Advanced array handling

I'd like to give you a brief overview about `String` arrays.

As we said previously, an array can be of any type. This means you can create an array of `String`s. If we wanted to write a program that "reads 10 lines from the keyboard and displays them in reverse order," here's what the code would look like:

```
Scanner console = new Scanner(System.in);
String[] array = new String[10];
for (int i = 0; i < 10; i++)
{
   array[i] = console.nextLine();
}
for (int i = 9; i >= 0; i--)
{
   System.out.println(array[i]);
}
```

Create a `Scanner` object
Create a 10-element array object
Loop from 0 to 9

Read a string from the keyboard and save it in the next cell of the array
Loop from 9 to 0

Display the next cell in the array

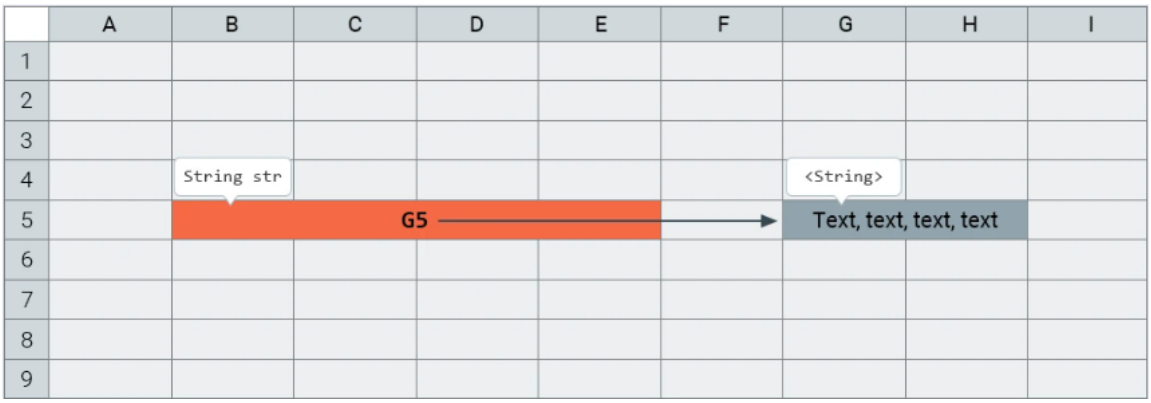
The code barely changed! We only had to replace `int` with `String` when creating the array. Well, and when reading the string from the keyboard, we also replaced the `nextInt()` method with `nextLine()`.

---

2. `String` array in memory

And one more useful fact. Let's consider 3 pictures:

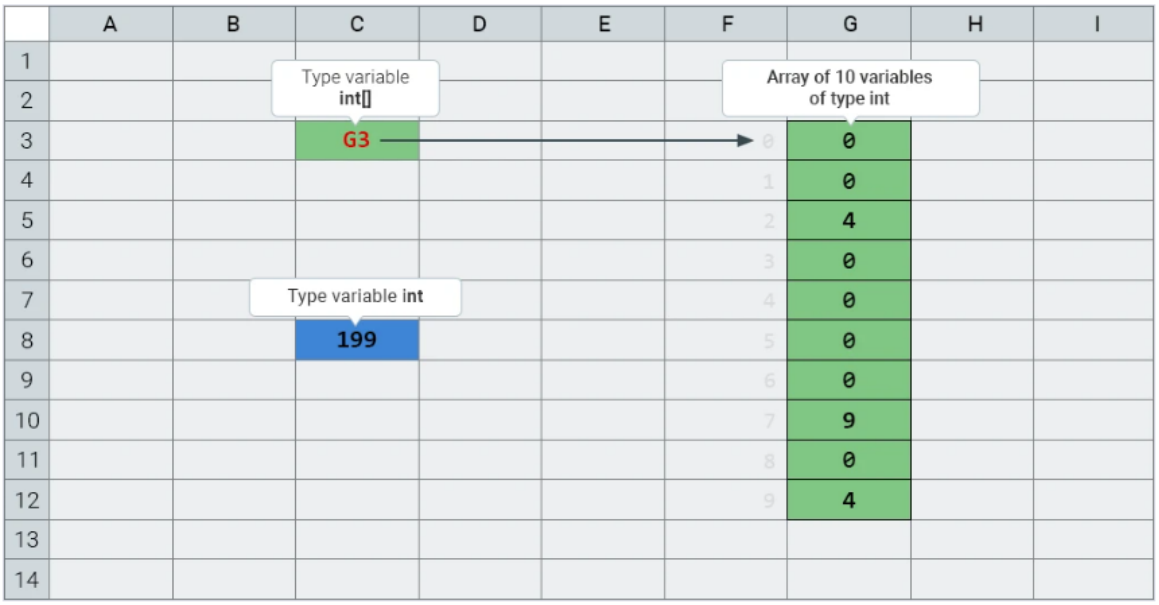
Picture 1. How a `String` object is arranged in memory:



This picture was taken from previous lessons.

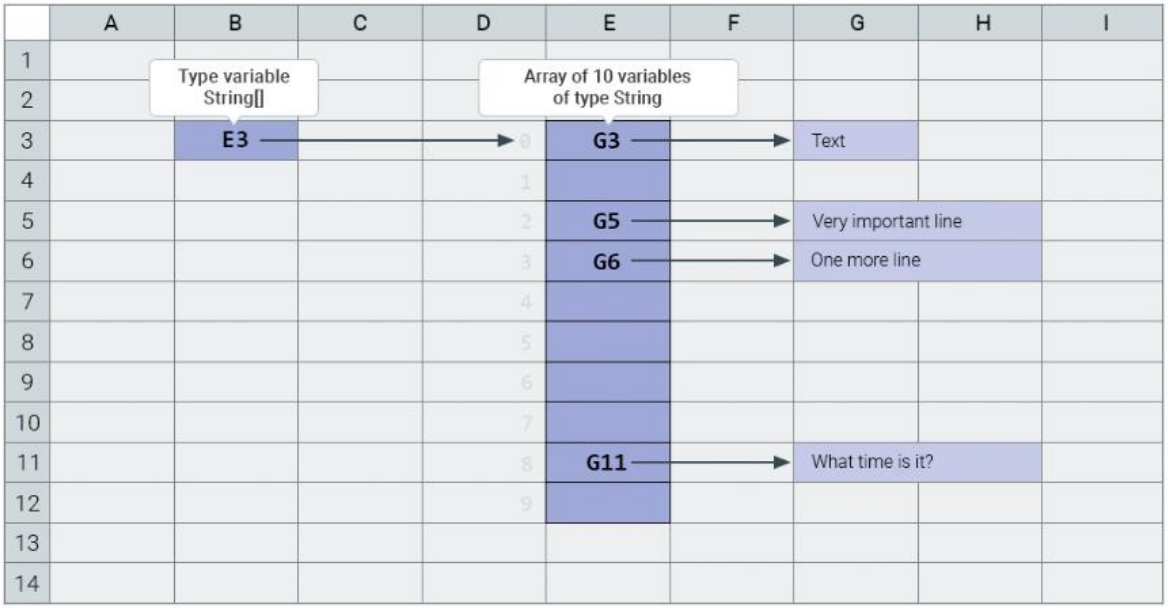
Please note that the text of the string is not stored directly in the variable: a separate block of memory is allocated for it. A `String` variable stores the address (reference) to an object that stores the text.

Picture 2. How an integer array is arranged in memory:



This picture is also familiar.

Picture 3. How a string array is arranged in memory:



On the left we see an array variable whose type is `String[]` (it stores the address of an array object).

In the middle, we have `String` array object itself.

And on the right are string objects that store some text.

The cells of the `String` array do not store the strings themselves (the text of the String objects). Instead, they store their addresses (references to them). In the same way that `String` variables store the addresses of string objects (where the text is stored).

Take this into consideration when you **compare array cells:**

`String[] array = new String[10];`	Create an array of 10 strings
`array[1] = "Hello";`	Put values into the array
`array[2] = array[1];`	
`array[3] = new String("Hello");`	
`// Compare`	
`array[1] == array[2];`	`true` (the references are equal)
`array[1] == array[3];`	`false` (the references are not equal)
`array[1].equals(array[3]);`	`true` (the strings are equal)
`array[1].equalsIgnoreCase(array[3]);`	`true` (the strings are still equal)

3. Fast array initialization in Java

Arrays are super useful, so Java's creators tried to make working with them as convenient as possible.

The first thing they did was to simplify array initialization, the process by which you supply the initial values of an array.

After all, in addition to data read from somewhere, a program also quite often needs its own internal data in order to work. For example, suppose we need to store the lengths of each month in an array. This is what the code might look like:

```
int[] months = new int[12];
months[0] = 31; // January
months[1] = 28; // February
months[2] = 31; // March
months[3] = 30; // April
months[4] = 31; // May
months[5] = 30; // June
months[6] = 31; // July
months[7] = 31; // August
months[8] = 30; // September
months[9] = 31; // October
months[10] = 30; // November
months[11] = 31; // December
```

But thanks to Java's creators, there is a way to write this more concisely:

```
// Lengths of months of the year
int[] months = new int[] { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

You can simply list all the values of the array, separated by commas!

Convenient, right? But that's not all.

As it happens, the compiler can determine the type of the container (array object) based on the type of the values of the array. And to determine the length of the array, it's trivial to count the number of elements written in the curly braces.

That means this code can be written even shorter:

```
// Lengths of months of the year
int[] months = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

.A thing of beauty, isn't it? 😳

This is called "fast array initialization". By the way, this works for types other than `int`...

```
// Names of months of the year
String[] months = {"January", "February", "March", "April",
"May", "June", "July", "August", "September", "October",
"November ", "December"};
```