Jagged arrays in Java

1. Jagged arrays

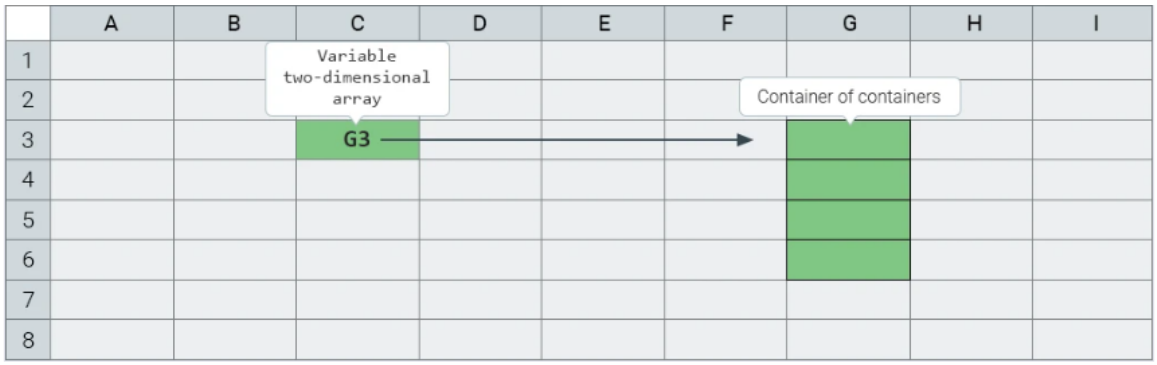
As a Java programmer you can not only swap the rows of a two-dimensional array, but also construct an array however you want.

Let's say you want the first row of a two-dimensional array to have a length of `10`, and you want the length of the second row to be `50`. Can we do that? Yes, we can.

First, we need to create a 'container of containers' — this is the first array, which will store references to arrays of rows. This is how it's done:

```
int[][] name = new int[height][];
```

You simply **omit the second dimension**, and the Java machine creates a container of containers. This is what will be in memory after executing this code:



And, well, you already know how to create one-dimensional arrays 😳

This is what the resulting code will look like:

```
// Matrix of important data
int[][] matrix = new int[2][];
matrix[0] = new int[10];
matrix[1] = new int[50]
```

Two-dimensional array

The zeroth row is an array of `10` elements
The first row is an array of `50` elements

We have just created a so-called "**jagged array**".

And if we now want to display all the elements of this array on the screen, then the array's `length` property will come in handy: after all, the lengths of the array's rows are different.

By the way, how do you find the length of a 'container of containers' in our example? It is also an array object, which means that it has a length. The correct answer is `matrix.length`.

How about for the arrays that comprise our rows? `matrix[0].length`

2. Working with a two-dimensional array

Suppose you want to display a two-dimensional array. How do you do that?

Our code will look something like this:

```
int[][] matrix = new int[3][];
matrix[0] = new int[]{1, 2, 3, 4, 5, 6};
matrix[1] = new int[]{1, 2, 3};
matrix[2] = new int[]{1};
for (int i = 0; i < matrix.length; i++) {
   for (int j = 0; j < matrix[i].length; j++)
      System.out.print( matrix[i][j] + " " );
   System.out.println();
}
```

Create an array
Fill the array with values

Outer loop that iterators over the rows of the array.
Inner loop that iterates over the cells of a single row.

You need two nested loops. The first we call outer, and the second — inner.

In the outer loop (the `i` variable), we sequentially go through all the rows (arrays) that make up our two-dimensional array. Each value of `i` corresponds to a row with that index.

In the inner loop (the `j` variable), we iterate over all the cells in the rows. Thanks to the inner loop, a row, which consists of the values of one one-dimensional array, will be displayed on the screen.

This is what will be displayed:

One row of the array is processed	1 2 3 4 5 6
Two rows of the array are processed	1 2 3 4 5 6 1 2 3
Three rows of the array are processed	1 2 3 4 5 6 1 2 3 1

3. Multidimensional arrays

One more interesting fact about arrays, one that you've probably already guessed. If you can make a two-dimensional array, then can you make a three-dimensional array?

Yes, you can create an array of any dimension. Such arrays are called 'multidimensional'.

Just for fun, let's create a multidimensional array that has 4 dimensions.

```
int[][][][] matrix = new int[2][3][4][5];
```

This code is too simple, isn't it?

What if you create it manually?

```
int[][][][] matrix;
matrix = new int[2][][][];                // Create a 2-element array of references to references to refere
for (int i = 0; i < matrix.length; i++)
{
  matrix[i] = new int[3][][];               // Create a 3-element array of references to references
  for (j = 0; j < matrix[i].length; j++)
  {
    matrix[i][j] = new int[4][];            // Create a 4-element array of references
    for (k = 0; k < matrix[i][j].length; k++)
      matrix[i][j][k] = new int[5];         // Create 5-element arrays of integers
  }
}
```

And that's just creating the array! Then you also need to work with it somehow.

Bonus task: write code that displays all the values in a three-dimensional array.