Encodings

1. Octal encoding

Speaking of encodings... As you know, in everyday life we use **decimal notation**: all our numbers are represented using 10 symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. There are 10 numerals, so the system is called decimal.

But programmers are big-time inventors. They immediately came up with encodings that use a different number of symbols. For example, 16, 8 and 2.

The alternative encoding that uses 8 symbols is the easiest: just drop the 8 and 9 and you get an octal encoding (the **octal numeral system**).

And, yes, you can use the octal system to specify numeric literals. If, of course, you really need to. It's easier than it sounds. You just need to write the number 0 before the number.

In other words, Java treats any integer literal starting with 0 as an octal value.

Examples:

Code	Notes
`int x = 015;`	x is 13: 1*8+5
`int x = 025;`	x is 21: 2*8+5
`int x = 0123;`	x is 83: 1*64+2*8+3 == $1*8^2+2*8^1+3*8^0$
`int x = 078;`	This will not compile: 8 is not one of the symbols used in the octal encoding.

It's unlikely that you will need to write octal numbers in your code, but you should know what they are. After all, you will have to read code written by others. And as mentioned above, programmers are big inventors.

Well, remember that you can't simply write 0 in front of every number.

2. Binary encoding

Binary encoding is even more interesting. If octal has only the digits 0-7, then binary has only 0 and 1. Why is this encoding necessary?

This has everything to do with the internal structure of a computer. Everything in a computer runs on electricity, and as it happens, the most efficient way to store and transmit something using electricity is to use two states: either there is no electricity in the wire (zero) and there is electricity (one).

This is the origin of the popularity of the binary numeral system.

In principle, it is not used very often in Java: Java is considered a high-level language, completely abstracted from the hardware it runs on. Indeed, do you really care what format is used to store and process data inside a computer?

But over the past decades, programmers have come to love the binary encoding (and other encodings based on it). As a result, Java has operators that take binary numbers as inputs. And the accuracy of floating-point numbers depends on their binary representation.

Basically, it is better for you to know about this encoding than to not know.

And as was the case with octal encoding, Java has a way to encode literals using the binary system. That is, literals comprised only of 0s and 1s. In order for the *Java compiler* to understand that the code contains a numeric literal encoded in binary rather than simply a decimal number consisting of zeros and ones, all binary literals are identified using the prefix **0b** (the 'b' comes from the word binary).

Examples:

Code	Notes
`int x = 0b100;`	x is 4: 1*4+0*2+0
`int x = 0b1111;`	x is 15: 1*8+1*4+1*2+1
`int x = 0b1111000111;`	x is 967: $1*2^9+1*2^8+1*2^7+1*2^6+0*2^5+0*2^4+0*2^3+1*2^2+1*2+1$;
`int x = 0b12000;`	This will not compile: 2 is not one of the symbols used in the binary encoding.

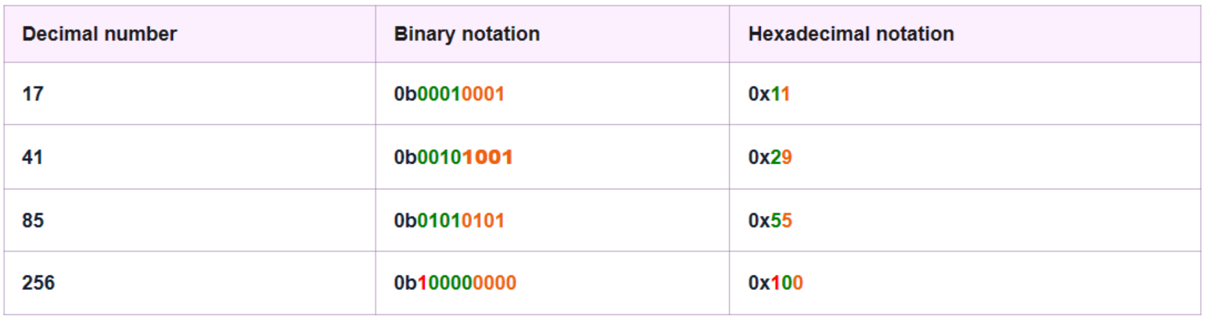
3. Hexadecimal encoding

In addition to octal and binary encodings, literals can also be written in hexadecimal. This is a very popular encoding.

That is because although binary notation is as close as possible to how numbers are actually stored, it is too difficult for humans to effectively work with such numbers: in binary, the number one million contains 20 digits, not not 7.

That's why programmers came up with the hexadecimal system. After all, 16 is 2 raised to the 4th power, so exactly 4 bits correspond to one hexadecimal digit. Roughly speaking, every 4 bits can now be written as a single hexadecimal digit.

The hexadecimal encoding also has its own unique prefix: **0x**. Examples:

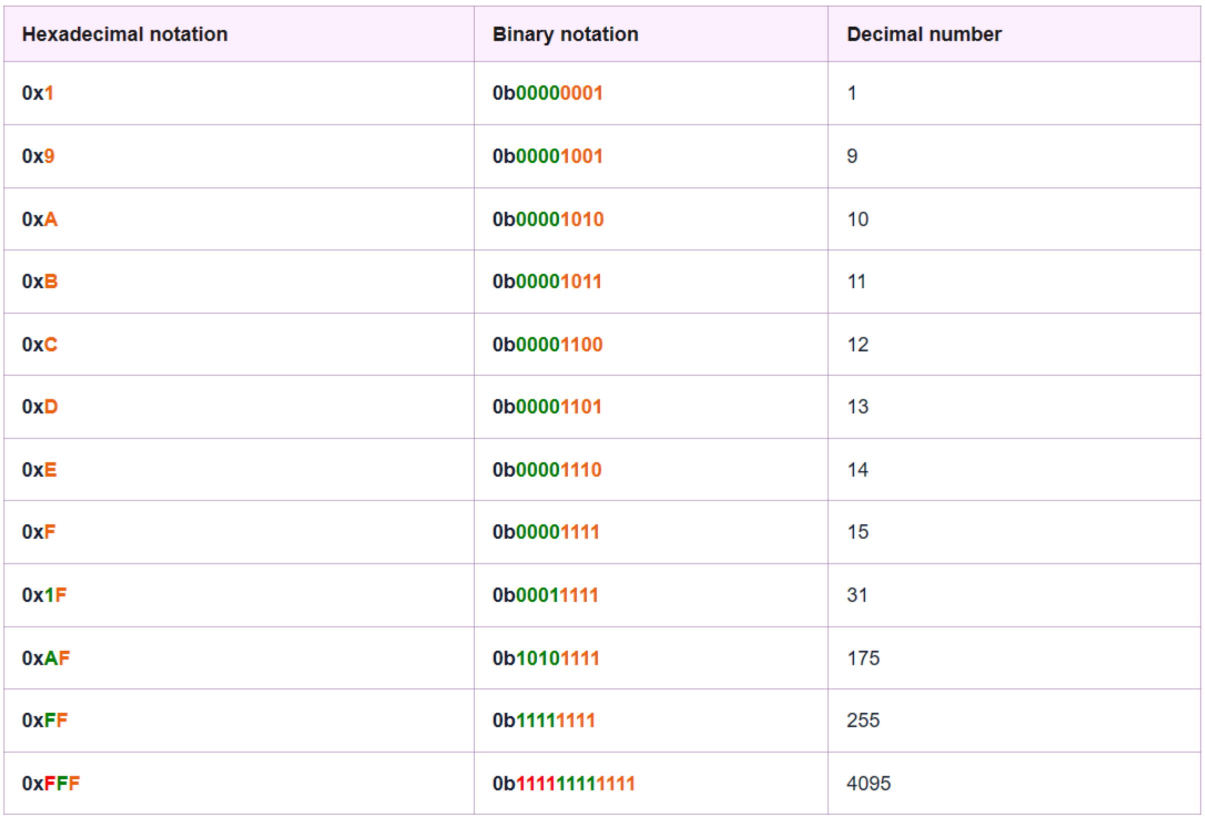


Decimal number	Binary notation	Hexadecimal notation
17	0b00010001	0x11
41	0b00101001	0x29
85	0b01010101	0x55
256	0b100000000	0x100

Ok, you say, it's clear enough how we got the octal system: we just threw out the numbers 8 and 9, but where do we get the 6 extra symbols for the hexadecimal system? I would like to see them!

It's all straightforward. The first 6 letters of the English alphabet were taken as the 6 missing symbols: **A** (10), **B** (11), **C** (12), **D** (13), **E** (14), **F** (15).

Examples:



Hexadecimal notation	Binary notation	Decimal number
0x1	0b00000001	1
0x9	0b00001001	9
0xA	0b00001010	10
0xB	0b00001011	11
0xC	0b00001100	12
0xD	0b00001101	13
0xE	0b00001110	14
0xF	0b00001111	15
0x1F	0b00011111	31
0xAF	0b10101111	175
0xFF	0b11111111	255
0xFFF	0b111111111111	4095

4. How to convert a number from hexadecimal

Converting a number from hexadecimal to decimal is very easy. Let's say you have the number **0xAFCF**. How much is that in decimal?

First, we have a positional number system, which means the contribution of each digit to the overall number increases by a factor of 16 as we move from right to left:

$$A*16^{3} + F*16^{2} + C*16^{1} + F$$

The symbol A corresponds to the number 10, the letter C corresponds to the number 12, and the letter F represents fifteen. We get:

$$10*16^{3} + 15*16^{2} + 12*16^{1} + 15$$

Raising 16 to the various powers that correspond to the digits, we get:

10*4096 + 15*256 + 12*16 + 15

We sum everything up and get:

45007

You know how this number is stored in memory:

0xAFCF

But now let's convert it to binary. In binary it would be:

0b1010111111001111

Every set of four bits corresponds to exactly one hexadecimal character. That's super convenient. Without any multiplication or exponentiation."