Multidimensional Array

Source http://www.cafeaulait.org/javatutorial.html

Two Dimensional Arrays

The most common kind of multidimensional array is a two-dimensional array. If you think of a one-dimensional array as a column of values you can think of a two-dimensional array as a table of values like so:

	c0	c1	c2	c3
r0	0	1	2	3
r1	1	2	3	4
r2	2	3	4	5
r3	3	4	5	6
r4	4	5	6	7

Here we have an array with five rows and four columns. It has twenty total elements. However we say it has dimension five by four, not dimension twenty. This array is not the same as a four by five array like this one:

	c0	c1	c2	c3	c4
r0	0	1	2	3	4
r1	1	2	3	4	5
r2	2	3	4	5	6
r3	3	4	5	6	7

We need to use two numbers to identify a position in a two-dimensional array. These are the element's row and column positions. For instance if the above array is called J then J[0][0] is 0, J[0][1] is 1, J[0][2] is 2, J[0][3] is 3, J[1][0] is 1, and so on.

Here's how the elements in a four by five array called M are referred to:

M[0][0]	M[0][1]	M[0][2]	M[0][3]	M[0][4]

M[1][0]	M[1][1]	M[1][2]	M[1][3]	M[1][4]
M[2][0]	M[2][1]	M[2][2]	M[2][3]	M[2][4]
M[3][0]	M[3][1]	M[3][2]	M[3][3]	M[3][4]

Declaring, Allocating and Initializing Two Dimensional Arrays

Two dimensional arrays are declared, allocated and initialized much like one dimensional arrays. However we have to specify two dimensions rather than one, and we typically use two nested `for` loops to fill the array.

The array examples above are filled with the sum of their row and column indices. Here's some code that would create and fill such an array:

```
class FillArray {

  public static void main (String args[]) {
    int[][] M;
    M = new int[4][5];
    for (int row=0; row < 4; row++) {
      for (int col=0; col < 5; col++) {
        M[row][col] = row+col;
      }
    }
  }
}
```

Of course the algorithm you would use to fill the array depends completely on the use to which the array is to be put. Here is a program which calculates the identity matrix for a given dimension. The identity matrix of dimension N is a square matrix which contains ones along the diagonal and zeros in all other positions.

```
class IDMatrix {

  public static void main (String args[]) {
    double[][] ID;
    ID = new double[4][4];
    for (int row=0; row < 4; row++) {
      for (int col=0; col < 4; col++) {
        if (row != col) {
          ID[row][col]=0.0;
        }
        else {
          ID[row][col] = 1.0;
        }
      }
    }
  }
}
```

In two-dimensional arrays ArrayIndexOutOfBounds errors occur whenever you exceed the maximum column index or row index. Unlike two-dimensional C arrays, two-dimensional Java arrays are not just one-dimensional arrays indexed in a funny way.

Exercises

1. Write a program to generate the HTML for the above tables.

Multidimensional Arrays

You don't have to stop with two dimensional arrays. Java lets you have arrays of three, four or more dimensions. However chances are pretty good that if you need more than three dimensions in an array, you're probably using the wrong data structure. Even three dimensional arrays are exceptionally rare outside of scientific and engineering applications.

The syntax for three dimensional arrays is a direct extension of that for two-dimensional arrays. Here's a program that declares, allocates and initializes a three-dimensional array:

```
class Fill3DArray {

  public static void main (String args[]) {
    int[][][] M;
    M = new int[4][5][3];
    for (int row=0; row < 4; row++) {
      for (int col=0; col < 5; col++) {
        for (int ver=0; ver < 3; ver++) {
          M[row][col][ver] = row+col+ver;
        }
      }
    }
  }
}
```

We need three nested `for` loops here to handle the extra dimension.

The syntax for still higher dimensions is similar. Just add another pair pf brackets and another dimension.

Unbalanced Arrays

Like C Java does not have true multidimensional arrays. Java fakes multidimensional arrays using arrays of arrays. This means that it is possible to have *unbalanced arrays*. An unbalanced array is a multidimensional array where the dimension isn't the same for all rows. IN most applications this is a horrible idea and should be avoided.

Searching

One common task is searching an array for a specified value. Sometimes the value may be known in advance. Other times you may want to know the largest or smallest element.

Unless you have some special knowledge of the contents of the array (for instance, that it is sorted) the quickest algorithm for searching an array is straight-forward linear search. Use a `for` loop to look at every element of the array until you find the element you want. Here's a simple method that prints the largest and smallest elements of an array:

```
   static void printLargestAndSmallestElements (int[] n) {
    int max = n[0];
```

```
    int min = n[0];
    for (int i=1; i < n.length; i++) {
      if (max < n[i]) {
        max = n[i];
      }
      if (min > n[i]) {
        min = n[i];
      }
    }
    System.out.println("Maximum: " + max);
    System.out.println("Minimum: " + min);
    return;
  }
```

If you're going to search an array many times, you may want to sort the array, before searching it. We'll discuss sorting algorithms in the next section.

Sorting

All sorting algorithms rely on two fundamental operations, comparison and swapping. Comparison is straight-forward. Swapping is a little more complex. Consider the following problem. We want to swap the value of `a` and `b`. Most people propose something like this as the solution:

```
class Swap1 {

  public static void main(String args[]) {
   int a = 1;
   int b = 2;
   System.out.println("a = "+a);
   System.out.println("b = "+b);
   // swap a and b
   a = b;
   b = a;
   System.out.println("a = "+a);
   System.out.println("b = "+b);
  }

}
```

This produces the following output:

```
a = 1
b = 2
a = 2
b = 2
```

That isn't what you expected! The problem is that we lost track of the value 1 when we put the value of `b` into `a`. To correct this we need to introduce a third variable, `temp`, to hold the original value of `a`.

```
class Swap2 {

  public static void main(String args[]) {
   int a = 1;
```

```
   int b = 2;
   int temp;
   System.out.println("a = "+a);
   System.out.println("b = "+b);
   // swap a and b
   temp = a;
   a = b;
   b = temp;
   System.out.println("a = "+a);
   System.out.println("b = "+b);
  }

}
```

This code produces the output we expect:

```
a = 1
b = 2
a = 2
b = 1
```

Bubble Sort

Now that we've learned how to properly swap the values of two variables, let's proceed to sorting. There are **many** different sorting algorithms. One of the simplest and the most popular algorithms is referred to as *bubble sort*. The idea of bubble sort is to start at the top of the array. We compare each element to the next element. If its greater than that element then we swap the two. We pass through the array as many times as necessary to sort it. The smallest value bubbles up to the top of the array while the largest value sinks to the bottom. (You could equally well call it a sink sort, but then nobody would know what you were talking about.) Here's the code:

```
import java.util.*;

class BubbleSort {

  public static void main(String args[]) {
   int[] n;
   n = new int[10];
   Random myRand = new Random();
   // initialize the array
   for (int i = 0; i < 10; i++) {
     n[i] = myRand.nextInt();
   }
   // print the array's initial order
   System.out.println("Before sorting:");
   for (int i = 0; i < 10; i++) {
     System.out.println("n["+i+"] = " + n[i]);
   }
   boolean sorted = false;
   // sort the array
   while (!sorted) {
     sorted = true;
     for (int i=0; i < 9; i++) {
       if (n[i] > n[i+1]) {
         int temp = n[i];
         n[i] = n[i+1];
```

```
          n[i+1] = temp;
          sorted = false;
        }
      }
    }

  // print the sorted array
  System.out.println();
  System.out.println("After sorting:");
   for (int i = 0; i < 10; i++) {
     System.out.println("n["+i+"] = " + n[i]);
   }

  }

}
```

In this case we have sorted the array in ascending order, smallest element first. It would be easy to change this to sort in descending order.

~~~ End of Article ~~~