Two Dimensional Arrays

Just as you can imagine a one dimensional array as a single row of cells, a two dimensional array can be imagined as a grid of squares.

Here is the general syntax of declaring a 2 dimensional array:

Type[][] varname = new Type[int expr][int expr];

Here is a specific example:

int[][] table = new int[5][10];

Generally, most people refer to the first index as the row of the array and the second as the column. However, it really does not matter how you think or visualize it as long as you are consistent with your interpretation. (I will explain this later once we get to an example where we print out information from a 2D array.)

Two dimensional arrays follow all the same rules that one dimensional ones do. Now, however, the expression table[3][4] stands for a single element in the array. The expression table[3] would be a one dimensional integer array of size 10.

Once again, the most common problems with two dimensional arrays are out of bound indexes and confusing array elements with array indexes. Finally, beginning students often switch the first index with the second index, which when done randomly can cause errors.

An Example

A common use for two dimensional arrays is the implementation of board games. Imagine writing a tic-tac-toe game. You could use a two dimensional character array. Here is the start of a class that would help implement tic-tac-toe:

```
public class TicTacToe {

    private char[][] board;
    private int whoseturn;
    private String[] players;
    final static private char[] pieces = { 'X' , 'O' };

    public TicTacToe(String player1, String player2) {

        board = new char[3][3];
        for (int i = 0; i<3; i++)
            for (int j=0; j<3; j++)
                board[i][j] = '_';

        whoseturn = 0;
        players[0] = player1;
        players[1] = player2;
    }

    public boolean inbounds(int row, int column) {

       if ((row < 0) || (column < 0))
           return false;
       if ((row > 2) || (column > 2))
           return false;
       return true;
    }
```

```
public boolean Move(int row, int column) {

    if  ( (board[row][column] == '_') &&
          inbounds(row,column)) {

        board[row][column] = pieces[whoseturn];
        return true;
    }
    else
       return false;
 }
```

The full implementation of this class is included in the Sample Programs.

Another example of utilizing a 2-D array is writing a Matrix class. In mathematics, matrices are objects that can be added, subtracted and multiplied, if their dimensions are appropriate for doing so. (Each matrix has a number of rows and columns, and a real number is stored in each entry.) Most applications of matrices use square matrices. Here is a small matrix class:

```
public class Matrix {

	private double[][] m;

	// values must be a square matrix
	public Matrix(double[][] values) {
		m = new double[values.length][values.length];
		for (int i=0; i<size; i++)
			for (int j=0; j<size; j++)
				m[i][j] = values[i][j];

	}

	public Matrix(int size) {
		m = new double[size][size];
		for (int i=0; i<size; i++)
			for (int j=0; j<size; j++)
				m[i][j] = 0;
	}

	public Matrix add(Matrix other) {

		if (this.length() != other.length()) return null;
		Matrix ans = new Matrix(other.length());

		for (int i=0; i<ans.length(); i++)
			for (int j=0; j<ans.length(); j++)
				ans.m[i][j] = this.m[i][j] + other.m[i][j];
		return ans;
	}

	public int length() {
		return m.length;
	}
}
```

Array of Arrays

By standard convention, a two dimensional array always has the same number of elements in each row. However, this isn't necessary in Java since arrays are dynamically allocated. It is perfectly acceptable to create an array of arrays where each row potentially has a different number of elements in it.

An example where you might want to do this is if you take samples of data at different "locations" but you might have a different number of data points for each sample.

Here are the mechanics involved (of course the type of element can be anything, it doesn't have to be an int):

```
int[][] data = new int[10][];
```

This statement allocates 10 array references, so our picture looks like this:

```
[]
[]
[]
[]
[]
[]
[]
[]
[]
[]
[]
```

In the current form, we have references to 10 arrays, but none of the space for those 10 arrays is yet allocated.

Then let's say that we want i+1 entries for the array stored at index i, we could allocate the space as follows:

```
for (int i=0; i<data.length; i++)
  data[i] = new int[i+1];
```

Now our picture looks like:

```
[]
[][]
[][][]
[][][][]
[][][][][]
[][][][][][]
[][][][][][][]
[][][][][][][][]
[][][][][][][][][]
```

In the following example (posted on the Sample Programs on the course web page), we read in data from a file with the following format:

The first line contains a single positive integer *n*, representing the number of regions from which data was collected.

The next *n* lines contain data for each region.

The first value on each line is a positive integer, *k*, representing the number of data values obtained for that particular region. The following *k* integers on that line are the data values for that region.

The sample program reads in data from a file with this format and then does some basic calculations for the data from each region.