CS 1113
LAB #8: Two Dimensional Arrays
Dr. Cline

Lab Goals

1) To teach you how to declare and index 2-dimensional arrays
2) To show you how to iterate over two dimensional arrays with nested loops
3) To demonstrate higher dimensional arrays

Motivation

Arrays are a powerful data structure, but sometimes we want to store a table of data, not just a linear list. That is where 2 dimensional arrays come in. In Java, to make a 2 dimensional array, we make an array of arrays. This is easy because an array is just a special kind of object in Java. Common uses of 2D arrays include digital images (2D arrays of color values), grades for students in different assignments, and stock quotes for multiple stocks over time.

Declaring 2D arrays

In Java, to make a 2D array, we simply declare an array where the type is an array. To see how this works, first look at how we create an int array with 10 elements. One might type

```
int[] arr1D = new int[10];
```

What the above code actually does is create a new object of type "int Array" that reserves enough space to store 10 integers. Since an int array (written int[]) is just a type of object, we can make an array of them, by appending another set of square brackets:

```
int[][] arr2D = new int[3][7];
```

This declares a two dimensional array of ints with 3 rows and 7 columns. By convention, we think of the array as being indexed in "Row Major Order", which means that the row (y value) comes before the column (x value). Thus, visually, the array looks like this:

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

The top left element is at position [0][0], and the bottom right element is at position [2][6]. The array elements are initialized just as in the 1D case (numerics are set to 0, and object types are set to null).

Write the answers to the numbered questions in a file called lab8answers.txt. Write your name at the top of this file.

Q1) What code would you use to create a 2D array of double with 4 rows and 9 columns?

To access one of the elements of a 2D array, we use double bracket notation again, so we can write

```
arr2D[0][1] = 1;
arr2D[2][3] = 4;
```

This would give us the array:

```
0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 4 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

Suppose that we added the lines

```
arr2D[1][1] = 7;
arr2D[2][5] = 8;
```

Q2) Write the array that would result after adding the two lines of code above to your one line of code from Q1.

Iterating over a 2D array

If we want to visit or process every element of a 2D array, we need a nested for loop to do it. Generally, when you iterate over a 2D array, you want to do it in row major order, because this is more memory efficient. The reason why is that adjacent elements in a row are stored contiguously (one after another) in memory. Another reason, for an English speaker, is that this is the same order your read a book—top to bottom, left to right.

The following code iterates over arr2D, changing each value to 100:

```
for (int r=0; r < arr2D.length; r++)
{
      for (int c=0; c < arr2D[r].length; c++)
      {
            arr2D[r][c] = 100;
      }
}
```

I'm using the variables "r" and "c" because they iterate of the rows and columns of the array, respectively. The outer loop iterates over all the rows, and the inner loop iterates over all the columns in a single row. Notice how the row index (r) comes before the column index (c). Also, notice how the loop bounds are handled in the interior loop. While the outer loop goes from 0 to arr2D.length, the inner loop goes from 0 to arr2D**[r]**.length, which is then length of row r.

Q3) What code would you use to add 1 to all the elements of arr2D? Write your answer in lab8answers.txt.

Part 1) Type in, or copy, and finish the following program, which computes and prints an addition table using nested loops and a 2D array:

```
// Lab 8a
// <Your name here>
// <Your section number here>

public class Lab8a
{
      public static void main(String[] args)
      {
            // Read an integer from the command line
            int tableSize = Integer.parseInt(args[0]);

            // Create a 2D array of ints to hold the addition table
            int table[][] = // [add code here];
```

```
            // Fill out the table using a nested loop so that
            // table[r][c] = r + c
            // [add code here]

            // Iterate over the table, printing each value
            // so that columns align (hint: use a nested loop,
            // and printf)
            // [add code here]
      }
}
```

When finished, with an input value of 4, the program should print something like:

```
 0  1  2  3
 1  2  3  4
 2  3  4  5
 3  4  5  6
```

Part 2) Compile and run the program with the command line argument 9.

```
java Lab8a 9
```

Part 3) Once you are satisfied that the program is working properly, run the program twice, with command line arguments of 9 and 10, redirecting the output to out8a.txt and out8b.txt.

Ragged Arrays

It is also possible to make each of the rows of an array a different size. This is called a "ragged" or "jagged" array. Ragged arrays can be quite useful if you have a list of lists, but each list can be a different length. You declare a ragged array by first declaring an array to hold all the rows, and then declaring each row separately in a loop, as shown in the code below.

Part 4) Start with the a copy of Lab8b.java (shown below). Fill in the code to create a subtraction table with all the non-negative elements of the table.

```
// Lab 8b
// <Your name here>
// <Your section number here>

public class Lab8b
{
      public static final int SIZE = 10;
      public static void main(String[] args)
      {
            int table[][] = new int[SIZE][];
            for (int r=0; r<table.length; r++)
            {
                  // Set table[r] to a new array of int with
                  // size r+1.
                  // [add code here]
            }

            // Write code that fills in the array with
            // a subtraction table, so that table[r][c] = r-c.
            // Hint: Use a nested loop, and make
            // sure the inner loop iterates goes from 0
```

```
            // to the length of the row: table[r].length
            // [add code here]

            // Iterate over the table, printing each value
            // so that columns align (hint: use a nested loop,
            // and printf)
            // [Your code here]
      }
}
```

Part 5) Once you are satisfied that the program is working properly, run it again, redirecting the output to the file outTriangle.txt.

Higher Dimensional Arrays

You may be thinking, if two dimensional arrays are just arrays of arrays, can I make higher dimensional arrays as well? Sure. You can make 3, 4, 5 and even higher dimensional arrays if you wish. All you have to do is append more square brackets on the end. For example, the following declares a 4 dimensional array of doubles:

```
double[][][][] arr4D = new double[100][100][100][100];
```

An array that size would have a hundred million elements! Are such structures useful? Yes, and they abound in scientific and multimedia applications. For example, an MRI scan produces a 3 dimensional volume of data that shows a person's body on the inside. A movie is just a time series of images (3 dimensions). A simulation to analyze the aerodynamics of an aircraft wing might compute a the airflow at every point in space around the wing over time. This is a 4 dimensional structure (3 spatial dimensions and 1 time dimension). To iterate over this array would require a nested loop that is 4 deep!

Q4) What code would you use to create a 3 dimensional array of int called b, with 10 elements in each dimension? Write the answer in lab8answers.txt.

Part 6) Finish the code below, writing the 3-deep nested loop needed to set all of the values in arr to 1.5 (hint: you will need 3 different loop variables to do this).

```
// Lab 8c
// <Your name here>
// <Your section number here>
// <Largest value that worked for you>

public class Lab8c
{
      public static void main(String[] args)
      {
            int SIZE = Integer.parseInt(args[0]);

            // Create a 3-dimensional array of double where
            // the size of each dimension is SIZE:
            double table[][][] = // [add code here];

            // Fill in the table using a 3-deep nested loop so that
            // table[a][b][c] = 1.5 for all a,b,c
            // [add code here]
      }
}
```

Part 7) Make sure that your code compiles and runs with a size value of 10.

Part 8) Re-run the program, increasing the SIZE value until you find the largest value you can use without getting an OutOfMemory error. Write the largest size value that worked for you in a comment at the top of Lab8c.java.

Part 9) Make sure that your programs follow the class coding standards for indentation, capitalization, and having your name at the top.

Hand in your lab

Hand in your lab to the D2L dropbox, including the files

```
lab8answers.txt
Lab8a.java
Lab8b.java
Lab8c.java
out8a.txt
out8b.txt
outTriangle.txt
```