Escaping characters

1. Reasons for escaping characters

Once upon a time, you learned that in order to write a string of characters in code, you need to wrap them in double quotes. The result is a *string literal*.

But what do we do if we need quotation marks inside a string literal? A string containing quotes — what could be easier?

Let's say we want to display the text `"Friends" was nominated for an "Oscar"`. How do you do that?

Code	Notes
`String s = ""Friends" was nominated for an "Oscar"";`	This option will not work!

The issue is that the compiler thinks you are writing completely unexpected code:

Code	Notes
`String s = ""Friends" was nominated for an "Oscar"";`	This option will not work!

After the compiler encounters double quotes in the code, it treats what follows as the beginning of a string literal. The next double quotation mark indicates the end of the string literal.

So how do you write double quotes inside of a literal?

---

2. Escaping characters

There is a way. It is called *escaping characters*. You just write the quotation marks within the string of text. And before the quotes, you add the \ (*backslash*) symbol.

This is what the string literal looks like when written properly:

Code	Notes
`String s = "\"Friends\" was nominated for an \"Oscar\"";`	This will work!

The compiler will interpret everything correctly and will not consider the quotation mark after the *backslash* as a normal quotation mark.

What's more, if you output this string to the screen, the quotes with *backslashes* will be processed correctly, and the text will be displayed without any backslashes: `"Friends" was nominated for an "Oscar"`

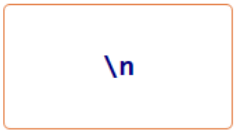
Another important point. A quotation mark preceded by a backslash represents a single character: we're simply using slick notation that doesn't interfere with the compiler's ability to recognize string literals in our code. You can assign quotes to a `char` variable:

Code	Notes
`char c = '\"';`	`\"` is one character, not two
`char c = '"';`	This is also possible: a double quotation mark inside single quotes

3. Common situations that occur when escaping characters

In addition to double quotes, there are many other characters that the compiler handles in a special way. For example, a line break.

How do we add a line break to a literal? There is also a special combination for this:



Line break character

If you need to add a line break to a string literal, you just need to add a couple of characters: `\n`.

Example:

Code	Console output
`System.out.println("Best regards, \n Anonymous");`	`Best regards,` `Anonymous`

There are a total of 8 special combinations like this, which are also called *escape sequences*. Here they are:

Code	Description
`\t`	Insert a tab character
`\b`	Insert a backspace character
`\n`	Insert a newline character
`\r`	Insert a carriage return character
`\f`	Insert a page feed character
`\'`	Insert a single quotation mark
`\"`	Insert a double quotation mark
`\\`	Insert a backslash

You're familiar with two of them, but what do the other 6 mean?

`\t` is a tab character

When this text appears in text, it is equivalent to pressing the `Tab` key while typing. It shifts the text that follows it and makes it possible to align text.

Example:

Code	Console output

`System.out.println("0\t1\t2\t3");`
`System.out.println("0\t10\t20\t30");`
`System.out.println("0\t100\t200\t300");`
	`0 1 2 3`
	`0 10 20 30`
	`0 100 200 300`

`\b` means 'go back one character'

This sequence in a string is equivalent to pressing the `Backspace` key on the keyboard. It removes the character that precedes it:

Code	Console output
`System.out.println("Hello\b\b World");`	`Hell World!`

`\r` is the carriage return character

This character moves the cursor to the beginning of the current line without changing the text. Whatever next is displayed next will overwrite the existing string.

Example:

Code	Console output
`System.out.println("Greetings\r World!");`	`World!`

`\f` is a page feed character

This symbol comes down to us from the days of the first dot matrix printers. Outputting this sequence to a printer would cause the printer to simply feeds out the current sheet, without printing any text, until a new page begins.

Now we would call it a *page break* or *new page*.

`\\` is a backslash

Everything is straightforward here. If we use a backslash to escape characters in our text, then how do we write a backslash character itself in the string?

It's simple: add a *backslash* to the text — you have to write two in a row.

Example:

Code	Console output
`System.out.println("c:\projects\my\first");`	The compiler will yell at you for unknown escaped characters.
`System.out.println("c:\\projects\\my\\first");`	That's how it's done right!

4. Unicode encoding

As you already know, each character displayed on the screen corresponds to a specific numerical code. A standardized set of these codes is called an *encoding*.

Once upon a time, when computers were newly invented, seven bits (less than one byte) were enough to encode every character. The first encoding contained only 128 characters. This encoding was called *ASCII*.

ASCII stands for American Standard Code for Information Interchange — a standard American code table for printable characters and some special codes.

It consists of 33 non-printable control characters (which affect how text and spaces are processed) and 95 printable characters, including numbers, uppercase and lowercase Latin letters, and several punctuation marks.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LV	VT	FF	CR	SO	SI
1	DEL	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

As computers grew in popularity, each country began to release its own encoding. Usually, they took ASCII as a starting point and replaced rarely used ASCII characters with symbols from their respective alphabets.

Over time, an idea emerged: create a single encoding that contains all the characters of every encoding in the world.

Thus, in 1993, the *Unicode* encoding was created, and the Java language became the first programming language that used this encoding as the standard for storing text. Now Unicode is the standard for the entire IT industry.

Although Unicode itself is the standard, it has several representations or Unicode transformation formats (UTF): UTF-8, UTF-16 and UTF-32, etc.

Java uses an advanced version of Unicode encoding — UTF-16: each character is encoded in 16 bits (2 bytes). It can accommodate up to 65,536 characters!

You can find almost every character of every alphabet in the world in this encoding. Naturally, nobody has the whole thing memorized. You can't know everything, but you can google everything.

To write a Unicode character in your program using its code, you need to write `\u` + *the code in hexadecimal*. For example, `\u00A9`

Code	Console output
`System.out.println("\u00A9 CodeGym");`	© CodeGym

5. Unicode: code point

"640 kilobytes ought to be enough for everyone! Or not". (Quote attributed to Bill Gates)

Life is rough, and over time, the UTF-16 encoding began to be inadequate. It turns out that there are a lot of Asian languages, and they have a lot of glyphs. And all these glyphs simply cannot be crammed into 2 bytes.

What can be done? Use **more bytes**!

But the char type is only 2 bytes and changing it to 4 is not so easy: billions of lines of Java code have been written all over the world, which would break if the char type suddenly becomes 4 bytes a Java machine. So we can't change the char type!

There is another approach. Remember how we escape characters by putting a backslash in front of them. Basically, we encoded a single character using multiple characters.

Java's creators decided to use the same approach.

Some characters that visually appear as a single character are encoded as two `char`s in a string:

Code	Console output
`System.out.println("\uD83D\uDD0A");`	🔊

Now your Java program can even output emojis to the console 😎.