**Balance the Brackets**

Make sure both sides are balanced with the same number of square brackets:

// legal

int[][] credentials = new int[3][2];

/\* Illegal. Missing the 2nd dimension \*/

int[][] credentials2 = new int[3];

**Square Bracket Placement**

Just like regular arrays, the square brackets can go to the right of the type, or the right of the identifier:

int[][] credentials = new int[3][2];

or

int credentials[][] = new int[3][2];

They can be split up as well (though this is uncommon). For example a 3 dimensional array could legally be declared as:

int[] moreStuff[][] = new int[3][3][2];

or

int[][] moreStuff[] = new int[3][3][2];

**Initializing the Final Dimension**

All dimensions must be given a size during its definition, except for the final dimension.  It is legal to initialize the final dimension after it has been defined.  For example the code below is legal.

int[][][] moreStuff = new int[3][3][];

moreStuff[0][2] = new int[2]; // Line A

moreStuff[0][2][0] = 0;

moreStuff[0][2][1] = 1;

While this is legal, note that Line A is adding a 3rd dimension to only one member of 2nd dimension.  In other words, if you tried to access another 2nd dimension value and add an int to its 3rd dimension, it would throw a NullPointerException at runtime:

/\* ILLEGAL! Only [0][2] has a third dimension. \*/

moreStuff[0][1][0] = 0;

Not only must the first dimension be given a size during it's definition, there can't be any gaps of size definition between the first dimension and the subsequent dimensions. Putting all of these rules together, note that the following three lines of code are illegal and would not compile:

int[][][] moreStuff = new int[3][][3];

int[][][] moreStuff2 = new int[][3][3];

int[][][] moreStuff3 = new int[][][3];