Structure of the String class

1. Structure of the `String` class

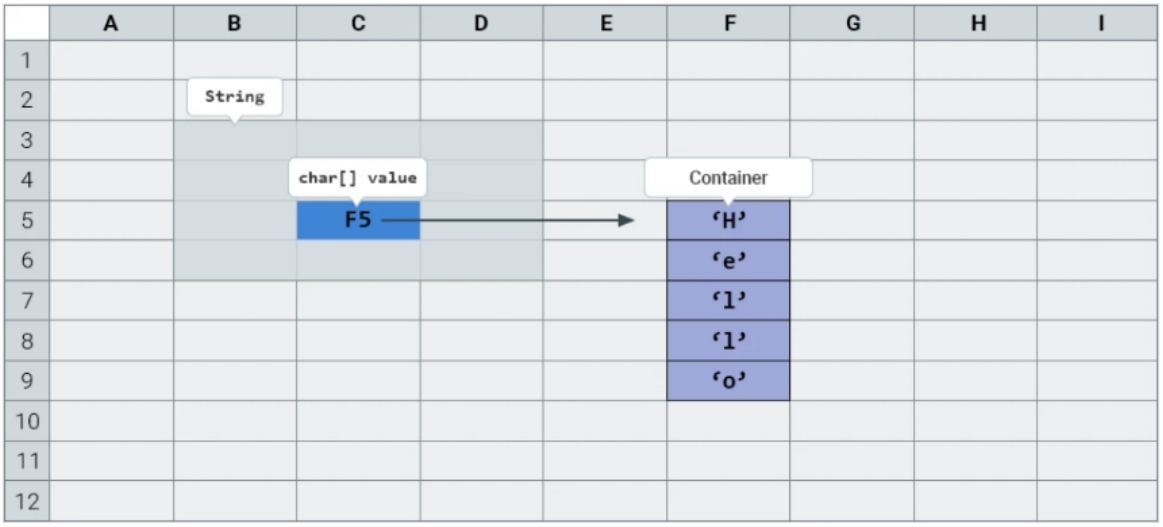
Today we will talk about the `String` class. After ints, the String class is the most popular class in Java. It is used absolutely everywhere. It has a bunch of useful methods that you are better off knowing.

The `String` class is the only class other than primitive types whose literals can be used in a `switch` statement; the compiler handles string addition and String objects in a special way; `String` objects are storied in memory in a special way. Basically, the `String` class is a very special class.

Also, the `String`class has a bunch of helper classes whose purpose is to further simplify working with strings in Java. When you learn all this, many thing will really become much easier for you to do. Well, we'll start from the very core of this ecosystem — the organization of the `String` class.

Array of characters

The structure of the `String` class is actually very simple: inside it is a character array (char array) that stores all the characters of the string. For example, this is how the word 'Hello' is stored:



Important!

Actually, this is not quite accurate. Because the `String` class is very important, it uses a lot of optimizations, and the data is internally stored not as a character array, but simply as a byte array.

2. Methods of the `String` class

The `String` class has a lot of methods: it has 18 constructors alone! So below we only mention the most basic of them:

Methods	Description
`int length()`	Returns the number of characters in the string
`boolean isEmpty()`	Checks whether the string is an empty string
`boolean isBlank()`	Checks that the string contains only whitespace characters: space, tab, new line, etc.
`char charAt(int index)`	Returns the character at the index position in the string.
`char[] toCharArray()`	Returns an array of the characters (a copy) that make up the string
`byte[] getBytes()`	Converts a string to a set of bytes and returns the array of bytes.
`String[] split(String regex)`	Splits a string into multiple substrings.
`String join(CharSequence delimiter, elements)`	Joins multiple substrings together
`String intern()`	Puts a string into the `string pool`.

Let's write a program that converts a file path from Unix style to Windows style. Unix uses the `/` character to separate folders, while Windows uses the `\` character.

Solution 1: using a char array

Code	Notes
`Scanner console = new Scanner(System.in);` `String path = console.nextLine();`	Create a Scanner object Read a line from the console

`char[] chars = path.toCharArray();`
`for (int i = 0; i < chars.length; i++)`
`  if (chars[i] == '/')`
`  chars[i] = '\\';`
	Convert a string to a character array
	Loop over the characters
	If the character is `/`,
	replace it with `\`. Don't forget about escaping.

`String result = new String(chars);` `System.out.println(result);`	Create a new string based on the character array. Display the string.

Solution 2: — using the `split()` and `join()` methods

Code	Notes

`Scanner console = new Scanner(System.in);`
`String path = console.nextLine();`
 	
`String[] array = path.split("/");`
 	
`String result = String.join("\\", array);`
 	
`System.out.println(result);`
	Create a Scanner object
	Read a line from the console
	
	Convert string to an array of strings. The `/` character is used as a separator (the extra two slashes are the result of double escaping).
	Concatenate all the strings in the array of strings. The `\` is used as a separator (we see it escaped).
	
	Display the string.

Solution 3: — using the `replace(char oldChar, char newChar)` method

Code	Notes

`Scanner console = new Scanner(System.in);`
`String path = console.nextLine();`
 	
`String result = path.replace('/', '\\');`
 	
`System.out.println(result);`
	Create a Scanner object
	Read a line from the console
	
	Simply replace one character with another
	(the second is escaped)
	Display the string.