Comparing strings by content

1. Comparing strings

This is all well and good. But you can see that the `s1` and `s2` strings are actually the same, meaning that they contain the same text. When comparing strings, how do you tell the program to look not at the addresses of `String` objects, but at their content?

To help us with this, Java's `String` class has the `equals` method. Calling it looks like this:

```
string1.equals(string2)
```

Comparing two strings

This method returns `true` if the strings are the same, and `false` if they are not the same.

Example:

Code	Note

`String s1 = "Hello";`
`String s2 = "HELLO";`
`String s3 = s1.toUpperCase();`
 	
`System.out.println(s1.equals(s2));`
`System.out.println(s1.equals(s3));`
`System.out.println(s2.equals(s3));`
	`// Hello`
	`// HELLO`
	`// HELLO`
	
	`false // They are different`
	`false // They are different`
	`true // They are the same, even though the addresses are different`

More examples:

Code	Explanation
`"Hello".equals("HELLO")`	`false`
`String s = "Hello";` `"Hello".equals(s);`	`true`
`String s = "Hel";` `"Hello".equals(s + "lo");`	`true`
`String s = "H";` `(s + "ello").equals(s + "ello");`	`true`

2. Case-insensitive string comparison

In the last example, you saw that the comparision `"Hello".equals("HELLO")` yields `false`. Indeed, the strings are not equal. But...

Clearly, the strings are not equal. That said, their content has the same letters and only differs by the case of the letters. Is there any way to compare them and disregard the case of the letters? That is, so that `"Hello".equals("HELLO")` yields `true`?

And the answer to this question is yes. In Java, the `String` type has another special method: `equalsIgnoreCase`. Calling it looks like this:

```
string1.equalsIgnoreCase(string2)
```

The name of the method translates roughly as *compare but ignore case*. The letters in the method's name include two vertical lines: the first is a lowercase `L`, and the second is an uppercase `i`. Don't let that confuse you.

Example:

Code	Note

`String s1 = "Hello";`
`String s2 = "HELLO";`
`String s3 = s1.toUpperCase();`
 	
`System.out.println(s1.equalsIgnoreCase(s2));`
`System.out.println(s1.equalsIgnoreCase(s3));`
`System.out.println(s2.equalsIgnoreCase(s3));`
	`// Hello`
	`// HELLO`
	`// HELLO`
	
	`true`
	`true`
	`true`

3. Example of string comparison

Let's give just one simple example: suppose you need to enter two lines from the keyboard and determine whether they are the same. This is what the code will look like:

```
Scanner console = new Scanner(System.in);
String a = console.nextLine();
String b = console.nextLine();
String result = a.equals(b) ? "Same" : "Different";
System.out.println(result);
```

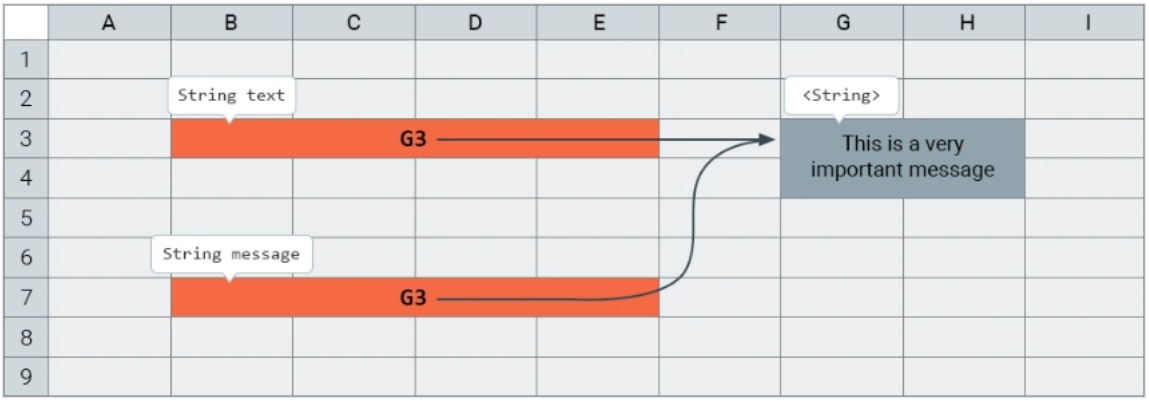
4. An interesting nuance of string comparison

There is one important nuance that you need to be aware of.

If the *Java compiler* finds multiple identical strings in your code (specifically in your code), then it will create only a single object for them in order to save memory.

```
String text = "This is a very important message";
String message = "This is a very important message";
```

And here's what memory will contain as a result:



And if you compare `text == message` here, then you get `true`. So don't be surprised by that.

If for some reason you really need the references to be different, then you can write this:

```
String text = "This is a very important message";
String message = new String ("This is a very important message");
```

Or this:

```
String text = "This is a very important message";
String message = new String (text);
```

In both of these cases, the `text` and `message` variables point to different objects that contain the same text.