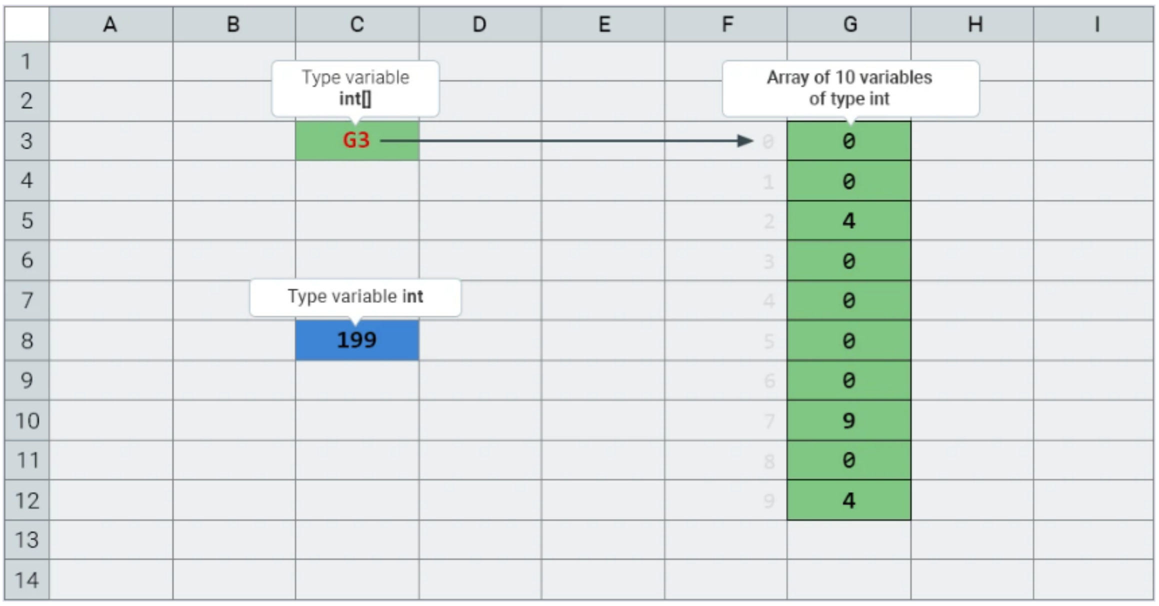
Nuances of working with arrays

1. Arrays in memory

In the previous examples, the illustrations were a little inaccurate.

When creating arrays (as when creating strings), two separate blocks of memory are allocated: one for storing the *array* (container) itself and a second block for the variable that stores its *address*. The picture below represents this clarification:



The memory allocated for the array of `10` `int` elements and the `int[]` variable that stores the *address* of the `int` array, is shown in green.

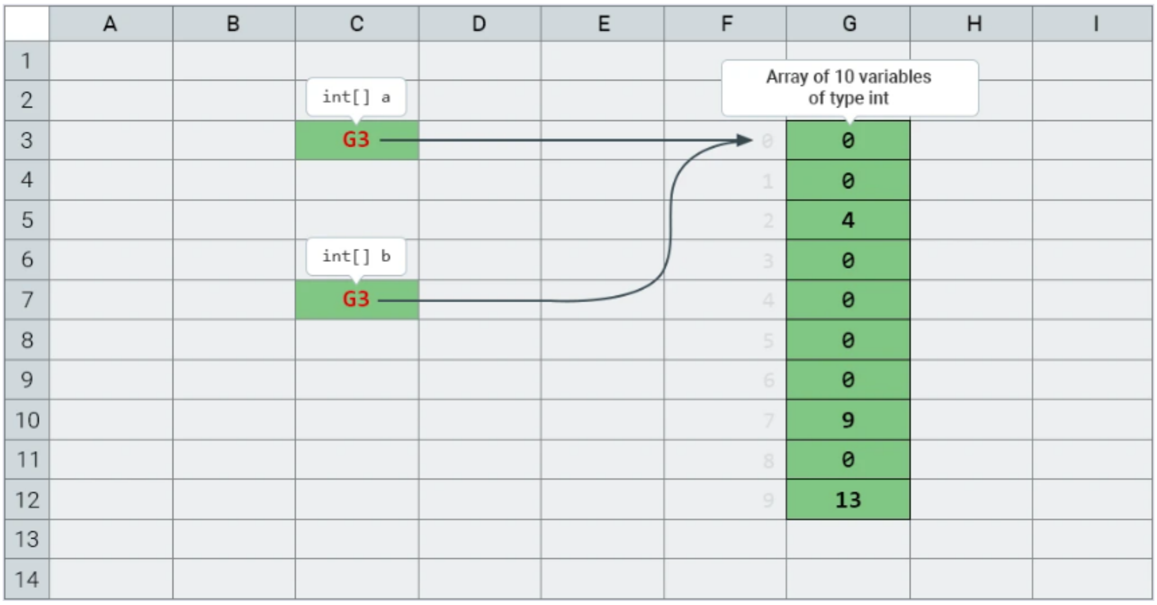
For comparison, an ordinary `int` variable storing the value 199 is shown in blue.

This is a little reminiscent of storing strings in memory, don't you think?

That's right, strings. And just as when you work with strings, you can assign array variables to one another:

Code	Explanation
`int[] a = new int[10];`	Create an array of `10` `int` elements.
`a[2] = 4;`	Assign the value `4` to the cell with index `2`.
`a[7] = 9;`	Assign the value `9` to the cell with index `7`.
`int[] b = a;`	In the `b` variable, save the address stored in the `a` variable. Now `a` and `b` point to the same array object in memory.
`a[9] = b[2] + a[7];`	In the array object's cell with index `9`, write the sum of the values that are stored in cells `2` (which stores the value `4`) and `7` (which stores the value `9`).

The array object stays right where it was, and the a and b variables store the same address (reference) to the same object. Look at the picture:



2. More details about working with arrays

You can create an array of elements of absolutely any type. To do this, simply write square brackets after the type name. In general, creating an array looks like this:

```
type[] name = new type[number];
```

Where type is the type of the elements we will store in the array. Name is the name of the variable we will use to refer to the array, and number is the number of cells in the array.

The example above is the canonical form for creating an array variable and array object. In reality, these are two separate entities.

You can create an array variable separately from an array object:

```
type[] name;
name = new type[number];
```

And one more point that is not insignificant:

You can use variables or even entire expressions as the `index array` and `number of array elements`.

Examples:

Code	Explanation

`int n = 100;`
`int[] a = new int[n];`
	Create an array of `n` elements

`int n = 100;`
`int[] a = new int[n * 2 + 3];`
	Create an array with `203` elements

`int n = 100;`
`int[] a = new int[n];`
`a[n-1] = 2;`
`a[n-2] = 3;`
`a[n/5] = a[n-1] + a[n-2]`
	`// a[99] = 2;`
	`// a[98] = 3;`
	`// a[20] = a[99] + a[98];`

Important:

By the way, you should be aware that if you try to access an array cell using an index that doesn't exist for the array (in our example, that means any integer not in the range `0..99`), then the program will crash with an `ArrayIndexOfBoundException`, meaning that the index was outside the bounds of the array.

3. Array length

As you saw in the previous example, you can create an array variable by itself and then later assign a value (a reference to an array object) to it somewhere in the code. You can even do this:

Code	Explanation

`int[] array;`
`if (a < 10)`
`    array = new int[10];`
`else`
`    array = new int[20];`
	Create an array variable whose type is `int[]`
	If the `a` variable is less than `10`,
	then create an array of `10` elements.
	Otherwise
	create an array of `20` elements

And now what else can you do with such an array? How do you know how many elements are in it?

To help with this, arrays have a special property (variable) named `length`. You can find the length of an array using this expression:

```
array.length;
```

Here `array` is the array variable's name and `length` is the name of the array's property. The value of the `length` property cannot be changed: the `length` property itself can be assigned to other variables, but nothing can be assigned to it (if you try to do this, the program simply will not compile).

We can continue with the previous example like this:

Code	Explanation
`int[] array;`	Create an array variable whose type is `int[]`
`if (a < 10)`	If the `a` variable is less than `10`,
`    array = new int[10];`	then create an array of `10` elements.
`else`	Otherwise
`    array = new int[20];`	create an array of `20` elements
`for (int i = 0; i < array.length; i++)`	Loop over all the elements of the array: from `0` to length `array.length - 1`
`{`	
`    System.out.println(array[i]);`	
`}`	

4. Summary of facts about arrays in Java

Let's recap what we know about arrays:

Fact 1. An array consists of many cells.

Fact 2. You access a specific cell by using its number (index).

Fact 3. All cells are of the same type.

Fact 4. The initial value of all cells is 0 (if the cells store numbers), `null` (if the cells store object references), or `false` (if the cells store `boolean` values). You will learn more about default values in this chapter.

Fact 5. `String[] list` is just the declaration of a variable. This does not create the container (array object) itself. To use the variable, you need to first create an array (container) and assign it to the variable. See the example below.

Fact 6. When we create an array object (container), we must indicate how large it is, i.e. how many cells it contains. This is done with a statement like: `new TypeName[n]`;

Fact 7. The length of an array can be found using the `.length` property.

Fact 8. After creating an array, you cannot change the type of its elements or the number of elements that it stores.

Code	Explanation

`String s;`
`String[] list;`
	`s` is `null`
	`list` is `null`

`list = new String[10];`
`int n = list.length;`
	The `list` variable stores a reference to an object: a string array consisting of `10` elements.
	`n` is `10`

`list = new String[0];`	Now `list` refers to an array of `0` elements. The array exists, but it can't store any elements.

`list = null;`
`System.out.println(list[1]);`
	An exception (program error) will be thrown, i.e. the program will crash. `list` stores a reference to `null`

`list = new String[10];`
`System.out.println(list[10]);`
	An array-out-of-bounds exception (program error) will be generated.
	If a `list` stores `10` elements/cells, then the valid indices are: `0` `1` `2` `3` `4` `5` `6` `7` `8` `9` — `10` elements.