CODEGYM

Java Developer in 12 months

Mentor-supported training program from CodeGym

MODULE 1. JAVA SYNTAX

Lesson 7 Arrays

CODEGYM

Lesson plan

- Two-dimensional arrays
- Jagged arrays in Java
- Multidimensional arrays

CODEGYM

Two-dimensional arrays

`int[] myArray = new int[8]` is a one-dimensional array.
An array whose elements are arrays, that is — an array of arrays, is two-dimensional.
It's like a table that has a row number and a column number.

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

name is the name of the array variable
width is the width of the table (in cells)
height is the height of the table.

Example:

```
int[][] data = new int[2][5];
data[1][1] = 5;
```

We create a two-dimensional array:
two columns and 5 rows.
Write 5 to cell (1, 1).

0	0
0	5
0	0
0	0
0	0

By the way, there's an interesting dilemma here:

When we create an array using `new int[2][5];`, do we have a table of 'two rows and 5 columns' or is it 'two columns and 5 rows'?" In other words, are we first specifying the width and then the height... or vice versa, first the height and then the width? Well, as we often say, everything is not so simple here.

Let's start with a question: How is an array stored in memory?
Of course, computer memory doesn't actually have a matrix in it: each location in memory has a sequential numeric address: 0, 1, 2, ... In our case, we speak of a 2 × 5 matrix, but in memory it is just 10 consecutive cells, nothing more. Nothing indicates where the rows and columns are.

You can also use fast initialization for two-dimensional arrays:

```
// Lengths of months of the year in each quarter
int[][] months = { {31, 28, 31}, {30, 31, 30}, {31, 31, 30}, {31, 30, 31} };
```

CODEGYM

Jagged arrays in Java

If you need to create a two-dimensional array whose rows varying in length:
First, you need to create a "container of containers", i.e. the first array, which will store references to one-dimensional arrays.

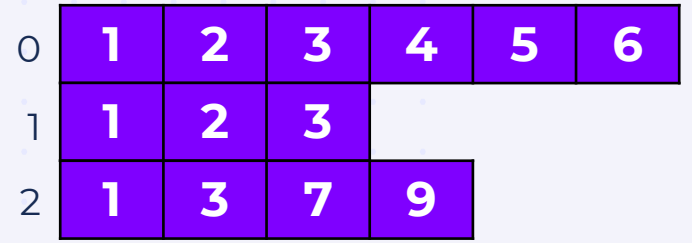
```
int[][] name = new int[height][];
```

Displaying a jagged two-dimensional array:

```
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, 3, 7, 9};
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 iterates over the rows of the array.
Inner loop that iterates over the cells of a single row.



CODEGYM

Multidimensional arrays

In Java, you can create an array whose elements are two-dimensional arrays. Such an array is said to be three-dimensional. You can also create a four-dimensional array, i.e. array of three-dimensional arrays. And so on.
In general, such arrays are called 'multidimensional', or n-dimensional, where n is the nesting depth

Let's create a multidimensional array that has 4 dimensions

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

Filling a four-dimensional array with default values (0):

```
int[][][][] matrix;
matrix = new int[2][][][];              // Create a 2-element array of references to references to references
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
   }
}
```

CODEGYM

Methods of Array class

Array class (its full name is java.util.Arrays), putting the most popular array-related actions into it. It has a lot of methods for every occasion, but the simplest and most often used are the next:

Method	Description
toString	Returns a string representation of the contents of the specified array.
deepToString	Returns a string representation of the "deep contents" of the specified array.
equals	Returns true if the two specified arrays are equal to one another.
deepEquals	Returns true if the two specified arrays are deeply equal to one another.
fill	Assigns the specified value to each element of the specified array
copyOf	Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length.
copyOfRange	Copies the specified range of the specified array into a new array.
sort	Sorts the specified array into ascending numerical order.
binarySearch	Searches the specified array of specified type for the specified value using the binary search algorithm.

CODEGYM

Homework

MODULE 1. JAVA SYNTAX

Complete Level 8

CODEGYM

Answers to questions

MODULE 1. JAVA SYNTAX