Chapter 6 – Arrays, Part D – 2d Arrays and the Arrays Class

Section 6.10 – Two Dimensional Arrays

1. The arrays we have studied so far are *one-dimensional* arrays, which we usually just call *arrays*. Two-*dimensional* (2d) arrays are used to model (represent) a *table* of information, with rows and columns. For example, sales figures for one week are stored in a table like this:

		Day of Week						
		Mon	Tue	Wed	Thu	Fri	Sat	Sun
Store Location	Valdosta	56.22	72.42	38.23	51.91	47.23	52.34	42.35
	Waycross	54.37	74.67	34.57	59.85	41.48	58.89	38.47
	Douglass	58.87	81.08	37.83	53.42	53.93	57.73	43.58
	Quitman	55.76	78.69	39.64	54.16	51.58	58.58	39.83

We notice that the table has 4 rows and 7 columns of data. In Java, we can create this table

```
double[][] sales;

sales = new double[4][7];
```

Which we can think of like this:

		Column Index (dayOfWeek)						
		0	1	2	3	4	5	6
Row Index (location)	0	56.22	72.42	38.23	51.91	47.23	52.34	42.35
	1	54.37	74.67	34.57	59.85	41.48	58.89	38.47
	2	58.87	81.08	37.83	53.42	53.93	57.73	43.58
	3	55.76	78.69	39.64	54.16	51.58	58.58	39.83

We see that there are two indices on a 2d array. We can think of the first as designating the row and the second as designating the column (when we are dealing with a rectangular array as shown above). Thus, to access individual elements we write:

```
sales[row][col]
```

$$0 \le row < sales.length$$

$$0 \le col < sales[0].length$$

The expression for the number of columns, *sales[0].length* is only valid for a rectangular array. This notation will be explained later.

For example:

```
sales[2][3]          // 53.42

sales[3][2]          // 39.64
```

2. You can also create a 2d array initializer:

```
int[][] scores =
  { { 87, 73, 91 },
    { 84, 95, 62 },
    { 92, 95, 99 },
    { 78, 54, 79 }  };
```

3. Consider the array we previously considered.

		Column Index (dayOfWeek)						
		0	1	2	3	4	5	6
Row Index (location)	0	56.22	72.42	38.23	51.91	47.23	52.34	42.35
	1	54.37	74.67	34.57	59.85	41.48	58.89	38.47
	2	58.87	81.08	37.83	53.42	53.93	57.73	43.58
	3	55.76	78.69	39.64	54.16	51.58	58.58	39.83

Notice that the first row is an (1d) array. Thus, we say that *scores[0]* is an (1d) array. Similarly, each row can be thought of as a 1d array: *scores[0], scores[1], scores[2], scores[3]*. Occasionally, we may pull of a row from a 2d array.

```
double[][] sales;
sales = new double[4][7];
 ...
double[] salesRow = sales[0];
```

Thus, we see that a 2d array, is an *array* of *arrays*, that a 2d array is a 1d array whose elements are each a 1d array (a row), that a 2d array is a 1d array whose elements are the 1d arrays that represent the rows. Thus, *sales* is a array whose elements are the arrays: *scores[0], scores[1], scores[2], scores[3]*.

4. The *total* number of *rows* in a 2d array is given by: *scores.length*, which in the case above is 4. Each *row* in the *scores* array has the same number of columns, which can be found by:

Row	Number of Columns	Value
0	`scores[0].length`	3
1	`scores[1].length`	3
2	`scores[2].length`	3
3	`scores[3].length`	3

Processing Rows

1. To loop through all the cells in a table, top-to-bottom (down the rows), left-to-right (across the columns):

```
// i = row = location
// j = column = day of week

 for( int i=0; i<sales.length; i++ )
{
      for( int j=0; j< sales[i].length; j++ )
      {
            System.out.print(sales[i][j] + " " );
      }
      System.out.println();
}
```

or:

```
for( int[] row : sales )
{
      for( int val : row ) System.out.print( val + " " );
      System.out.println();
}
```

2. We can also think of this as a *row-based* processing of the 2d array. Notice that the outer loop controls the rows while the inner loop controls the column. So, using the approach above, the outer loop specifies a particular row and then the inner loop iterates over the columns in that row.

Example 1

Write a method which returns an array of the totals for each store for the week.

```
public static double[] calcStoreTotals ( double[][] sales )
{
   double[] tots = new double[ sales.length ];

   for( int loc=0; loc<sales.length; loc++ )
   {
      for( int day=0; day<sales[0].length; day++ )
      {
         tots[loc] += sales[loc][day];
      }
   }
   return tots;
}
```

We can call this method with code like:

```
double[][] sales;
sales = new double[4][7];
...
double[] totals = calcStoreTotals( sales );
```

Processing Columns

1. We can also do a *column-based* processing. For example, we might want to compute the totals for each day, across all stores (e.g. add up the values in each column). To do this, we will let the outer loop control the columns while the inner loop controls the rows. So, using this approach, the outer loop specifies a particular column (day) and then the inner loop iterates over the rows (locations) in that row.

```
// i = row = location
// j = column = day of week

for( int col=0; col<sales[0].length; col++ )
{
     for( int row=0; row<sales.length; row++ )
     {
          System.out.print(sales[row][col] + " " );
     }
     System.out.println();
}
```

Example 2

Write a method which returns an array of the totals for each day of the week.

```
public static double[] calcDailyTotals ( double[][] sales )
     {
        double[] tots = new double[ sales[0].length ];

        for( int day=0; day<sales[0].length; day++ )
        {
           for( int loc=0; loc<sales.length; loc++ )
           {
              tots[day] += sales[loc][day];
           }
       }
        return tots;
     }
```

Example 3

A *square* array (matrix) is one where the number of rows and columns is the same. Assume you have a square matrix. Write a method which returns the sum of the elements along the main diagonal

```
public static double calcMainDiagTotal ( double[][] matrix )
{
   double total = 0;

   for( int i=0; i<matrix.length; i++ )
      total += matrix[i][i];

   return total;
}
```

Example 3

A *square* array (matrix) is one where the number of rows and columns is the same. Assume you have a square matrix. Write a method which returns the sum of the elements along the main diagonal.

```
public static double calcMainDiagTotal ( double[][] matrix )
{
   double total = 0;

   for( int i=0; i<matrix.length; i++ )
      total += matrix[i][i];

   return total;
}
```

Example 4

Assume you have a square matrix. Write a method which returns the sum of the elements along the off diagonal.

```
public static double calcOffDiagTotal ( double[][] matrix )
{
   double total = 0;

   for( int col=0, row=matrix.length-1; col<matrix.length; col++, row-- )
      total += matrix[row][col];

   return total;
}
```

Example 5

Write a method that determines if a 2d arrays is square.

```
public static boolean isSquare ( double[][] matrix )
{
   int nRows = matrix.length;

   for( int row=0; row<nRows; row++ )
      if( nRows != matrix[row].length )
         return false;
   return true;
}
```

Example 6

Write a program to read student names and test scores from the user. Prompt for the number of students and the number of test scores. Print the names followed by the scores for that student.

```
public static void main(String[] args)
{
      String[] students;
      int[][] tests;

      students = buildStudentsArray();

      tests = buildTestsArray( students.length );

      getStudentsAndScores( students, tests );

      printStudentsAndScores( students, tests );
}

public static String[] buildStudentsArray()
{
      int numStudents;

      Scanner in = new Scanner( System.in );

      System.out.print( "How many students are there?" );
      numStudents = in.nextInt();

      String[] s = new String[numStudents];

      return s;
}

public static int[][] buildTestsArray( int numStudents )
{
      int numTests;

      Scanner in = new Scanner( System.in );

      System.out.print( "How many tests are there?" );
      numTests = in.nextInt();

      int[][] tests = new int[numStudents][numTests];

      return tests;
}
```

```
public static void getStudentsAndScores( String[] names, int[][] scores )
{
      Scanner in = new Scanner( System.in );

      for( int i=0; i<scores.length; i++ )
      {
            System.out.print( "What is student " + (i+1) + "'s name?" );
            names[i] = in.next();

            for( int j=0; j<scores[i].length; j++ )
            {
                  System.out.print( "What is test score " + (j+1) + "?" );
                  scores[i][j] = in.nextInt();
            }
      }

}

public static void printStudentsAndScores( String[] names, int[][] scores )
{

      for( int i=0; i<scores.length; i++ )
      {
            System.out.print( names[i] + ": " );

            for( int j=0; j<scores[i].length; j++ )
            {
                  System.out.print( scores[i][j] + " " );
            }

            System.out.println();
      }
}
```

Jagged Arrays

1. Since a 2-d array is an *array* of *arrays*, the number of columns can be different for each row. For instance,

		Columns			
		[0]	[1]	[2]	[3]
	[0]	87	73	58	91
Rows	[1]	84	95	62	
	[2]	92	95		
	[3]	78			

This is called a *jagged* array (book says *ragged*). You can create one with an array initializer:

```
int[][] jaggedArray =

  { { 87, 73, 58, 91 },
    { 84, 95, 62 },
    { 92, 95 },
    { 78 }  };

for( int i=0; i<jaggedArray.length
       System.out.print( jaggedArray[i].length + “, “ )

       ---> 4, 3, 2, 1
```

2. You can also create one with code like this (for example);

```
int[][] jaggedArray;

jaggedArray = new int[4][];

jaggedArray[0] = new int[4];
jaggedArray[1] = new int[3];
jaggedArray[2] = new int[2];
jaggedArray[3] = new int[1];
```

3. *Jagged* arrays are very useful. Suppose you wanted to write a program so that a professor could record the final average for each student in each of her classes. Naturally, the number of students in each class would be different.

		Student					
		[0]	[1]	[2]	[3]	[4]	[5]
	[0]	87	73	58	91		
Class	[1]	84	95	62			
	[2]	92	95	88	69	83	74

Example 7

a. Problem: Write a program that allows a professor to enter the grades for students in each of his classes.

b. Algorithm – First Pass:

```
Get number of classes
Loop over number of classes
    Get number of students in current class
    Loop over number of students in current class
        Read score
```

c. Algorithm – Second Pass:

```
Get number of classes
Create array specifying number of classes (rows)
Loop over number of classes
    Get number of students in current class
    Create 1d array of students for current class (specify columns for current row)
    Loop over number of students in current class
        Read score into array
```

d. Code

```
public static void main(String[] args)
{

      int numClasses, numStudents;
      int[][] tests;

      Scanner in = new Scanner( System.in );

      System.out.print("How many classes do you have?" );
      numClasses = in.nextInt();

      tests = new int[ numClasses ][];

      for( int i=0; i<numClasses; i++ )
      {
            System.out.print( "How many students in class " + (i+1) +
                                "? " );
            numStudents = in.nextInt();

            tests[i] = new int[ numStudents ];

            for( int j=0; j<numStudents; j++ )
            {
               System.out.print("What is final Average for student "
                             + (j+1) + " in class " + (i+1) +"? " );

               tests[i][j] = in.nextInt();
            }
      }
      printFinalAverages( tests );
}

public static void printFinalAverages( int[][] scores )
{
      for( int i=0; i<scores.length; i++ )
      {
            System.out.print( "Class " + (i+1) + ": " );

            for( int j=0; j<scores[i].length; j++ )
            {
                  System.out.print( scores[i][j] + " " );
            }

            System.out.println();
      }
}
```

The java.util.Arrays Class

The documentation for the java.util.Arrays classs is found using the *Java API* (application programming interface) (see link on website under “Links”).

The Arrays class has several useful, static methods: *sort, binarySearch, toString, equals*. As an example, consider this code:

```
int[] scores =  {92, 83, 87, 43, 62, 97, 77};
int[] scores2 = {92, 83, 87, 43, 62, 97, 77};

System.out.println( "scores  = " + Arrays.toString( scores ) );
System.out.println( "scores2 = " + Arrays.toString( scores ) );

System.out.println( "Are scores & scores2 equal? " +
                    Arrays.equals( scores, scores2 ) );

Arrays.sort(scores);

System.out.println( "scores, sorted = " + Arrays.toString(scores) );
System.out.println( "Are scores & scores2 equal? " +
                    Arrays.equals(scores, scores2) );

int[] keys = { 42, 43, 44, 53, 62, 65, 77, 79, 83, 85, 87, 90, 92, 95,
               97, 100 };

for( int key : keys )
{
     System.out.println( "Arrays.binarySearch( scores, " +
                         key +
                         " ) = " +
                         Arrays.binarySearch( scores, key ) );
}
```

Output

```
scores  = [92, 83, 87, 43, 62, 97, 77]
scores2 = [92, 83, 87, 43, 62, 97, 77]

Are scores & scores2 equal? true

scores, sorted =
     [43, 62, 77, 83, 87, 92, 97]

Are scores & scores2 equal? false

Arrays.binarySearch( scores, 42 ) = -1
Arrays.binarySearch( scores, 43 ) = 0
Arrays.binarySearch( scores, 44 ) = -2
```

```
Arrays.binarySearch( scores, 53 ) = -2
Arrays.binarySearch( scores, 62 ) = 1
Arrays.binarySearch( scores, 65 ) = -3
Arrays.binarySearch( scores, 77 ) = 2
Arrays.binarySearch( scores, 79 ) = -4
Arrays.binarySearch( scores, 83 ) = 3
Arrays.binarySearch( scores, 85 ) = -5
Arrays.binarySearch( scores, 87 ) = 4
Arrays.binarySearch( scores, 90 ) = -6
Arrays.binarySearch( scores, 92 ) = 5
Arrays.binarySearch( scores, 95 ) = -7
Arrays.binarySearch( scores, 97 ) = 6
Arrays.binarySearch( scores, 100 ) = -8
```