Lab 8 - Reference

This handout presents a brief overview of declaring and accessing two-dimensional arrays with Java. It also discusses how to generate pseudorandom numbers.

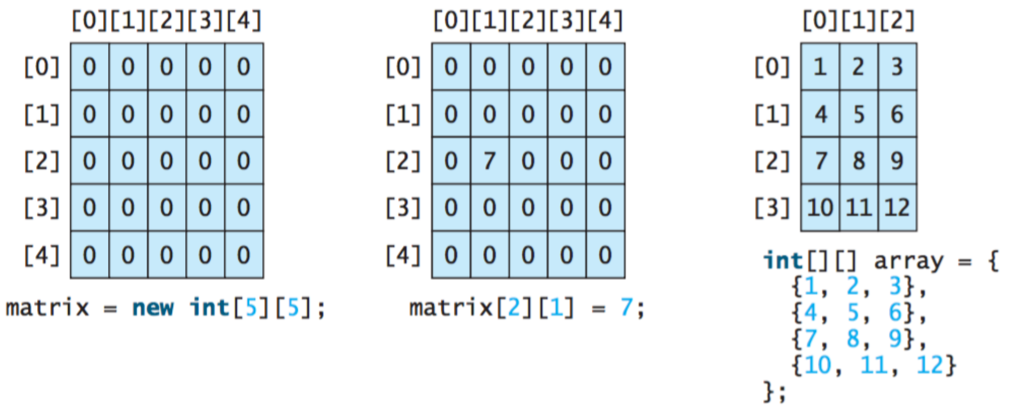
1. Declaring a Two-Dimensional Array

The syntax for declaring a two-dimensional array in Java is:
elementType[][] arrayRefVar;
For instance:
int[][] matrix;

You can allocate memory for this array using the following syntax:
matrix = **new int**[5][5];

The numbers correspond to the length of rows and columns, respectively. Below are a few different ways from your textbook to access and declare two-dimensional arrays:

Two subscripts are used in a two-dimensional array, one for the row and the other for the column. As in a one-dimensional array, the index for each subscript is of the `int` type and starts from `0`, as shown in Figure 8.1a.



```
matrix = new int[5][5];
```

(a)

```
matrix[2][1] = 7;
```

(b)

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

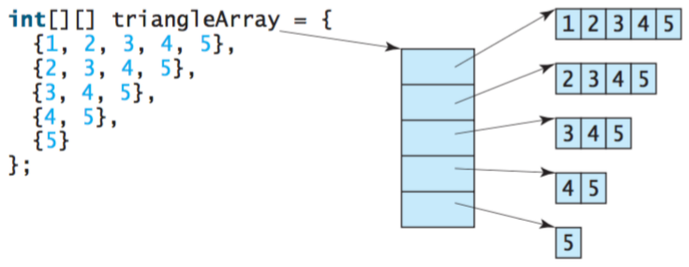
(c)

2. Accessing the Dimensions

In Java, multidimensional arrays are arrays where each element is another one-dimensional array. Recall that for one-dimensional arrays, we can access the array length as: **matrix**.length. For our matrix, this would yield the number of rows.

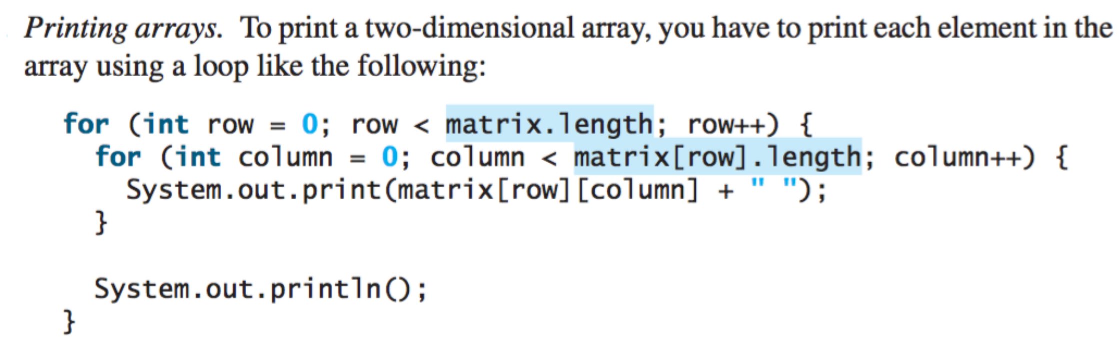
To obtain the number of columns in a rectangular array, we can look at the length of any row: **x[0].length**, **x[1].length**, . . . , and **x[x.length-1].length**. Assuming that you are dealing with a rectangular array, it would be logical to employ **x[0].length** to obtain the number of columns.

Note that you can create an array where each row has a different number of columns. This is called a ragged array. The example shown below has 5 rows and 5,4,3,2,1 columns, respectively.



```
int[][] triangleArray = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```

As with one-dimensional arrays, we utilize loops to access individual elements of the array. For a two-dimensional array, we generally use two loops to initialize the array with certain values or to print it. One loop accesses the rows and the second one accesses the columns. Below is an example from your textbook that will loop through each row and column in a two-dimensional array called **matrix** and print each element. Notice that after printing all elements in each row, it moves to the next line.



Printing arrays. To print a two-dimensional array, you have to print each element in the array using a loop like the following:

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

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

3. Random numbers

We have multiple convenient options to generate pseudorandom numbers in Java. One popular method is **java.lang.Math.random()** that returns a **double** value between 0 and 1.

Another option is to use **java.util.Random** class. The **Random** class allows one to generate pseudorandom numbers as **int, long, double, float** or **Boolean**. It also allows one to construct a **Random** object with a given seed. If multiple **Random** objects have the same seed, they will generate the same sequence of pseudorandom numbers.

Note that **Math.random** calls **Random.nextDouble()** internally. Thus, it may be more efficient to use an instance of **java.util.Random**. For example, the following code creates a **Random** object with a given seed and returns a pseudorandom integer between 0 and 99.

```
import java.util.Random;
…
Random rnd = new Random();
int seed = 1331; //Optional: set seed to create repeatable results
rnd.setSeed(seed);
int myrandom0to99 = rnd.nextInt(100); //returns a random int value between 0 to 99.
```