Theory: Multi-dimensional array

§1. An array of arrays

Some structures such as matrices and tables are conveniently modeled by two-dimensional arrays. Java provides a possibility to process two and more-dimensional arrays.

To create a multi-dimensional array we should use an array as an element of another array. In this case, we create an array of arrays.

Nested loops are often used to iterate through multi-dimensional arrays.

Current topic:

Multi-dimensional array

Topic depends on:

✓ Iterating over arrays

§2. 2-dimensional arrays

Here is an example that creates a two-dimensional array:

```
// two-dim array - the array of arrays
int[][] twoDimArray = {
        {1, 2, 3, 1}, // first array of int
        {3, 4, 1, 2}, // second array of int
        {4, 4, 1, 0}  // third array of int
};
```

In this case, the length of `twoDimArray` is 3 (because it includes 3 arrays as elements). The length of each nested array is 4.

Now if you'd like to get an integer element from the array you should write two indexes:

```
int number = twoDimArray[0][2]; // it is 3
```

In this case, the first index specified an element (nested array or row) of `twoDimArray`. The second index specified the element inside the nested array.

Important, all nested arrays can have a different length. See an example below:

```
int[][] twoDimArray = new int[3][];

twoDimArray[0] = new int[] { 1, 2, 3, 4 }; // the length is 4
twoDimArray[1] = new int[] { 5, 7, 3};     // the length is 3
twoDimArray[2] = new int[] { 8 };          // the length is 1

// let's output the array
for (int i = 0; i < twoDimArray.length; i++) {
    System.out.println(Arrays.toString(twoDimArray[i]));

}
```

The code above outputs:

```
[1, 2, 3, 4]
[5, 7, 3]
[8]
```

§3. 3-dimensional arrays

You can create an array with more than 2 dimensional (3-dim, 4-dim, 5-dim and so on).

Let's create 3-dimensional array of integers:

```
int[][][] cubic = new int[3][4][5];
```

Actually, this cubic array is represented as three 2-dimensional arrays 4x5.

Let's fill each 2D array of the cubic by the following rules:

- the first 2D array must contain elements equal 1;
- the second 2D array must contain elements equal 2;
- the third 2D array must contain elements equal 3.

The classic way to do that is to use three **for** loops: one outer loop and two nested.

```
int[][][] cubic = new int[3][4][5]; // a three-dimensional array (cube)

int current = 1; // it stores a value to fill elements


for (int i = 0; i < 3; i++) { // iterating through each 2D array ("table" or "matrix")

 for (int j = 0; j < 4; j++) { // iterating through each 1D array ("vector") array of a "matrix"

    for (int k = 0; k < 5; k++) { // iterating through each element of a vector

            cubic[i][j][k] = current; // assign a value to an element
        }

    }

    current++; // get the next value to the next "matrix"

}



for (int i = 0; i < 3; i++) {

    for (int j = 0; j < 4; j++) {

        for (int k = 0; k < 5; k++) {

            System.out.print(cubic[i][j][k] + " ");

        }

        System.out.println();

    }

    System.out.println();

}
```

This code prints:

```
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1

2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2



3 3 3 3 3

3 3 3 3 3

3 3 3 3 3

3 3 3 3 3
```

So, each 2D array (or "matrix") has its own value.

Table of contents:

It is also possible to use **for-each** loop and methods of the class `Arrays` to fill and print multidimensional arrays.

```
// this code fills the 3-dimensional array
int current = 1;
for (int[][] dim2Array : cubic) {     // for each 2-dim array
    for (int[] vector : dim2Array) {  // for each 1-
dim array (vector) of 2-dim array
        Arrays.fill(vector, current); // fill the vector
    }
   current++; // the next current
}


// this code prints all 2-dimensional arrays

for (int[][] dim2Array : cubic) {

    for (int[] vector : dim2Array) {

        System.out.println(Arrays.toString(vector));

    }

    System.out.println();

}
```

This code prints three 2-dim arrays:

```
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]

[2, 2, 2, 2, 2]
[2, 2, 2, 2, 2]
[2, 2, 2, 2, 2]
[2, 2, 2, 2, 2]


[3, 3, 3, 3, 3]

[3, 3, 3, 3, 3]

[3, 3, 3, 3, 3]

[3, 3, 3, 3, 3]
```