9 minutes reading

You are already familiar with arrays: this structure allows you to store elements of the same type. But why limit yourself to just one dimension? It would be so convenient to store matrices, 3D objects, or even 4D objects in one array! And now is the time: the adventure into multidimensional arrays is starting in three...two...one!

An array of arrays

First, let's figure out what we mean by a **multidimensional array**.

We can say that a multidimensional array is an array of arrays. That is, to create a multidimensional array, we need to think of an array as an element of another array. Eventually, we get a multidimensional array.

A multidimensional array makes it very easy to represent things that have more than one dimension: for example, 3D objects with length, height, and width. The universe we live in could be described with four dimensions, time being the fourth dimension, so it is 4D. Higher levels like 5D and so forth are hard to imagine at first, but when you put it into practice, they turn out to be handy and not too hard!

Let's look at some more down-to-earth examples. A seat in the theater can be indicated with a 2D-array: one index for the row, and another for the number of the seat in that row. If you want to write a game that uses maps such as Sea Battle, two-dimensional arrays will be very helpful in setting coordinates on the map. Besides, some mathematical structures are conveniently represented as multidimensional arrays.

Let's take a look at a case of a multidimensional array that is used quite often in practice: a **two-dimensional array**.

2-dimensional arrays

If a one-dimensional array can be represented as a single sequence of elements, then an intuitive way of representing a two-dimensional array is a **matrix** or a **table**. If you're working with matrices or tables in your program, it makes sense to present them in the form of two-dimensional arrays.

Let's create a two-dimensional array of `int`s with 3 rows and 3 columns. Here is what it looks like:

```
int[][] twoDimArray = {
        {0, 0, 0}, // first array of ints
        {0, 0, 0}, // second array of ints
        {0, 0, 0}  // third array of ints
};
```

You can picture it as a table:

0	0	0
0	0	0
0	0	0

We can say that the arrays with three zero elements are **nested** in the `twoDimArray`. The main array that contains other arrays is called the **main** array.

Here's an interesting feature: nested arrays do not necessarily have to be of the same size. In the example below, each new embedded array has different lengths:

```
int[][] twoDimArray = {
        {0, 0},       // the length is 2
        {1, 2, 3, 4}, // the length is 4
        {3, 3, 3}     // the length is 3
};
```

You can create nested arrays with different numbers of elements in the same 2D array.

Access the elements

Let's see how we can access an element of an array. The idea is the same as for one-dimensional arrays, but now we have to write *two* indices: first the index of the element of the main array, and then the index of the nested array.

Suppose we need to access an element that is in the first row and the first column. How do we find this particular element? As you recall, a 2D array is an array of arrays. So, start by selecting one of the nested arrays by its index in the main array. The principle is similar to a 1D array.

```
int[][] twoDimArray = {
        {3, 4, 5},  // [0]
        {6, 7, 8},  // [1]
};
```

First, go to the main array and choose the nested array with its index:

twoDimArray[0]	3	4	5
twoDimArray[1]	6	7	8

Second, in this nested array, choose the required element with its index. This is also like in simple arrays:

twoDimArray[0][0]	twoDimArray[0][1]	twoDimArray[0][2]
3	4	5

Let's create a new variable `int number` and put in it the element of the first row and the first column of our array:

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

Remember that in all arrays, indexing starts with 0!

The following code will show all the elements of the two-dimensional array `twoDimArray`:

```
System.out.println(twoDimArray[0][0]); // 3
System.out.println(twoDimArray[0][1]); // 4
System.out.println(twoDimArray[0][2]); // 5
System.out.println(twoDimArray[1][0]); // 6
System.out.println(twoDimArray[1][1]); // 7
System.out.println(twoDimArray[1][2]); // 8
```

Working with 2D arrays

In the previous examples, we were creating 2D arrays by enumerating all the elements. But one of the most popular ways to create a multidimensional array is using a `for` loop.

Let's create `twoDimArray` with 2 rows and 10 columns and fill it with ones. To get access to every element we need to iterate through both `for` loops. The first `for` loop chooses the nested array and the second `for` loop iterates over all the elements of the nested array.

```
int[][] twoDimArray = new int[2][10];

for (int i = 0; i < twoDimArray.length; i++) {
    for (int j = 0; j < twoDimArray[i].length; j++) {
        twoDimArray[i][j] = 1;
    }
}
```

You can print all nested arrays:

```
for (int i = 0; i < twoDimArray.length; i++) {
     System.out.println(Arrays.toString(twoDimArray[i
}
```

The output will be:

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

And to print every element we also need two `for` loops. In the example below, we increase all the elements by one and print them to the standard output:

```
for (int i = 0; i < twoDimArray.length; i++) {
    for (int j = 0; j < twoDimArray[i].length; j++) {
        twoDimArray[i][j]++;
        System.out.print(twoDimArray[i][j] + " ");
    }
    System.out.println();
}
```

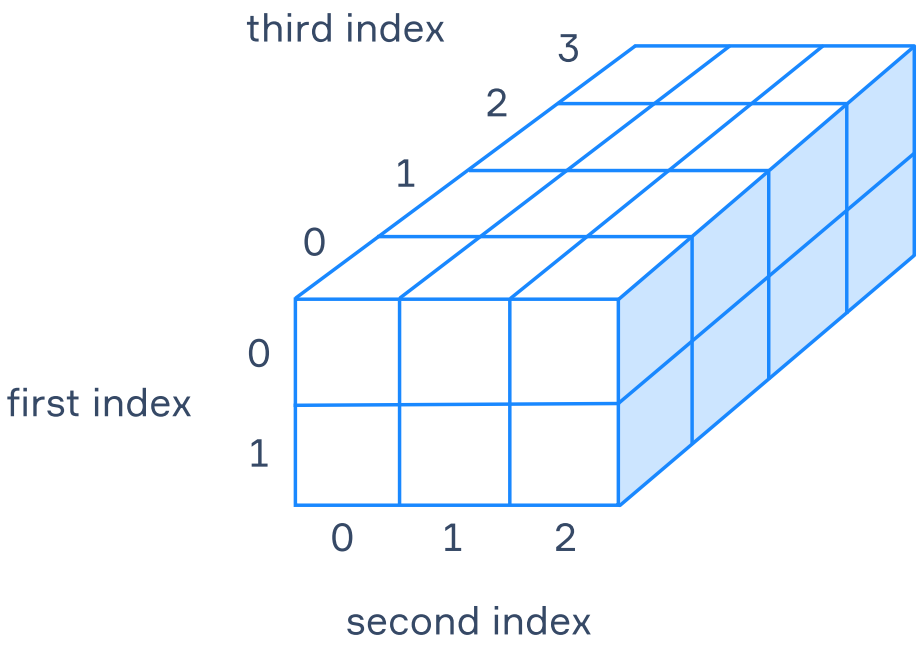
So, the output will be:

```
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
```

Multidimensional arrays (>2)

We are finally ready to deal with more complex concepts. There are arrays with more than two dimensions. Even though it is more difficult to understand what they look like, don't worry: you will get used to working with them.

First of all, a **three-dimensional** array can be represented as a cube or a box: it has exactly three dimensions — length, width, and height. Take a look at a three-dimensional array with 24 elements:



The following practical situation also can help you to understand three-dimensional arrays: imagine that you need to figure out where the car is in a multi-story parking lot. Then you have to set three numbers, or three coordinates: floor, row, and place in a row.

Last, but not least: you can imagine a three-dimensional array like this:

[0,0,0,0]	[1,1,1,1]	[2,2,2,2]
[3,3,3,3]	[4,4,4,4]	[5,5,5,5]

In each element of a two-dimensional array, you have another nested array.

The code below creates the three-dimensional array you just saw above:

```
int[][][] threeDimArray = new int[2][3][4];

int element = 0;

for (int i = 0; i < threeDimArray.length; i++) {
    for (int j = 0; j < threeDimArray[i].length; j++)
        for (int k = 0; k < threeDimArray[i][j].lengt
            threeDimArray[i][j][k] = element;
        }
        element++;
    }
}
```

Here `2` is the number of rows, `3` is the number of columns and `4` is the number of elements in a nested array.

And let's print the nested arrays:

```
for (int i = 0; i < threeDimArray.length; i++) {
    for (int j = 0; j < threeDimArray[i].length; j++)
```

```
            System.out.print(Arrays.toString(threeDimArra
        }
        System.out.println();
    }
```

The output will be:

```
[0, 0, 0, 0] [1, 1, 1, 1] [2, 2, 2, 2]
[3, 3, 3, 3] [4, 4, 4, 4] [5, 5, 5, 5]
```

Accordingly, in order to refer to an element of the three-dimensional array, we need *three* indices:

```
System.out.println(threeDimArray[0][0][0]); // 0
System.out.println(threeDimArray[0][1][0]); // 1
System.out.println(threeDimArray[1][0][1]); // 3
System.out.println(threeDimArray[1][2][3]); // 5 - th
```

Notice, that you can simplify your code by using *for-each* loops and methods of the class `Arrays` to fill and print multidimensional arrays.

And, of course, you can create arrays of other dimensions by analogy — 4, 5, 6, and so on. Just remember that an element of a multidimensional array has as many indices as dimensions of that array.

Conclusion

Let's recap. You have figured out what multidimensional arrays are and how to create them in Java. Here are the main points to take away:

- a multidimensional array is essentially an array of arrays;
- indexing starts with 0;
- to find an element of a multidimensional array, you need the number of indices equal to the array dimension;
- you can assemble arrays of different sizes in a multidimensional array.

Related topics

Topic prerequisites:

1. Iterating over arrays

Comments (26) Useful links (4) Show discussion