JAVA ARRAYS

Variables hold only one value. The integer variables you have set up have held only one number. However, more than one value can be held at a time using arrays. It is like a list of items. Think of an array as the columns in a spreadsheet. You can have a speadsheet with only one column, or lots of columns. The data held in a single-list array might look like this:

Id	Array_values
0	24
1	35
2	40
3	16

Like a spreadsheet, arrays have a position number for each row. The positions in an array start at 0 and go up sequentially. Each position in the array can then hold a value.

To set up an array of numbers like in the image above, you have to tell Java what kind of data is going in to your array (integers, string, boolean values, etc.).

You then need to say how many positions the array has. You set them up like this:

```
int [] aryNums;
```

The only difference between setting up a normal integer variable and an array is a pair of square brackets after the data type. The square brackets are enough to tell Java that you want to set up an array. The name of the array above is aryNums. Just like normal variables, you can call them almost anything you like.

Although the above procedures do not say how many positions the array should hold; to do that you have to set up a new array object.

```
aryNums= new int [6];
```

You start with your array name, followed by the equals sign. After the equals sign, you need the Java keyword new, and then your data type again. After the data type comes, a pair of square brackets. In between the square brackets you need the size of the array. The size is how many positions the array should hold.

So we've just told Java to set up an array with 6 positions in it. After this line is executed, Java will assign default values for the array. Because we've set up an integer array, the default values for all 6 positions will be zero (0).

To assign values to the various positions in an array, you do if in the normal way:

```
aryNums [0] = 24;
```

Here, a value of 24 is being assigned to position O in the array called aryNums. Again, the square brackets are used to refer to each position. If you want to assign a value of 35 to array position 1, the code would be this:

```
aryNums [1] = 35;
```

And to assign a value of 40 to array position 2, it's this:

```
aryNums [2] = 40;
```

Don't forget, because arrays start at 0, the third position in an array has the index number 2.

If you know what values you are going to be in the array, you can set them up like this instead:

```
int [] aryNums = {24, 35, 40, 16}
```

This method of setting up an array uses curly brackets after equals sign. In between, the curly brackets, you type out the values that the array will hold. The first value will then be position 0, the second value position 1, and so on. Note that you still need the square brackets after int, but not the new keyword, or the repetition of the data type and square brackets. But this is just for data types of int values, string, and char values, otherwise, you need the new keyword. So you can do this:

```
String [] aryStrings = {“Autumn”, “Spring”, “Summer”, “Winter”};
```

Arrays and Loops

Arrays come in their own with loops, exemplified in their assignment:

```
john [0] = “Emeka”;
```

But that is not terribly practical if you have to assign many values to an array. For example: imagine a lottery program that has to assign the numbers 1 to 49 to positions in an array. Instead of typing a long list of array positions and values; you can use a loop. Here is some code that does just that:

```
int [] lottery_numbers = new int [49];
int i;
for (i=0; i<lottery_numbers.length; i++) {
      lottery_numbers [i] = i+1;
      System.out.println (lottery_numbers [i]);
}
```

So we set up an array to hold 49 integer values, then comes the loop code. Notice the end condition of the loop:

```
i < lottery_numbers.length;
```

Length is a property of array objects that you can use to get the size of the array (how many position it has). So this loop will keep going round and round while the value in the variable i is less than the size of the array.

To assign values to each position in the array, we use this line:

```
lottery_numbers [i] = i + 1;
```

Instead of a hard code value between the square brackets of the array name, we have the variable called i. You can also use loops to display or access, the several contents in an array.

Sorting Arrays

Some inbuilt java methods allow us to sort our arrays. To use any of them we need to reference the Java library called Arrays, from `util` like this:

```
import java.util.Arrays;
```

You can use the sort method quite easily. For example:

```
Arrays.sort (john);
```

The above code sorts in ascending order and not in descending order.

Arrays and Strings

We can place strings of text into arrays. This is done the same way as for integers:

```
String [] people = new String [2];
people [0]= “Young”;
people [1]= “Ogada”;
```

We can perform a sort on string arrays, just like we can with integers. But the sort is an alphabetical ascending one, meaning that ‘aa’ will come before ‘ab’. However, Java uses Unicode characters to compare one letter in your string to another. This means that uppercase letter will come before lowercase letters.

Multi-Dimensional Arrays in Java

The arrays we have been using so far have only held one column of data. We can also set up an array to hold more than one column. These are called multidimensional arrays. As an example, think of a spreadsheet with rows and columns. If you have 3 rows and 5 columns then your spreadsheet can hold 15 numbers. It looks like this:

	A	B	C	D	E
0	14	27	20	40	35
1	37	18	29	32	30
2	19	25	30	34	39

A multidimensional array is one that can hold all the values above. You set them up like this:

```
int [] aryNums= new int [3] [5];
```

They are set up in the same way as a normal array except you have two sets of square brackets. The first set of square brackets is for the rows and the second set of square brackets is for the columns.

```
aryNums [0] [1] = 27;
```

For example:

```
public static void main (String [] args) {
   int [] aryNumbers = new int [6] [5];
   aryNumbers [0] [0] = 10;
   aryNumbers [0] [1]= 12;
   aryNumbers [0] [2] = 43;
   aryNumbers [0] [3] = 11;
   aryNumbers [0] [4] = 22;
   aryNumbers [1] [0] = 20;
   aryNumbers [1] [1] = 45;
   aryNumbers [1] [2] = 56;
   aryNumbers [1] [3] = 1;
   aryNumbers [1] [4] = 33;
   aryNumbers [2] [0] = 30;
   aryNumbers [2] [1] = 67;
   aryNumbers [2] [2] = 32;
   aryNumbers [2] [3] = 14;
   aryNumbers [2] [4] = 44;
   aryNumbers [3] [0] = 40;
   aryNumbers [3] [1] = 12;
   aryNumbers [3] [2] = 87;
   aryNumbers [3] [3] = 14;
   aryNumbers [3] [4] = 55;
   aryNumbers [4] [0] = 50;
   aryNumbers [4] [1] = 86;
   aryNumbers [4] [2] = 66;
```

```
    aryNumbers [4] [3] = 13;
    aryNumbers [4] [4] = 66;
    aryNumbers [5] [0] = 60;
    aryNumbers [5] [1] = 53;
    aryNumbers [5] [2] = 44;
    aryNumbers [5] [3] = 12;
    aryNumbers [5] [4] = 11;
    int rows = 6; int columns = 5; int i, j;
    for (i=0; i<rows; i++) {
      for (j=0; j<coulms; j++) {
           System.out.print (aryNums [i] [j]+"
");
      }
      System.out.println (" ");
    }
}
```

To access all the items in a multidimensional array we use one loop inside of another–a double for loop.

The first for loop is used for the rows; the second is used for the columns. The first time round the first loop, the value of the variable i will be 0. The code inside of the loop is another loop. The whole of the second loop will be executed while the value of the variable i is 0. The second for loop use a variable called

j. The i and the j variables can then be used to access the complex array.

Array Lists

If you don't know how many items are going to be held in your array, you may be better off using something called an `ArrayList`. An ArrayList is a dynamic data structure, meaning items can be added and removed from the list. A normal array in Java is a static data structure, because you stuck with the initial size of the array.

To set up an ArrayList, you first have to import the package from the `java.util` library:

```
import java.util.ArrayList;
```

You can then create a new ArrayList object:

```
ArrayList listTest = new ArrayList ();
```

Notice that you don't need any square brackets this time.

Once you have a new ArrayList objects, you can add elements to it with the add method:

```
listTest.add (“first item”);
listTest.add (“second item”);
listTest.add (“third item”);
listTest.add (7);
```

In between the round brackets of add you put what it is you want to add to the ArrayList. You can only add objects, however. The first three items we’ve added to the list above are String objects. The fourth item is a number. But this will be a number object of type, integer, rather than the primitive data typed int.

Items in the list can be referenced by an index number, and by using the get method:

```
listTest.get (3);
```

This line will get the item at index position 3 on the list. Index numbers start counting at zero, so this will be the fourth item.

You can also remove items from an ArrayList by using the remove method with either the Index number or the value on the list:

```
listTest.remove (2); or
listTest.remove (“second item”);
```

Removing an item will resize the ArrayList, so you have to be careful when trying to get an item on the list when using its Index number. If we’ve removed item number 2, then our list above will contain only 3 items. Trying to get the item with the Index number 3 would then result in an error.

Iterator

To go through each item in your ArrayList you can set up something called an `Iterator`. This class is found in the `java.util` library:

```
import java.util.iterator;
```

You can then attach your ArrayList to a new Iterator object:

```
Iterator it = listTest.iterator ();
```

This sets up a new Iterator object called “it” that can be used to go through the items in the ArrayList called `listTest`. The reason for using an iterator object is because it has methods called next and `hasNext`. You can use these in a loop:

```
while (it.hasNext) {
     System.out.println (it.next());
}
```

The method `hasNext` returns a Boolean value. The value will be false if there are no more items in the ArrayList. The `next` method can be used to go through all the items in the list.