Working with strings in Java

Module 1. Java Syntax
Level 10, Lesson 4

AVAILABLE

1. Comparing strings

One of the most common operations with strings is comparison. The String class has over ten different methods that are used to compare one string with another string. Below we will take a look at seven of the main ones.

Methods	Description
`boolean equals(String str)`	Strings are considered equal if all of their characters match.
`boolean equalsIgnoreCase(String str)`	Compares strings, ignoring the case of letters (ignores whether they are uppercase or lowercase)
`int compareTo(String str)`	Compares strings lexicographically. Returns 0 if the strings are equal. The return value is less than zero if the current string is less than the string parameter. The return value is greater than if the current string is greater than the string parameter.
`int compareToIgnoreCase(String str)`	Compares strings lexicographically while ignoring case. Returns 0 if the strings are equal. The return value is negative if the current string is less than the string parameter. The return value is greater than if the current string is greater than the string parameter.
`boolean regionMatches(int toffset, String str, int offset, int len)`	Compares parts of strings
`boolean startsWith(String prefix)`	Checks whether the current string starts with the string `prefix`
`boolean endsWith(String suffix)`	Checks whether the current string ends with the string `suffix`

Let's say you want to write a program that asks the user for a path to a file and then checks the file type based on its extension. The code of such a program might look something like this:

Code	Notes
`Scanner console = new Scanner(System.in);`	Create a `Scanner` object
`String path = console.nextLine();`	Read a line from the console
`if (path.endsWith(".jpg") || path.endsWith(".jpeg"))`	Check whether the string `path` ends with the given string
`{`	
`   System.out.println("This is a jpeg!");`	
`}`	
`else if (path.endsWith(".htm") || path.endsWith(".html"))`	
`{`	
`   System.out.println("This is an HTML page");`	
`}`	
`else if (path.endsWith(".doc") || path.endsWith(".docx"))`	
`{`	
`   System.out.println("This is a Word document");`	
`}`	
`else`	
`{`	
`   System.out.println("Unknown format");`	
`}`	

2. Searching for substrings

After comparing strings, the second most popular operation is finding one string inside another. The String class also has a few methods for this:

Methods	Description
`int indexOf(String str)`	Searches for the string `str` in the current string. Returns the index of the first character of the first occurrence.
`int indexOf(String str, int index)`	Searches for the string `str` in the current string, skipping the first `index` characters. Returns the index of the occurrence.
`int lastIndexOf(String str)`	Searches for the string `str` in the current string, starting from the end. Returns the index of the first occurrence.
`int lastIndexOf(String str, int index)`	Searches for the string `str` in the current string from the end, skipping the first `index` characters.
`boolean matches(String regex)`	Checks whether the current string matches a pattern specified by a regular expression.

The `indexOf(String)` and `indexOf(String, index)` methods are often used in combination. The first method lets you find the first occurrence of the passed substring in the current string. And the second method lets you find the second, third, etc. occurrences by skipping the first index characters.

Suppose we have a url like "**https://domain.com/about/reviews**", and we want to replace the domain name with "**codegym.cc**". Urls can have all sorts of different domain names, but we know the following:

- The domain name is preceded by two forward slashes — "`//`"
- The domain name is followed by a single forward slash — "`/`"

Here's what the code for such a program would look like:

Code	**Notes**
`Scanner console = new Scanner(System.in);`	Create a Scanner object
`String path = console.nextLine();`	Read a line from the console
`int index = path.indexOf("//");`	Get the index of the first occurrence of the string "`//`"
`int index2 = path.indexOf("/", index + 2);`	We get the index of the first occurrence of the string `/`, but look only after an occurrence of the characters `//`.
`String first = path.substring(0, index + 2);`	We get the string from the beginning to the end of the characters `//`
`String last = path.substring(index2);`	We get the string from `/` to the end.
`String result = first + "codegym.cc" + last;`	We concatenate the strings and the new domain.
`System.out.println(result);`	

The `lastIndexOf(String)` and `lastIndexOf(String, index)` methods work the same way, only the search is performed from the end of the string to the beginning.

3. Creating substrings

In addition to comparing strings and finding substrings, there is another very popular action: getting a substring from a string. As it happens, the previous example showed you a `substring()` method call that returned part of a string.

Here is a list of 8 methods that return substrings from the current string:

Methods	Description
`String substring(int beginIndex, int endIndex)`	Returns the substring specified by the index range `beginIndex..endIndex`.
`String repeat(int count)`	Repeats the current string n times
`String replace(char oldChar, char newChar)`	Returns a new string: replaces the character `oldChar` with the character `newChar`
`String replaceFirst(String regex, String replacement)`	Replaces the first substring, specified by a regular expression, in the current string.
`String replaceAll(String regex, String replacement)`	Replaces all substrings in the current string that match the regular expression.
`String toLowerCase()`	Converts the string to lowercase
`String toUpperCase()`	Converts the string to uppercase
`String trim()`	Removes all spaces at the beginning and end of a string

Here is a summary of the available methods:

`substring(int beginIndex, int endIndex)` method

The `substring` method returns a new string that consists of characters in the current string, starting at the character with index `beginIndex` and ending at `endIndex`. As with all intervals in Java, the character with index `endIndex` is not included in the interval. Examples:

Code	Result
`"Hellos".substring(0, 3);`	`"Hel"`
`"Hellos".substring(1, 4);`	`"ell"`
`"Hellos".substring(1, 6);`	`"ellos"`
`"Hellos".substring(1);`	`"ellos"`

If the `endIndex` parameter is not specified (which is possible), then the substring is taken from character at beginIndex to the end of the string.

`repeat(int n)` method

The repeat method simply repeats the current string `n` times. Example:

Code	Result
`"Hello".repeat(3);`	`"HelloHelloHello"`
`"Hello".repeat(2);`	`"HelloHello"`
`"Hello".repeat(1);`	`"Hello"`
`"Hello".repeat(0);`	`""`

`replace(char oldChar, char newChar)` method

The `replace()` method returns a new string in which all characters `oldChar` are replaced with the character `newChar`. This does not change the length of the string. Example:

Code	Result
`"Programming".replace('Z', 'z');`	`"Programming"`
`"Programming".replace('g', 'd');`	`"Prodrammind"`
`"Programming".replace('a', 'e');`	`"Progremming"`
`"Programming".replace('m', 'w');`	`"Prograwwing"`

`replaceFirst()` and `replaceAll()` methods

The `replaceAll()` method replaces all occurrences of one substring with another. The `replaceFirst()` method replaces the first occurrence of the passed substring with the specified substring. The string to be replaced is specified by a regular expression. We will delve into regular expressions in the *Java Multithreading* quest.

Examples:

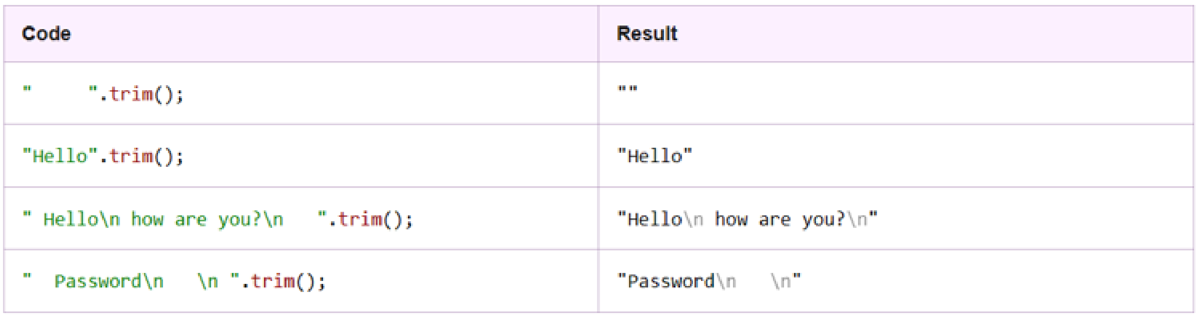
Code	Result
`"Good news everyone!".replaceAll("e.", "EX");`	`"Good nEXs EXEXyonEX"`
`"Good news everyone!".replaceAll("o.", "-o-");`	`"G-o-d news every-o-e!"`
`"Good news everyone!".replaceFirst("e.", "EX");`	`"Good nEXs everyone!"`
`"Good news everyone!".replaceFirst("o.", "-o-");`	`"G-o-d news everyone!"`

`toLowerCase() and toUpperCase()` methods

We got to know these methods when we first learned about calling the methods of the `String` class.

`trim()` method

The `trim()` method removes leading and trailing spaces from a string. Does not touch spaces that are inside a string (i.e. not at the beginning or end). Examples:



Code	Result
`"     ".trim();`	`""`
`"Hello".trim();`	`"Hello"`
`" Hello\n how are you?\n   ".trim();`	`"Hello\n how are you?\n"`
`"  Password\n   \n ".trim();`	`"Password\n   \n"`