond)

For each additional dimension, add an additional subscript brackets to the declaration and use an additional subscript number to access elements.

Declaration of a two-dimensional array:

The dimensions of a two-dimensional array can be thought of as the row and column numbers of a matrix.

<type> [ ] [ ] <arrayname> = new <type> [<number of rows>] [<number of columns>]

As with one-dimensional arrays, the valid subscripts range between 0 and one less than the number of elements in each dimension.
Example:

```
double [ ] [ ] prices = new double [25] [10];
```

This is an array with 25 rows and 10 columns.

Setting and using elements of 2-D arrays:

Individual data elements are referred to by putting a two valid subscript numbers in parentheses:

The element in the row numbered 9 and the column numbered 3 would be:

```
prices [ 9, 3 ]
```

Additional dimensions would add additional subscript numbers within the parentheses and separated by commas.

Accessing each element of a two-dimensional array can be achieved by a double loop, for example:

```
for (int row = 0; row < 25; row++) {
        for (int col = 0; col < 10; col++) {
                ‘ do processing of array element here
                prices [ row ] [ col ] =  prices [ row ] [ col ] + tax;
        }
}
```