3 Java as an example of a simple programming language

- Programming languages
- Lexical structure of Java programs
- Variable – Name – Value – Type
- Literals and Constants
- Truth values
- Assignments and Expressions
- Basic control structures in Java
- Integer Numbers
- Floating Point Numbers
- Characters and Strings
- Additional Operators
- Methods

- A **program** is the implementation of an algorithm with a programming language in order to execute it on a computer.
- Very simple languages are in principle (!) sufficient to implement all computable algorithms.
  Unfortunately, then the programming is very laborious.
- To make the implementation of specific types of algorithms more comfortable, a large number of programming languages have been developed.
- Thus programming languages are usually optimized for a specific scope.

Programming languages often follow one of the common basic patterns of programming (**programming paradigms**):

- imperative (Fortran, Cobol, Algol, Pascal, Modula, C, Java,...)
- object oriented (Smalltalk, C++, Delphi, Java, C#,...)
- functional (Lisp, OCAML, Haskell...)
- logical (Prolog,...)

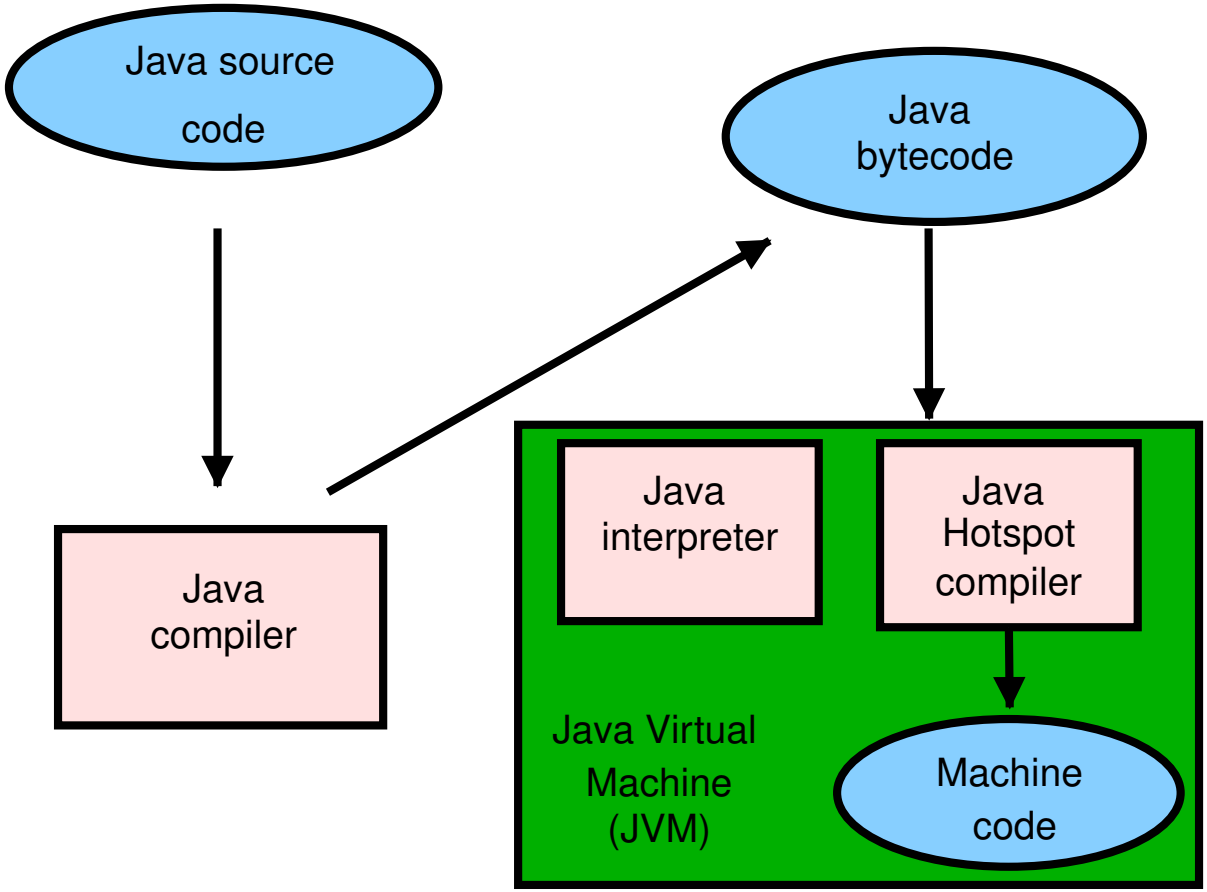
More recent languages are often extended by components of other paradigms (e.g., Java with functional components).

Java as an example of a current programming language:

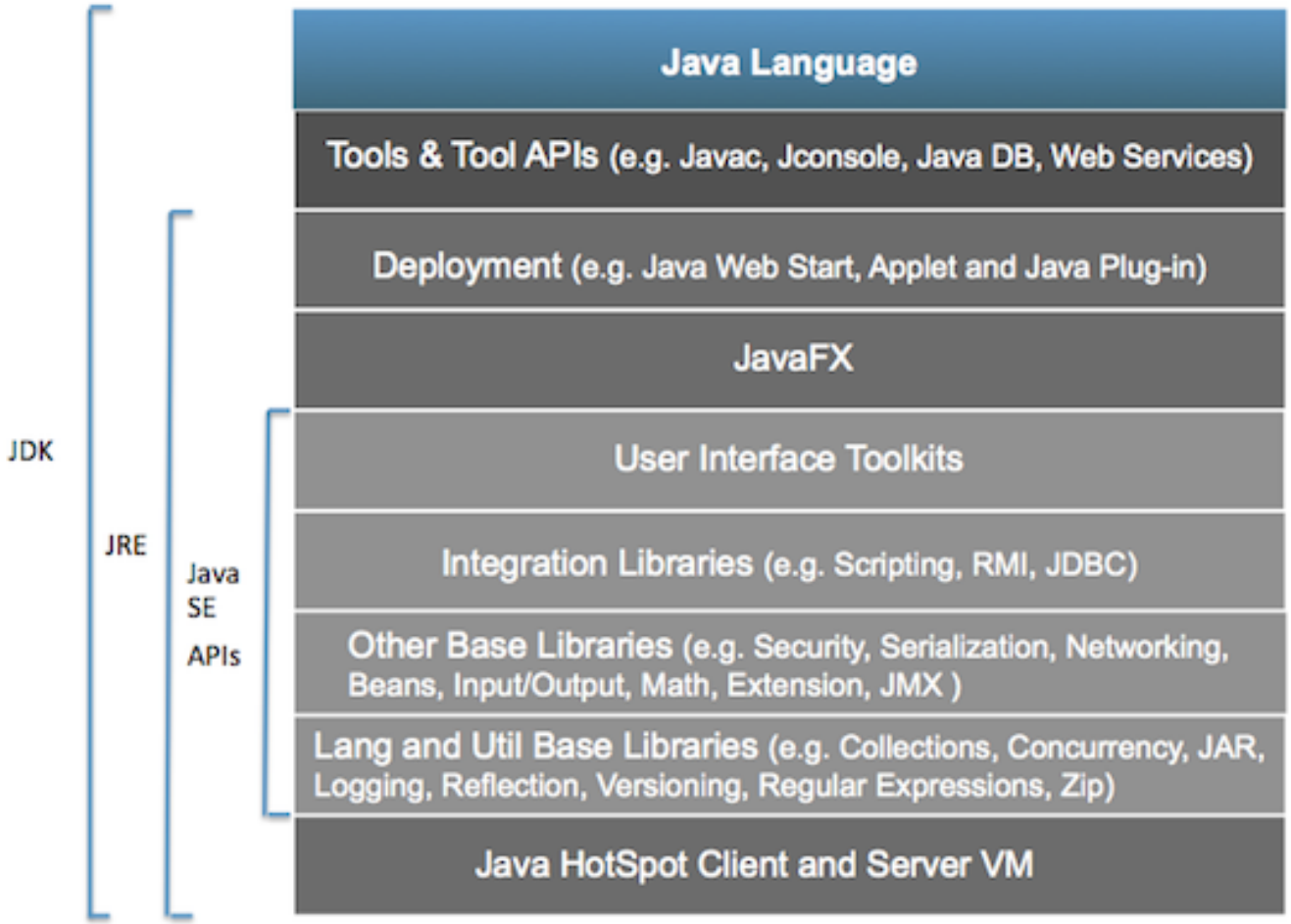
- Java is: imperative, object oriented, platform independent, supported by libraries, Internet enabled
- The core of Java is relatively simple.
- In order to provide independence of the actual processor and operating system, a large number of ready program components is provided.
-

Java version	year	packages	classes/interfaces	members
1.0	1996	8	212	2125
1.1	1997	23	504	5478
1.2	1998	59	1520	ca. 16000
1.3	2000	76	1842	ca. 20000
1.4	2002	135	2991	ca. 32000
5.0	2004	165	>3000	
SE 6	2006	?	?	?
SE 7	2011	> 200	> 4000	?
SE 8 (LTS)	2014	> 200	> 4000	?
SE 9	2017		> 6000	
SE 10	3/2018			
SE 11	9/2018			
SE 12	3/2019			

Java uses source code and byte code that is machine-independent:



Conceptual architecture of Java SE 7:



3 Java as an example of a simple programming language
- Programming languages
- Lexical structure of Java programs
- Variable – Name – Value – Type
- Literals and Constants
- Truth values
- Assignments and Expressions
- Basic control structures in Java
- Integer Numbers
- Floating Point Numbers
- Characters and Strings
- Additional Operators
- Methods

Elements of a simple Java program

```
1  import javax.swing.JOptionPane;
2  /* Greatest common divisor  */
3  class K3B01E_GCD {
4    public static void main(String[] args) {
5
6    int a,b;
7
8    a  = Integer.parseInt(JOptionPane.showInputDialog ("A = "));
9    b  = Integer.parseInt(JOptionPane.showInputDialog ("B = "));
10
11   while (a !=  b){
12     if (a > b) a = a - b;
13     else  b  = b - a;
14   }
15
16   JOptionPane.showMessageDialog (null, "The GCD is: " + a);
17   }
18 }
```

- line 1: **package import**
- line 2: **comment**
  pause
- line 3: **class header**
- line 4: **method header**
- line 6: **declarations**
- lines 8-9: **input**
- lines 11-14: **implementation of algorithm**
- line 16: **output**

Lexical components of a Java program

The source code of a Java program includes:

- names (of variables, methods, classes, ...)
- literals (constants)
- operators
- separation characters
- reserved terms
- comments
- white space

Reserved terms in Java

The following lists contains the Java keywords with a special meaning:

`abstract`	`assert`	`boolean`	`break`	`byte`
`case`	`catch`	`char`	`class`	`continue`
`default`	`do`	`double`	`else`	`enum`
`extends`	`final`	`finally`	`float`	`for`
`if`	`implements`	`import`	`instanceof`	`int`
`interface`	`long`	`native`	`new`	`package`
`private`	`protected`	`public`	`return`	`short`
`static`	`strictfp`	`super`	`switch`	`synchronized`
`this`	`throw`	`throws`	`transient`	`try`
`void`	`volatile`	`while`		

Reserved but not in use:

`const`	`goto`

Reserved terms are ‘terminals’ of the language definition and must not be used as names for variables etc.!

Comments

- ‘Comments’ are used to facilitate reading the source code of a program.
- Comments to not influence program execution!
- Without comments, programs often are so hard to understand that even the author does not understand them after a short while
- It cannot be expected from a corrector to grade programs without comments! From now on: programs without or with not enough comments will receive fewer points!
- Possible forms of comments in Java:

```
1  /*      a comment  */
2  //      a comment until the end of the line
3  /**     documentation that can be processed by javadoc  */
```

White space

- White space makes a program more readable.
- Java provides the following options for inserting white space:
  - **`SP`** (*space* with the space key)
  - **`HT`** (*horizontal tab* with the tab key)
  - **`FF`** (*form feed*)
  - **`LF`** (*line feed* with the enter key)
  - **`CR`** (*carriage return*)
- While space between symbols is ignored.
- White space within symbols is a (syntax) error
- For a guideline how comments and white space should be used see *'Code Conventions for the Java Programming Language'* (**`http://www.oracle.com/technetwork/java/codeconventions-150003.pdf`**)

Important for comments and white space:

- Software is maintained during 80% of its life time.
- This maintenance is hardly ever done only by the original author!
- Following the conventions improves readability of software, such that it can be understood more quickly and more easily.

Separation characters

Separation characters delimit instructions, conditions, blocks, etc.

- '**(**' and '**)**' are the 'standard parenthesis', especially in 'expressions'.
- '**[**' and '**]**' are used with arrays.
- '**{**' and '**}**' mark blocks in programs,
  which can be used to merge multiple instructions to one (composite) instruction.
- '**;**' finishes instructions
- '**,**' is used for lists of things

Operators

Operators are used to write down arithmetic operations, arithmetic comparisons, logical connectives, and bit operations:

`=`	`<`	`>`	`!`	`~`	`?`	`:`				
`==`	`<=`	`>=`	`!=`	`&&`	`||`	`++`	`--`			
`+`	`-`	`*`	`/`	`&`	`|`	`^`	`%`	`<<`	`>>`	`>>>`
`+=`	`-=`	`*=`	`/=`	`&=`	`|=`	`^=`	`%=`	`<<=`	`>>=`	`>>>=`

3 Java as an example of a simple programming language

- Programming languages
- Lexical structure of Java programs
- Variable – Name – Value – Type
- Literals and Constants
- Truth values
- Assignments and Expressions
- Basic control structures in Java
- Integer Numbers
- Floating Point Numbers
- Characters and Strings
- Additional Operators
- Methods

To store data in programs **variables** are used:

- Variables had originally been abstractions of memory cells.
- They have a **name**, a **type**, and a **value**.
- They have to be declared before they can be used.
- They change their value due to assignments, increments, or decrements.
- They always contain a value which is compatible to their type.
- *Local variables* are valid from their declaration to the end of the corresponding block. (see later)
- *Local variables* must be explicitly assigned a value before they can be read. (see later)

A **variable name** consists of a sequence of uppercase and lowercase letters, digits, and some special characters.

- Be careful: Java distinguishes uppercase and lowercase letters!
- '`myVar`', '`myvar`', '`myVAR`' are different names!
- Names should follow the rules in the *Code Conventions for the Java Programming Language* .
- Thus all variable names should begin with a lowercase letter ...
- ... and in compound names the inner names should begin with a capital letter ('camelCase')
- good names:

  `myVariable`, **`i`**, **`length`**, **`averageCircleSize`**, **`newPrize1`**, **...**
- bad names:

  `Myvariable`, **`LENGTH`**, **`_i`**, **`$var`**, **`averagecirclesize`**,**...**

Every variable has a '**type**' that determines which type of data it can store.

Examples for types in Java are:

- **`byte`** (for integers from $-$**128** to **127**)
- **`short`** (for integers from $-$**32768** to **32767**)
- **`int`** (for integers from $-$**2147483648** to **2147483647**)
- **`long`** (for larger integers until about $\pm \mathbf{9 \cdot 10^{18}}$)
- **`float`** (for floating point numbers, pretty inexact)
- **`double`** (for floating point numbers, less inexact)
- **`char`** (for single letters)
- **`String`** (for texts)

(more details later...)

```
1 int a, b;           // a and b are declared with type 'int'
2 int c = 1, d = 3; // declaration with initialization
3 b = 1;              // assignment of value 1 to b
4 a = b + 2;          // assignment of value 3 to a
```

Examples for usage of variables

```
1  int width, length;
2  float myReal, yourFloat;
3
4  int i =1, j, k = 0;          // declaration with initialization
5
6  short _short, sHort, $short; // possible, but bad style
7
8  { int x = 1, y;         // declaration at start of block, if possible
9     y = x; }
10
11 { int x = 0;
12    x = x + 1; }         //  an assignment is not an equation !!
13
14 int x = 3, y = 5;
15 { int help;                          //  swap values
16    help = x; x = y; y = help; }  //  of x and y
```

3 Java as an example of a simple programming language
- Programming languages
- Lexical structure of Java programs
- Variable – Name – Value – Type
- Literals and Constants
- Truth values
- Assignments and Expressions
- Basic control structures in Java
- Integer Numbers
- Floating Point Numbers
- Characters and Strings
- Additional Operators
- Methods

Literals are used to represent numbers in the **source code of the program**
(in contrast to their internal representation in the computer)

- As **integers** are interpreted:
  - sequences of digits
  - sequences of digits that end with `l` or `L`
  - leading `0x` followed by hexadecimal digits, maybe with trailing `l`/`L`
- Literals that are interpreted as integers are mapped to the type `int` or the type `long` (requires suffix 'l' or 'L'):

`793677`, `1`, `35`	`int`
`0L`, `17L`, `3094l`	`long`
`0xDadaCafe`	`int`, hexadecimal (prefix '0x')
`0xC0B0L`	`long`, hexadecimal (prefix '0x', suffix 'L')
`01234567`	`int`, attention: octal (prefix '0')
`- 23`	not a literal (but: operator and literal)
`9876543210`	compiler error, value too large for `int`

- Sequences of digits that include at least one of the following characters are interpreted as **floating point numbers**
  - '.' as decimal point
  - **`E`** or **`e`** for the exponent (decimal, scientific notation)
  - suffix 'F' or 'f' (to explicitly select the type **`float`**)
  - suffix 'D' or 'd' (to explicitly select the type **`double`**)
- Examples are:

`75.286`	**`double`**
`.17E-9`, `3.e+15`, `5E19`	**`double`**
`14d`, `4711D`	**`double`**
`.17E-9F`, `3.e+15f`	**`float`**
`9876543210f`	**`float`**
`1e50f`	compiler error, value too large for **`float`**

- Tip: use only the type **`double`**...!

- A variable that is declared with the prefix '**`final`**' is an (immutable) '**constant**':

```
1 final float PI = 3.14f;
2 final int ONE = 1, NULL = 0;
3 final int MY_INTEGER, YOUR_INTEGER;
```

- Predefined are for example **`Double.NaN`**, **`Double.MAX_VALUE`**, **`Integer.MIN_VALUE`** or **`Long.MAX_VALUE`**
- Names should follow the rules in the '*Code Conventions for the Java Programming Language*' ,
  for example
  - the name of constants should consist only of capitals.
  - the names of compound names should be connected with '*underscore*' ('_').

3 Java as an example of a simple programming language
- Programming languages
- Lexical structure of Java programs
- Variable – Name – Value – Type
- Literals and Constants
- Truth values
- Assignments and Expressions
- Basic control structures in Java
- Integer Numbers
- Floating Point Numbers
- Characters and Strings
- Additional Operators
- Methods

boolean is the data type for truth values (after George Boole).

Possible values are **`true`**, **`false`**

Examples:

```
1 boolean bool1, bool2, bool3;
2 bool1 = true;
3 int i = 1;
4 bool2 = i == 0;     //  yields 'false'
5 if (true) bool3 = false;
6 if (bool2) System.out.print("i␣is 0");
```

- Operators to generate truth values:
  '**==**', '**!=**', '**<**', '**<=**', '**>**' '**>=**'
- Operators to combine truth values:
  '**!**', '**&&**', '**||**'
- more details later...

3 Java as an example of a simple programming language
- Programming languages
- Lexical structure of Java programs
- Variable – Name – Value – Type
- Literals and Constants
- Truth values
- Assignments and Expressions
- Basic control structures in Java
- Integer Numbers
- Floating Point Numbers
- Characters and Strings
- Additional Operators
- Methods

- *Syntax* of **assignments**:

  `x = expression`

  If **`x`** has one of the types **`byte`**, **`short`**, **`int`**, **`long`**, **`float`**, **`double`**,
  then **`expression`** resembles arithmetic expressions from elementary mathematics
  and consists of variables, constants, operators, and parentheses.
- *Semantics* of assignments:
  Compute the value of **`expression`** and assign it to the variable **`x`**.
  - Variables appearing in **`expression`** must have a value.
  - The type of **`expression`** must be compatible to the type of **`x`**.
- In addition the value of an assignment is the assigned value, i.e., '**`=`**' is also an operator!
  Example: '**`x=y=2`**'

Arithmetic Operators

- Operators for addition '`+`' and subtraction '`-`'
  - Example: `x = -(y + z);`
  - Example: `x = +y + (z - w);`
- Operator for multiplication '`*`'
  - Example: `x = y * (z + 7)`
- Operator for division '`/`'
  - Example: `x = z / y`
  - Important: With integer values the result is rounded towards zero, i.e., `99/100` yields `0`, similar to `(-99)/100`

- Operator for remainder with Euclidean division '`%`'
  - Example: `x = z % y`
  - If `z` and `y` are both non-negative, then this is the modulo function, `9%5` yields `4`

  Be careful with negative arguments:
  - in Java/C/JavaScript/PHP `z%y` is equivalent to `z-(z/y)*y`, which is *not* the modulo function for negative arguments.
    `(-9)%5` thus yields `-4`
  - In Perl/Python/Ruby `z%y` is equivalent to $z - \lfloor \frac{z}{y} \rfloor * y$.
    `(-9)%5` yields `1` in this case
- Operators for increment '`++`' and decrement '`--`'
  - Example: `x++` corresponds to `x = x + 1`
  - Example: `x--` corresponds to `x = x - 1`
- ... and more (see later)

Operator precedence:

In expressions that are not fully parenthesized the order of execution is determined by operator precedence (or order of operations):

Example: `a-b*c/d+e` is interpreted as `((a-((b*c)/d))+e)`

The predefined precedence rules are:

precedence-level	operator	operation	processing-direction
1	'`++`'	increment (postfix)	L->R
	'`--`	decrement (postfix)	
2	'`+`'	plus (unary, prefix)	R->L
	'`-`'	minus (unary, prefix)	
3	'`*`'	multiplication	L->R
	'`/`'	division	
	'`%`'	remainder	
4	'`+`'	addition	L->R
	'`-`'	subtraction	
5	'`=`'	assignment	R->L

Type conversion with elementary types:

- It is possible to mix different types in arithmetic expressions,
- this requires conversion between the types.

In Java conversions are used in three ways:

- with an assignment
- during the evaluation of arithmetic expressions
- with explicit casting

- ‘**widening**’ = conversion into a type with larger value space, therefore (usually) unproblematic
  ‘Widening’ done automatically when required:

byte	in	**short**,	**int**,	**long**,	**float**,	**double**
short	in		**int**,	**long**,	**float**,	**double**
char	in		**int**,	**long**,	**float**,	**double**
int	in			**long**,	**float**(1),	**double**
long	in				**float**(1),	**double**(1)
float	in					**double**

  With (1) some precision may be lost, see below
- ‘**narrowing**’ = conversion into type with smaller/inappropriate value space.
  - here we may lose information,
  - since the target type may not be able to represent the original value.
  - this requries an explicit type conversion (‘cast’).
  - **double** → **float** → **long** → **int** →**short**/**char** → **byte**

Example: consider variables `float floatNumber; int intNumber;`

- conversion during assignment (only ‘widening’ allowed):
  - `floatNumber = intNumber;`
- Conversion during the evaluation of arithmetic expressions:
  - `floatNumber = floatNumber / intNumber;`

  division with `float`-copy of `intNumber` (widening!)
- conversion with explicit casting, in particular with narrowing:
  - `intNumber = (int) floatNumber;`

Examples for casting:

```
 (float) 1.79E+308  →  infinity
  (long) 1.79E+308  →  9223372036854775807
   (int) 1.79E+308  →  2147483647
(short) 2147483647  →  -1
       (int) 1.0e6  →  1000000
        (int) 1.11  →  1
       (int) -1.49  →  -1
        (byte) 259  →  3
```

Conditions are required to generate Boolean values
(for example for branches in the program execution)

- The evaluation of a condition yields either **`true`** or **`false`**.
- Possible forms of conditions:
  - **`true`**
  - **`false`**
  - simple condition:

    `expression comparison_operator expression`

  - compound condition:

    `unary_logical_operator condition`

    or

    `condition binary_logical_operator condition`

Comparison operators yield values of the type **`boolean`**

Java operator	mathematical symbol	pronounced
`==`	$=$	‘equal to’
`!=`	$\neq$	‘not equal to’
`<`	$<$	‘less than’
`<=`	$\leq$	‘less than or equal to’
`>`	$>$	‘greater than’
`>=`	$\geq$	‘greater than or equal to’

Logical operators combine values of the type **`boolean`**

Java operator	mathem. symbol	pronounced	precedence	order
unary, i.e., one operand				
`!`	$\neg$	'not'	1	$R \rightarrow L$
binary, i.e., two operands				
`&&`	$\wedge$	'and'	2	$L \rightarrow R$
`||`	$\vee$	'or'	3	$L \rightarrow R$

Truth tables:

`A`	**`!A`**
`true`	**`false`**
`false`	**`true`**

`A`	**`B`**	**`A&&B`**	**`A||B`**
`true`	**`true`**	**`true`**	**`true`**
`true`	**`false`**	**`false`**	**`true`**
`false`	**`true`**	**`false`**	**`true`**
`false`	**`false`**	**`false`**	**`false`**