Arrays class

1. `Arrays` class

As we noted previously, arrays are a very useful and frequently used in programming.

Java's creators very quickly noticed that Java programmers often write the same code when working with arrays. For example, copy part of an array to another array, code to fill each cell of an array with the same value, display the contents of an array in a convenient format, etc.

That's why they created the special `Arrays` 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 first we'll consider just 10 of them — the simplest and most often used.

---

2. `Arrays.toString()`

The first method we'll look at is called `Arrays.toString()`. But first, a little background.

Each array in Java has a `toString()` method, which returns a 'textual representation of the array'. You can get a textual representation of an array using the following statement:

```
String str = name.toString();
```

Where `name` is the name of the array variable, and `str` is the name of the variable that will store the string representation of the array.

But if you try to print the array to the screen using the `System.out.println(name)` method, you will most likely see something like this:

```
String str = Arrays.toString(name);
```

Examples:

`int[] array = {1, 2, 3};`
`String str = Arrays.toString(array);`
	The `str` variable will contain the string:
	`"[1, 2, 3]"`

`int[] array = {};`
`String str = Arrays.toString(array);`
	The `str` variable will contain the string:
	`"[]"`

`String[] array = {"Hi", "How's", "life?"};`
`String str = Arrays.toString(array);`
	The `str` variable will contain the string:
	`"[Hi, How's, life?]"`

3. `Arrays.deepToString()`

But if you try to convert a two-dimensional array to a string (to display it) `using the Arrays.toString()` method, you'll see something familiar:

```
[I@37afeb11, I@37afeb21, I@37afeb31]
```

This is all because the cells of a two-dimensional array store references to one-dimensional arrays. And how are one-dimensional arrays converted to a string? Exactly as you see above.

What can be done? How do we correctly display a two-dimensional array?

To this end, the `Arrays` class has another special method — `deepToString()`. Calling it looks like this:

```
String str = Arrays.deepToString(name);
```

This method can be passed arrays that are two-dimensional, one-dimensional, three-dimensional or, in general, any dimension, and it will always display the elements of the array.
Note: the `Arrays.deepToString()` method does not work with one-dimensional arrays of primitives (for example, `int[]`).

Examples:

`Integer[] array = {1, 2, 3};`
`String str = Arrays.deepToString(array);`
	The `str` variable will contain the string:
	`"[1, 2, 3]"`

`int[][] array = { {1, 1}, {2, 2}, {3, 3} };`
`String str = Arrays.deepToString(array);`
	The `str` variable will contain the string:
	`"[[1, 1], [2, 2], [3, 3]]"`

`int[][][] array = { {{1, 2, 3}, {1}}, {{}} };`
`String str = Arrays.deepToString(array);`
	The `str` variable will contain the string:
	`"[[[1, 2, 3], [1]], [[]]]"`

4. `Arrays.equals()`

We figured out how to display arrays on the screen, but what about comparing arrays?

To compare strings, we have `equals` and `equalsIgnoreCase` methods, but what methods do arrays have?

The good news is that arrays have an `equals` method. The bad news is that it doesn't compare the contents of the arrays. The `equals` method of arrays does the same thing as the == operator — it compares references. Examples:

Examples:

`int[] x1 = {1, 2, 3};`
`int[] x2 = {1, 2, 3};`
`boolean b = x1 == x2;`
	`false` (the references are not equal)

`int[] x1 = {1, 2, 3};`
`int[] x2 = {1, 2, 3};`
`x1.equals(x2);`
	The `equals` method of `arrays` simply compares the references of two arrays.
	
	`false` (the references are not equal)

What can be done? How do we compare arrays based on their contents?

And again the `Arrays` comes to our rescue, or more specifically, its `Arrays.equals()` method. This is how we call it:

```
Arrays.equals(name1, name2)
```

The method returns `true` if the arrays are of equal length and their elements are equal. Otherwise, it returns `false`.

Examples:

`int[] x1 = {1, 2, 3};`
`int[] x2 = {1, 2, 3};`
`x1.equals(x2);`
	The `equals` method of `arrays` simply compares the references of two arrays.
	
	`false` (the references are not equal)

`int[] x1 = {1, 2, 3};`
`int[] x2 = {1, 2, 3};`
`Arrays.equals(x1, x2);`
	`true` (the contents are equal)

`int[] x1 = {1, 2, 3};`
`int[] x2 = {1, 2, 3, 4};`
`Arrays.equals(x1, x2);`
	`false` (the contents of the arrays are different)

5. `Arrays.deepEquals()`

And, as you probably already guessed, the `Arrays.equals` method will not work correctly for two-dimensional arrays: it treats two-dimensional arrays like a one-dimensional array whose elements are addresses of one-dimensional arrays.

Thus, to correctly compare multidimensional arrays (`n = 1, 2, 3,...`), they came up with the `Arrays.deepEquals()` method. Calling it looks like this:

```
Arrays.deepEquals(name1, name2)
```

The method returns `true` if the arrays are of equal length and their elements are equal. Otherwise, it returns `false`. If the elements inside the array are also arrays, then the `Arrays.deepEquals()` method is used to compare them, and so on.

Examples:

`int[][] x1 = {{1, 2, 3}, {4, 5, 6}};`
`int[][] x2 = {{1, 2, 3}, {4, 5, 6}};`
`x1.equals(x2);`
	The `equals` method of `arrays` simply compares the references of two arrays.
	
	`false` (the references are not equal)

`int[][] x1 = {{1, 2, 3}, {4, 5, 6}};`
`int[][] x2 = {{1, 2, 3}, {4, 5, 6}};`
`Arrays.equals(x1, x2);`
	The `Arrays.equals` method will compare `x1` and `x2` as one-dimensional arrays that store references. They contain different references.
	`false` (the contents of the arrays are not equal)

`int[][] x1 = {{1, 2, 3}, {4, 5, 6}};`
`int[][] x2 = {{1, 2, 3}, {4, 5, 6}};`
`Arrays.deepEquals(x1, x2);`
	`true` (the contents are equal)