Learning Objectives: String Methods

- **Identify the functions and applications of the following string methods:**
  - trim()
  - replace()
  - startsWith()
  - indexOf()
  - toUpperCase()
  - toLoweCase()
  - valueOf()
  - concat()

Trim & Replace

The trim() Method

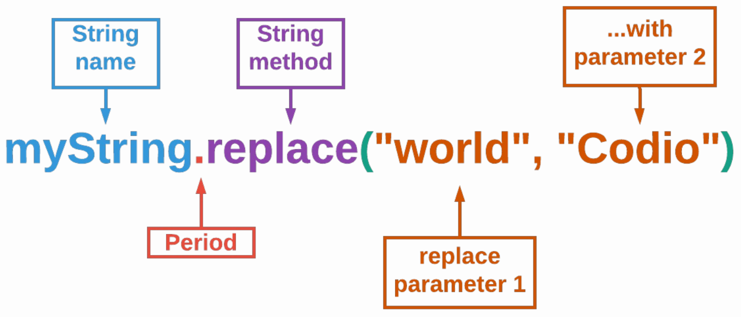
The `trim()` method removes white space characters from the beginning and end of a string. `trim()` will return a modified copy of the original string.

```
String myString = "     Hello     world     ";
System.out.println(myString.trim());
```

Note that the `trim()` method only removes leading and trailing white spaces and not the ones that are located in between characters.

The replace() Method

Contrary to the `trim()` method, the `replace()` method is much more flexible and can replace whites spaces and characters **anywhere** within the string. To use the `replace()` method, specify the characters you want to be replaced as the first parameter and the new characters as the second parameter. This will cause the second parameter characters to replace all occurrences of the first parameter characters.



.guides/img/StringReplace

The image above showcases the usage of the `replace()` method where the string `Codio` will replace all occurrences of the string `world`.

Let’s try some of the examples.

```
String string1 = "    Hello    world    ";
String string2 = string1.replace(" ", "");
//replace all whitespaces with no spaces
//string2 now becomes "Helloworld"

String string3 = "Codio";
String string4 = string1.replace("world", "Codio");
//replace all occurences of "world" with "Codio"
//string 4 now becomes "HelloCodio"

System.out.println(string2); //print string2
```

challenge

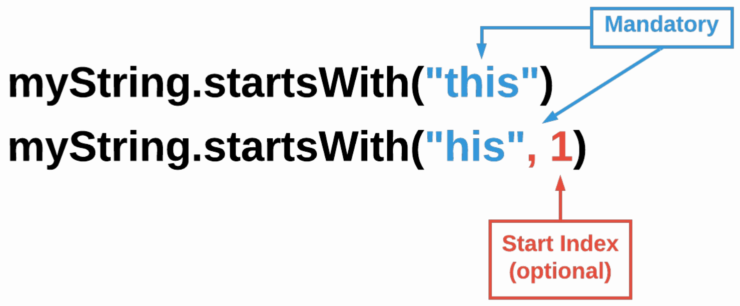
What happens if you:

- Change `String string2 = string1.replace(" ", "");` in the code to `String string2 = string1.replace("Hello", "Hi");`?
- Change the print statement to `System.out.println(string4);`?
- Add `String string5 = string4.replace(" ", "");` to the line **before** the print statement and change the print statement to `System.out.println(string5);`?
- Add another string `String string6 = string5.replace("oC", "o C");` **before** the print statement and change the print statement to `System.out.println(string6);`? **Note:** There is a whitespace between the `o` and the `C`.

Starts With

The startsWith() Method

The `startsWith()` method returns either `true` or `false` depending on whether a string starts a specific character(s). For example, `myString.startsWith("this")` will return `true` if `myString` starts with `"this"`. If not, it will return `false`. The `startsWith()` method has two parameters. The first parameter, a string, is mandatory. `startsWith` will start the comparison with the first character in the string by default. However, you can change where the comparison starts and ends with an optional second parameter. **Remember** that string indices start at `0`, which corresponds with the first letter in the string!



.guides/img/StringStartsWith

▼ The endsWith() method

You can use the `endsWith()` method to see if a string ends with a specific character(s). Unlike the `startsWith()` method, there is no optional second parameter. Here is an example, given the string `this is a string`, `myString.endsWith(“ring”)` will return `true`.

```
String myString = "this is a string";
boolean myBool = myString.startsWith("this");

System.out.println(myBool);
```

challenge

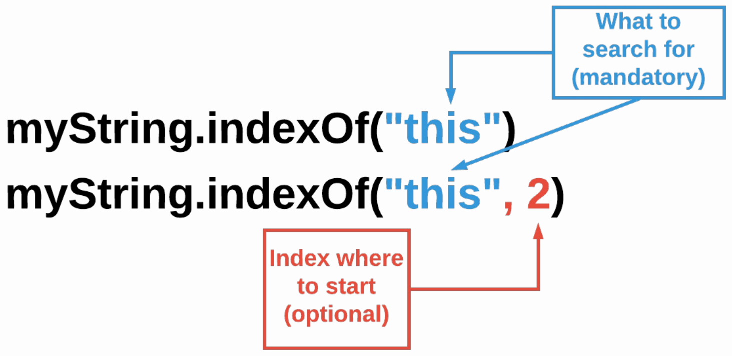
What happens if you:

- Set `myBool` to `myString.startsWith("This")`?
- Set `myBool` to `myString.startsWith("is", 2)`?
- Set `myBool` to `myString.startsWith("is", 4)`?
- Set `myBool` to `myString.startsWith("is", 5)`?

Index Of

The indexOf() Method

Similar to the `startsWith()` method, the `indexOf()` method searches a specific character in a string. The difference, however, is that `indexOf()` returns an index number, not boolean. If the word or character is found, the index of the first letter of the occurring character will be returned. If not, `-1` is returned. You can tell `indexOf()` where to start the search by specifying an index as an optional second parameter. By default, `indexOf()` will search the entire string.


What to search for (mandatory)
myString.indexOf("this")
myString.indexOf("this", 2)
Index where to start (optional)

.guides/img/StringIndexOf

▼ **The lastIndexOf() method**

You can use the `lastIndexOf()` method to search for a specific character(s) in a string that occurs last. Here is an example, given the string `this is his string`, `myString.lastIndexOf(“his”)` will return `8`. If you don’t want the system to search the whole string, you can specify an index as a second parameter to direct the system where to start searching.

```
String string1 = "The brown dog jumps over the lazy fox.";
String string2 = "brown";

System.out.println(string1.indexOf(string2));
```

challenge

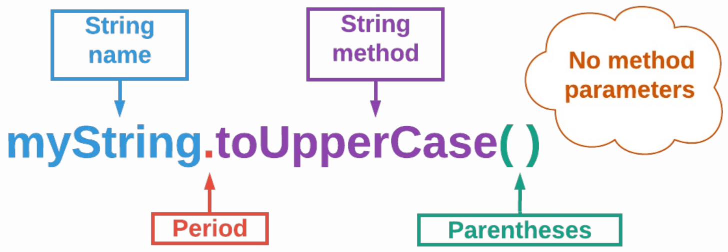
What happens if you:

- Set `string2` to `"zebra"`?
- Change `string2` back to `"brown"` and change the `print` statement to `System.out.println(string1.indexOf(string2, 10))`?
- Set `string2` to `"he"` and change the `print` statement to `System.out.println(string1.indexOf(string2, 4))`?

Uppercase & Lowercase

The toUpperCase() Method

The `toUpperCase()` method returns a copy of the original string with all uppercase characters. **Note** that there is no parameter for this method.



.guides/img/StringToUpperCase

Translation: Convert all the characters of `myString` to uppercase.

```
String myString = "the big brown dog";

System.out.println(myString.toUpperCase());
```

challenge

What happens if you:

- Set `myString` to `"ThE bIg BrOwN dOg"`?
- Set `myString` to `"THE BIG BROWN DOG"`?
- Set `myString` to `"123!@#"`?

The toLowerCase() Method

The `toLowerCase()` method creates a copy of a string, and returns the copy with all lowercase characters. Like the `toUpperCase()` method, the `toLowerCase()` does not take any parameters either.

```java
String myString = "THE BIG BROWN DOG";

System.out.println(myString.toLowerCase());
```

challenge

What happens if you:

- Set `myString` to `"tHe BiG bRoWn DoG"`?
- Set `myString` to `"the big brown dog"`?
- Set `myString` to `"214%#%"`?

Value Of

The valueOf() Method

In a previous module, you were introduced to the `valueOf()` method. The `valueOf()` method returns a string of a specified data type. To use this method, you'll need the keyword `String`, followed by `valueOf`, followed by the data you want to convert in parentheses `()`.

```
int a = 5;
String b = "Five";

System.out.println(String.valueOf(a) + b);
```

challenge

What happens if you:

- Replace `int a = 5;` with `double a = 5.5;`?
- Replace `double a = 5.5;` in your current code with `boolean a = true;`?
- Replace `System.out.println(String.valueOf(a) + b);` in your current code with `System.out.println(a + b);`?

Note that if you do not convert the data appropriately to strings, you will not be able to combine them!

Concat

The concat() Method

An alternative way to **concatenate** or combine strings is to use the `concat()` method. The `concat()` method works in the same way as adding literal strings together using the `+` operator.

```
String a = "High";
String b = " Five";

System.out.println(a.concat(b));
```

challenge

What happens if you:

- Change the print statement to `System.out.println(a.concat(b + "!"));`?
- Replace `String b = " Five";` with `int b = 5;`?
- Change the print statement back to `System.out.println(a.concat(b));`?

important

IMPORTANT

NOTE that the `concat()` method is exclusively for strings. Thus, you cannot include other data types like `int`s when using `concat()` unless they are converted to strings first. In the example above, the statement `System.out.println(a.concat(b + "!"));` causes the system to internally convert `b`, which is an `int`, into a string `"5"`. This conversion makes it possible to add `a` and `b` together.