Java's nested Arrays

An array element can actually itself be an array. It's known as a nested array, or an array assigned to an outer array's element.

This is how Java supports two and three dimensional arrays of varying dimensions.

Two-Dimensional Array

A two-dimensional array can be thought of as a table or matrix of values with rows and columns.

You can use an array initializer for this, which I'm showing on this slide.

Array Initializer formatted over multiple lines

```
int[][] array = {
                        {1, 2, 3},
                        {11, 12, 13},
                        {21, 22, 23},
                        {31, 32, 33}
                        };
```

Array Initializer declared on one line

```
int[][] array = {{1, 2, 3}, {11, 12, 13}, {21, 22, 23}, {31, 32, 33}};
```

Two-Dimensional Array

Notice the two sets of square brackets on the left side of the assignment in the declaration.

Using this type of declaration tells Java we want a two dimensional array of integers.

Array Initializer formatted over multiple lines

```
int[][] array = {
                        {1, 2, 3},
                        {11, 12, 13},
                        {21, 22, 23},
                        {31, 32, 33}
                        };
```

Array Initializer declared on one line

```
int[][] array = {{1, 2, 3}, {11, 12, 13}, {21, 22, 23}, {31, 32, 33}};
```

Two-Dimensional Array

Here, I show the same declaration with array initializers that mean the same thing.

The first example just uses white space to make it more readable.

In this example, all the nested arrays have the same length.

Array Initializer formatted over multiple lines

```
int[][] array = {
                        {1, 2, 3},
                        {11, 12, 13},
                        {21, 22, 23},
                        {31, 32, 33}
                        };
```

Array Initializer declared on one line

```
int[][] array = {{1, 2, 3}, {11, 12, 13}, {21, 22, 23}, {31, 32, 33}};
```

Two-dimensional Array

A 2-dimensional array doesn't have to be a uniform matrix though.

This means the nested arrays can be different sizes, as I show with this next initialization statement.

```
int[][] array = {
                    {1, 2},
                    {11, 12, 13},
                    {21, 22, 23, 24, 25}
                    };
```

Here, we have an array with 3 elements.

Each element is an array of integers (a nested array).

Each nested array is a different length.

If you find that confusing, don't worry. It should all make sense shortly.

Two-dimensional Array

You can initialize a two-dimensional array and define the size of the nested arrays, as shown here.

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

This statement says we have an array of 3 nested arrays, and each nested array will have three ints.

Two-dimensional Array

The result of this initialization is shown in the table on this slide.

Java knows we want a 3x3 matrix of ints and defaults the values of the nested arrays to zeros, as it would for any array.

	0	**1**	**2**
0	0	0	0
1	0	0	0
2	0	0	0

Two-dimensional Array

You can initialize a two-dimensional array without specifying the size of the nested arrays.

Here, we're specifying only the outer array size by specifying the length, only in the first set of square brackets.

We've left the second set of square brackets empty.

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

The result of this initialization is an array of 3 null elements.

We are limited to assigning integer arrays to these elements, but they can be any length.

Two-dimensional Array

There are a lot of ways to declare a two-dimensional array.

I'll cover the two most common ways here.

The most common, and in my opinion, the clearest way to declare a two-dimensional array is to stack the square brackets, as shown in the first example.

```
int[][] myDoubleArray;
int[] myDoubleArray[];
```

You can also split up the brackets as shown in the second example, and you'll likely come across this in Java code out in the wild.

Accessing elements in multi-dimensional arrays

When we access a one dimensional array element, we do it with square brackets and an index value.

So this code sets the first element in the array to 50:

```
array[0] = 50;
```

To access elements in a two-dimensional array, we use two indices, so this code sets the first element in the first array to 50.

```
secondArray[0][0] = 50;
```

This next code sets the second element, in the second array to 10.

```
secondArray[1][1] = 10;
```

Accessing elements in multi-dimensional arrays

The code on this slide is similar to the code we have used in IntelliJ, using nested traditional for loops.

In this case, we're not using any local variables, but accessing array elements and variables directly.

```
for (int i = 0; i < array2.length; i++) {
    for (int j = 0; j < array2[i].length; j++) {        // while i = 0, j will loop from 0 to 3
        System.out.print(array2[i][j] + " ");
    }
}
```

Accessing elements in multi-dimensional arrays

This table shows the indices, which are used to access the elements in the two-dimensional array in our code sample.

When we loop through the outer loop, we're accessing each row of elements.

I've highlighted the first row, which would be the elements accessed, when i = 0 for the outer for loop.

When we loop through the inner loop, we're accessing each cell in the array.

A cell in this matrix can be any type.

In our code, each is an integer value, and we know they've all been initialized to zero.

	j = 0	j = 1	j = 2	j = 3
i = 0	[0][0]	[0][1]	[0][2]	[0][3]
i = 1	[1][0]	[1][1]	[1][2]	[1][3]
i = 2	[2][0]	[2][1]	[2][2]	[2][3]