Arrays revisited

Java, as with most languages, supports multi-dimensional arrays - 1-dimensional, 2-dimensional, 3-dimensional, ... In practice most arrays are one-dimensional, and two-dimensional (rows and columns) are also quite common. Higher dimensional arrays are less common but the same principles apply.

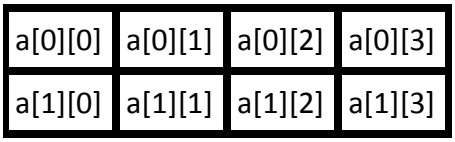
Two-dimensional arrays are used whenever the model data is best represented with rows and columns, or has two varying aspects (eg, gender and age, weight and height etc)

Terminology. Other terms you will see for a two-dimensional array are *matrix* or *table*.

Visualizing two-dimensional arrays

2-dimensional arrays are usually represented with a row-column "spreadsheet" style. Assume we have an array, a, with two rows and four columns.

```
int[][] a = new int[2][4];  // Two rows and four columns.
```



Two-dimensional arrays are usually visualized as a matrix, with rows and columns. This diagram shows the array a with its corresponding subscripts.

Style: Use named constants for array sizes

It's useful to define constants for the number of rows and columns. The reason named constants can better code is that these numbers may represent part of the problem domain and they will be used in other parts of the code. If a change is every necessary, changing one named constant will change all parts of the program. Each numeric "constant" should only appear once (the "DRY" principle).

```
static final int ROWS = 2;
static final int COLS = 3;
. . .
int[][] board = new int[ROWS][COLS];
```

Initial values

You can assign initial values to an array in a manner very similar to one-dimensional arrays, but with an extra level of braces. This would allocate a 3x3 board

```
int[][] board = new int[][] {{1,0,0},{0,1,0},{1,2,1}};
```

Use nested for loops to process 2-dimensional arrays

Index names. Two-dimensional arrays are almost always processed with nested for loops. The two index variables are often called *i* and *j* (for the row and column), but it is better to use *row* and *col* or *r* and *c*. However, if the row and column have meanings, like *month* and *year*, use those names rather than a meaning neutral name.

Iterating down rows, then across columns is often better. The most common practice is for the outer loop be for the row and the inner loop for the column. If your program requires the outer iteration by columns, that's fine, but the default row-first iteration gives the best performance because "locality of reference" improves the performance of cache and virtual memory.

```
// Purpose: Show use of nested loops to display 2D array.
//        * Nested for loops to traverse entire 2D array by rows.
//        * Building the output in a String (StringBuilder more efficient).

import javax.swing.JOptionPane;

public class TwoDimTest {
   //... Define named constants to centralize definitions
   //    and prevent use of "magic numbers".
   static final int ROWS = 2;
   static final int COLS = 4;

   public static void main(String[] args) {
      int[][] a2 = new int[ROWS][COLS];

      String output = "";   // Accumulate text here (should be StringBuilder).
      //... Print array in rectangular form using nested for loops.
      for (int row = 0; row < ROWS; row++) {
         for (int col = 0; col < COLS; col++) {
            output += " " + a2[row][col];
         }
         output += "\n";
      }
      JOptionPane.showMessageDialog(null, output);
   }

}
```

Arrays of arrays

Java builds multi-dimensional arrays from many one-dimensional arrays, the so-called "arrays of arrays" approach.

There are a couple of interesting consequences of this: Rows may be different sizes. Also, each row is an object (an array) that can be used independently.

Using .length instead of named constants

Named constants make a program very readable, but they may not be available if the array has been passed as a parameter. You can get the size of each dimension with the .length attribute. This is the most general style.

Using for loop

```
for (int row = 0; row < a2.length; row++) {
        for (int col = 0; col < a2[row].length; col++) {
           output += " " + a2[row][col];
        }
        output += "\n";

     }
```

Using *foreach* loop

```
for (int[] row : a2) {
        for (int val : row) {
           output += " " + val;
        }
        output += "\n";

     }
```

Note: the *foreach* can not be used when indexing other than forward by single steps, and it only applies to reading the elements of the array, not writing them.

Sorting algorithms

http://courses.cs.vt.edu/~csonline/Algorithms/Lessons/SelectionCardSort/selectioncardsort.swf

http://courses.cs.vt.edu/~csonline/Algorithms/Lessons/InsertionCardSort/insertioncardsort.swf

Really good multi- sort page

https://www.toptal.com/developers/sorting-algorithms/

Sorting demos:
cs50 Harvard on youtube