Theory: Array

18 minutes 0 / 5 problems solved

Skip this topic Start practicing

Introduction to arrays

When you need to process multiple objects of the same type, you can save them in an **array** and then process together as a single unit. It is a very convenient approach if you do not know how many objects the program will process during runtime.

You may consider an **array** as a collection of elements of the same type. All elements are stored in the memory sequentially.

The collection provides one name for its elements. The possible number of elements to be stored is established when the array is created and cannot be changed. But a stored element can be modified at any time.

The picture below illustrates an array of five floating-point numbers. Each element has an integer index (0-4) to be accessed.

Index	0	1	2	3	4
Element	10.8	14.3	13.5	12.1	9.7

An array of five floating-point elements

The first element has the index 0, the last element has the index equal **array size – 1**.

In Java, an array has the following important features:

- an array is a reference type;
- all array's elements are stored in the memory sequentially ;
- each element of the array is accessed by its numerical index, the first element has the **index 0**;
- the last element is accessed by the index equal to **array size – 1**;
- it is possible to create an array to store elements of any type.

Declaration, instantiation, initialization

To create an array filled with elements we should:

- declare a variable of an array type (declaration);
- create an instance of the array object (instantiation);
- initialize the array by some values (initialization).

When we declare a variable, we define its type and name. Instantiation happens when memory is allocated for this object. Initializing of the array object means that we put the certain values of the array object into the memory of our program.

To declare an array we must use two special characters **[]** after the name of the type of elements in the array:

```
int[] array; // declaration's form 1
```

or after the name of an array variable:

```
int array[]; // declaration's form 2: less used in practice
```

Next, we will use the first form of declaration because it is mostly used in practice.

Creating an array with the specified elements

Java provides several ways to create an array with the specified elements.

The simplest way to instantiate and initialize an array is to enumerate all its elements:

```
int[] numbers = { 1, 2, 3, 4 }; // instantiating and initializing an array of 1, 2, 3, 4
```

Another way is to initialize an array using variables:

```
int a = 1, b = 2, c = 3, d = 4;
int[] numbers = { a, b, c, d }; // instantiating and initializing an array of 1, 2, 3, 4
```

In this case, we should have all the elements at the moment of the array creation.

Creating an array using the keyword "new"

The most general way to create an array is to use the special keyword `new` and specify the necessary number of elements:

```
int n = ...; // n is a length of an array
int[] numbers = new int[n];
```

This form is useful when the number of elements is known before starting the program. When we create an instance of the array object with indicated length like `[n]` or `[5]` and don't enumerate its elements explicitly, the array is initialized with default values of its type.

Now, the array has `n` elements. Each element is equal to zero (the default value of the type Int). Next, we should make an explicit initialization of elements.

> The size of an array cannot be greater than `Integer.MAX_VALUE`. Actually, it is even slightly smaller than this value.

It's possible to separate declaration and instantiation in two lines:

```
int[] numbers; // declaration
numbers = new int[n]; // instantiation and initialization with default values
```

Also, we can write the keyword `new` and enumerate all elements of an array:

```
float[] floatNumbers; // declaration
floatNumbers = new float[] { 1.02f, 0.03f, 4f }; // instantiation and initialization
```

The length of an array

To obtain the length of an existing array, access the special property `arrayName.length`. Here is an example:

```
int[] array = { 1, 2, 3, 4 }; // an array of numbers

int length = array.length; // number of elements of the array

System.out.println(length); // 4
```

Accessing elements

The values of elements of an array can be changed. To set (get) a value to (from) array the index is used.

Set the value by the index:

```
array[index] = val;
```

Get the value by the index

```
val = array[index];
```

Indexes of an array have numbers from 0 to length – 1 inclusive.

Let's see an example.

```
int[] numbers = new int[3]; // numbers: [0, 0, 0]
numbers[0] = 1; // numbers: [1, 0, 0]
numbers[1] = 2; // numbers: [1, 2, 0]
numbers[2] = numbers[0] + numbers[1]; // numbers: [1, 2, 3]
```

This code works as follow:

- in the first line, the array of integer named numbers with three elements is created. It is initialized with default values, which is 0 for the int type;
- in the second line, the value "1" is assigned to the very first element of the array by its index (do not forget, the first element has the index 0);
- in the third line, the value "2" is assigned to the second element of the array by its index (numbers[1] is the second element);
- in the last line, the sum of the first two elements is assigned to the third element by its index.

If we try to access a non-existing element by an index then a runtime exception happens.

For instance, let's try to get the fourth element (with index 3) of the considered array `numbers`.

```
int elem = numbers[3];
```

The program throws `ArrayIndexOutOfBoundsException.`

Be careful while indexing elements of an array.

The utility class Arrays

If you need to process arrays, you can use standard methods grouped in the utility class `Arrays`.

- convert array to string using `Arrays.toString(array)` and then print it:

```
byte[] famousNumbers = { 0, 1, 2, 4, 8, 16, 32, 64 };
String arrayAsString = Arrays.toString(famousNumbers); // [0, 1, 2, 4, 8, 16, 32, 64]
System.out.println(arrayAsString);
```

- sorting a whole array or a part of it using `Arrays.sort(array)`:

```
long[] bigNumbers = { 200000000L, 400000000L, 100000000L, 300000000L }; // it's unsorted

Arrays.sort(bigNumbers); // sorting whole array

System.out.println(Arrays.toString(bigNumbers)); // [100000000, 200000000, 300000000, 400000000
```

- comparing arrays: two arrays are equal if they contain the same elements in the same order:

```
int[] numbers1 = { 1, 2, 5, 8 };
int[] numbers2 = { 1, 2, 5 };
int[] numbers3 = { 1, 2, 5, 8 };

System.out.println(Arrays.equals(numbers1, numbers2)); // it prints "false"
System.out.println(Arrays.equals(numbers1, numbers3)); // it prints "true"
```

- filling a whole array or a part of it by some values:

```
int size = 10;
char[] characters = new char[size];

// It takes an array, start index, end index (exclusive) and the value for filling the array
Arrays.fill(characters, 0, size / 2, 'A');
Arrays.fill(characters, size / 2, size, 'B');

System.out.println(Arrays.toString(characters)); // it prints [A, A, A, A, A, B, B, B, B, B]
```

Of course, the `Arrays` class contains a lot of other useful methods, including array copying, search in arrays, and so on. For details see here.