What we will cover:
Array Contents in Java Memory
Ragged Arrays
Processing 2d Arrays

Array Contents in Java Memory
To further solidify the array of arrays concept, note what happens when you pass an array directly to the System.out.println() method. Consider this code:

```java
int[][] data = new int[5][5];
System.out.println(data);
System.out.println(data[0]);

System.out.println();

double[][] data2 = new double[5][5];
System.out.println(data2);
System.out.println(data2[0]);
```

What do you think this should print?

Class Exercise 38

Ragged Arrays
Up to this point, we have only looked at arrays that have both dimension sizes specified. Java also provides the ability to create different sized arrays where each “row” can have a different number of “columns.” We refer to these arrays as ragged arrays because of how they look visually.

The book uses an example of an array that looks like an inverted triangle:

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

Instead of a grid, this array looks like an inverted triangle. Note that the lengths of the interior arrays could be of any length. If the number of rows is known, you can declare an array with only the first dimension specified.

```java
int[][] triangleArray = new int[5][];
```

We could also have arrays with less organized raggedness:

```java
int[][] raggedArray = {
    {2, 9, 12, 15},
    {5, 67, 8, 9, 10},
    {2},
    {3, 4},
    {19, 17, 28, 5, 3, 5, 67}
};
```
Processing 2d Arrays

Because of the varying nature and dimensions of 2d arrays, we need to specify general ways to work with them. As you saw with single-dimensional arrays, there is a tried-and-true solution involving for loops. Let’s concentrate first on form, then specific interactions.

To access a 2d array, we must provide an index pair: a row and column if we stick to our earlier visual descriptions. Iteratively processing those arrays requires the ability to walk through elements in the manner we have typically described: we start with a row, process all columns, and then move to the next row.

A general algorithm for working with each individual element in a 2d array can be described as such: Start with idea of the grid and “lock” onto the first row. Process all columns, then “lock” onto the second row. Repeat until all rows have been visited.

Notice that a nested for loop solves this problem for us (remember the multiplication table example?):

```java
for (int row = 0; row < matrix.length; row++) {
    for (int column = 0; column < matrix[row].length; column++) {
        // we can now access matrix[row][column]
    }
}
```

At this point, anything we put in the inner loop’s body has access to the current element. We could initialize that element to a random value or print it out.

Class Exercise 39