Multiplying Matrices in Java

Multiplying matrices can be a daunting task in mathematics. This lesson will explain matrix multiplication, and show you how to use Java to multiply matrices.

Matrices

A **matrix** is a table of numbers arranged in rows and columns. Think of creating a table in Microsoft Word. You can create any combination of rows and columns. A matrix is described in terms of the number of rows times the number of columns. For example, 3 x 3 is a matrix with three rows and three columns.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Multiplying Matrices

When multiplying matrices there are a couple things to remember. Matrix multiplication is not commutative. In order to be able to multiply matrices, they have to meet one requirement. The number of columns in the first matrix has to equal the number of rows in the second matrix. The size of the resulting matrix will be the determined by the number of rows in the first matrix and the number of columns in the second matrix. In this example, the initial matrix has 3 columns and the second matrix has 3 rows so they can be multiplied. The resulting matrix will be 3 x 3.

$$\begin{bmatrix} 3 & 5 & 7 \\ 9 & 17 & 12 \\ 32 & 21 & 5 \end{bmatrix} \text{ x } \begin{bmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \\ 13 & 15 & 17 \end{bmatrix} = [129, \ldots]$$

To multiply matrices we find the **dot product**. We multiply each of the terms in the first row (3, 5, 7) by the corresponding terms in the first column (1, 7, 13) of the second matrix then add those numbers together. It looks like this:

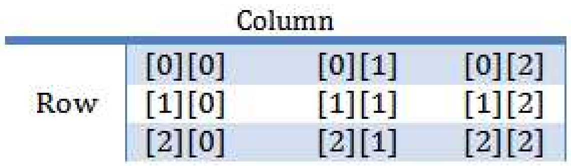
(3 * 1) + (5 * 7) + (7 * 13) = 129

We continue to do this for each term in the resulting matrix. Now, we could do all of this by hand, but we have a powerful tool at our disposal, Java. With a little coding effort, we can write a routine that will multiply our matrices.

Multiplying Matrices in Java

First, let's review how Java will see our matrices. In Java, we create two-dimensional arrays (rows/columns) to hold our data. The rows and columns will be referenced as follows. Remember that Java starts counting at zero!

We can now define the matrix we covered earlier in our code. In order to create a matrix, we use a 2-dimensional array, int [ ] [ ] myMatrix. To pre-fill the array, the entire array is denoted with curly brackets. Each row has its own set of curly brackets.



		Column	
	[0][0]	[0][1]	[0][2]
Row	[1][0]	[1][1]	[1][2]
	[2][0]	[2][1]	[2][2]

```java
int[][] myMatrixA = {
  {3,5,7},
  {9,17,12},
  {32,21,5}
};
int[][] myMatrixB = {
  {1,3,5},
  {7,9,11},
  {13,15,17}
};
```

Now comes the tricky part. We will have THREE nested loops that walk across the rows of the first matrix, then down the columns of the second matrix. These loops will take care of the math, but we have to set them up right.

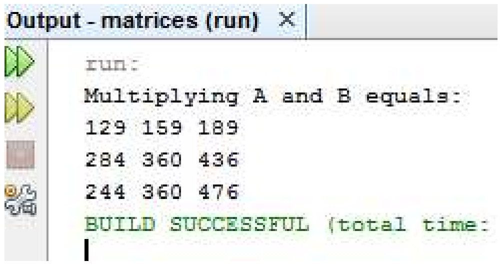
First, we will set up some counters for our loops. We then create variables to count the rows and columns for each matrix. We'll need these to know when to stop processing (we can't go past the end of the array or there will be nasty error messages). Finally, we process the array multiplication. The code follows the same mathematical logic we described above.

```java
//counters
int i, j, k, m, n;
//rows and columns for each matrix
int rowsA = myMatrixA.length;
int colsA = myMatrixA[0].length;
int rowsB = myMatrixB.length;
int colsB = myMatrixB[0].length;
//new matrix to hold result
int[][] myMatrixC = new int [rowsA][colsB];
//start across rows of A
for (i = 0; i < rowsA; i++) {
  //work across cols of B
  for(j = 0; j < colsB; j++) {
   //now complete the addition and multiplication
   for(k = 0; k < colsA; k++) {
    myMatrixC[i][j] += myMatrix [i][k] * myMatrixB[k][j];
   }
  }
}
```

Next, we'll want to show the result of our work. Again we will need a 'for' loop in order to cycle through each row/column of the new matrix. The final loop completes the calculation we discussed earlier (the += sign adds the final product of the row/column multiplication). It may seem strange to have a third loop since we are multiplying only two matrices, but we have to do the multiplication and addition functions AND create a third matrix!

```
System.out.println("Multiplying A and B equals: ");
  for(m = 0; m < myMatrixC.length; m++) {
   for n = 0; n < myMatrixC[0].length; n++) {
    System.out.print(myMatrixC[m][n] + " ");
   }
    System.out.println();
  }
```

When we run the output, the following displays. Remember that the first value we determined was 129:



```
run:
Multiplying A and B equals:
129 159 189
284 360 436
244 360 476
BUILD SUCCESSFUL (total time:
```