Java Tutorials

Why Use Java?

* Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
* It is one of the most popular programming languages in the world
* It has a large demand in the current job market
* It is easy to learn and simple to use
* It is open-source and free
* It is secure, fast and powerful
* It has huge community support (tens of millions of developers)
* Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
* As Java is close to [C++](https://www.w3schools.com/cpp/default.asp) and [C#](https://www.w3schools.com/cs/default.asp), it makes it easy for programmers to switch to Java or vice versa.

To check if you have Java installed on a Windows PC, search in the start bar for Java or type the following in Command Prompt (cmd.exe):

C:\Users\Your Name>java -version

## Setup for Windows

To install Java on Windows:

1. Go to "System Properties" (Can be found on Control Panel > System and Security > System > Advanced System Settings)
2. Click on the "Environment variables" button under the "Advanced" tab
3. Then, select the "Path" variable in System variables and click on the "Edit" button
4. Click on the "New" button and add the path where Java is installed, followed by **\bin**. By default, Java is installed in C:\Program Files\Java\jdk-11.0.1 (If nothing else was specified when you installed it). In that case, You will have to add a new path with: **C:\Program Files\Java\jdk-11.0.1\bin**  
   Then, click "OK", and save the settings
5. At last, open Command Prompt (cmd.exe) and type **java -version** to see if Java is running on your machine

Save the code in Notepad as "Main.java". Open Command Prompt (cmd.exe), navigate to the directory where you saved your file, and type "javac Main.java":

C:\Users\*Your Name*>javac Main.java

This will compile your code. If there are no errors in the code, the command prompt will take you to the next line. Now, type "java Main" to run the file:

C:\Users\*Your Name*>java Main

The output should read:

Hello World

## System.out.println()

Inside the main() method, we can use the println() method to print a line of text to the screen:

**Note:** The curly braces {} marks the beginning and the end of a block of code.

System is a built-in Java class that contains useful members, such as out, which is short for "output". The println() method, short for "print line", is used to print a value to the screen (or a file).

Don't worry too much about System, out and println(). Just know that you need them together to print stuff to the screen.

## Java Variables

Variables are containers for storing data values.

In Java, there are different **types** of variables, for example:

* String - stores text, such as "Hello". String values are surrounded by double quotes
* int - stores integers (whole numbers), without decimals, such as 123 or -123
* float - stores floating point numbers, with decimals, such as 19.99 or -19.99
* char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
* boolean - stores values with two states: true or false

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

int myNum = 15;

myNum = 20; // myNum is now 20

System.out.println(myNum);

## Final Variables

If you don't want others (or yourself) to overwrite existing values, use the final keyword (this will declare the variable as "final" or "constant", which means unchangeable and read-only):

### **Example**

final int myNum = 15;

myNum = 20; // will generate an error: cannot assign a value to a final variable

## Other Types

A demonstration of how to declare variables of other types:

### **Example**

int myNum = 5;

float myFloatNum = 5.99f;

char myLetter = 'D';

boolean myBool = true;

String myText = "Hello";

You can also use the + character to add a variable to another variable:

String firstName = "John ";

String lastName = "Doe";

String fullName = firstName + lastName;

## System.out.println(fullName);

## Declare Many Variables

To declare more than one variable of the **same type**, you can use a comma-separated list:

### **Example**[**Get your own Java Server**](https://www.w3schools.com/java/java_server.asp)

Instead of writing:

int x = 5;

int y = 6;

int z = 50;

System.out.println(x + y + z);

You can simply write:

int x = 5, y = 6, z = 50;

System.out.println(x + y + z);

## One Value to Multiple Variables

You can also assign the **same value** to multiple variables in one line:

### **Example**

int x, y, z;

x = y = z = 50;

System.out.println(x + y + z);

The general rules for naming variables are:

* Names can contain letters, digits, underscores, and dollar signs
* Names must begin with a letter
* Names should start with a lowercase letter, and cannot contain whitespace
* Names can also begin with $ and \_ (but we will not use it in this tutorial)
* Names are case-sensitive ("myVar" and "myvar" are different variables)
* Reserved words (like Java keywords, such as int or boolean) cannot be used as names

## Java Data Types

Data types are divided into two groups:

* Primitive data types - includes byte, short, int, long, float, double, boolean and char
* Non-primitive data types - such as [String](https://www.w3schools.com/java/java_strings.asp), [Arrays](https://www.w3schools.com/java/java_arrays.asp) and [Classes](https://www.w3schools.com/java/java_classes.asp) (you will learn more about these in a later chapter)

## Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Size** | **Description** |
| byte | 1 byte | Stores whole numbers from -128 to 127 |
| short | 2 bytes | Stores whole numbers from -32,768 to 32,767 |
| int | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647 |
| long | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| float | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits |
| double | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits |
| boolean | 1 bit | Stores true or false values |
| char | 2 bytes | Stores a single character/letter or ASCII values |

## Numbers

Primitive number types are divided into two groups:

**Integer types** stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are byte, short, int and long. Which type you should use, depends on the numeric value.

**Floating point types** represents numbers with a fractional part, containing one or more decimals. There are two types: float and double.

Even though there are many numeric types in Java, the most used for numbers are int (for whole numbers) and double (for floating point numbers). However, we will describe them all as you continue to read.

## Integer Types

### **Byte**

The byte data type can store whole numbers from -128 to 127. This can be used instead of int or other integer types to save memory when you are certain that the value will be within -128 and 127:

### **Example**[**Get your own Java Server**](https://www.w3schools.com/java/java_server.asp)

byte myNum = 100;

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_byte)

### **Short**

The short data type can store whole numbers from -32768 to 32767:

### **Example**

short myNum = 5000;

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_short)

### **Int**

The int data type can store whole numbers from -2147483648 to 2147483647. In general, and in our tutorial, the int data type is the preferred data type when we create variables with a numeric value.

### **Example**

int myNum = 100000;

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_int)

### **Long**

The long data type can store whole numbers from -9223372036854775808 to 9223372036854775807. This is used when int is not large enough to store the value. Note that you should end the value with an "L":

### **Example**

long myNum = 15000000000L;

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_long)

## Floating Point Types

You should use a floating point type whenever you need a number with a decimal, such as 9.99 or 3.14515.

The float and double data types can store fractional numbers. Note that you should end the value with an "f" for floats and "d" for doubles:

### **Float Example**

float myNum = 5.75f;

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_float)

### **Double Example**

double myNum = 19.99d;

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_double)

Use float or double?

The **precision** of a floating point value indicates how many digits the value can have after the decimal point. The precision of float is only six or seven decimal digits, while double variables have a precision of about 15 digits. Therefore it is safer to use double for most calculations.

### **Scientific Numbers**

A floating point number can also be a scientific number with an "e" to indicate the power of 10:

### **Example**

float f1 = 35e3f;

double d1 = 12E4d;

System.out.println(f1);

System.out.println(d1);

Char

Alternatively, if you are familiar with ASCII values, you can use those to display certain characters:

### **Example**

char myVar1 = 65, myVar2 = 66, myVar3 = 67;

System.out.println(myVar1);

System.out.println(myVar2);

System.out.println(myVar3);

## Strings

The String data type is used to store a sequence of characters (text). String values must be surrounded by double quotes:

### **Example**

String greeting = "Hello World";

System.out.println(greeting);

The String type is so much used and integrated in Java, that some call it "the special **ninth** type".

A String in Java is actually a **non-primitive** data type, because it refers to an object. The String object has methods that are used to perform certain operations on strings. **Don't worry if you don't understand the term "object" just yet**. We will learn more about strings and objects in a later chapter.

## Non-Primitive Data Types

Non-primitive data types are called **reference types** because they refer to objects.

The main difference between **primitive** and **non-primitive** data types are:

* Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
* Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
* A primitive type has always a value, while non-primitive types can be null.
* A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.

# **Java Type Casting**

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

* **Widening Casting** (automatically) - converting a smaller type to a larger type size  
  byte -> short -> char -> int -> long -> float -> double
* **Narrowing Casting** (manually) - converting a larger type to a smaller size type  
  double -> float -> long -> int -> char -> short -> byte

## Widening Casting

Widening casting is done automatically when passing a smaller size type to a larger size type:

### **Example**

public class Main {

public static void main(String[] args) {

int myInt = 9;

double myDouble = myInt; // Automatic casting: int to double

System.out.println(myInt); // Outputs 9

System.out.println(myDouble); // Outputs 9.0

}

}

## Narrowing Casting

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

### **Example**

public class Main {

public static void main(String[] args) {

double myDouble = 9.78d;

int myInt = (int) myDouble; // Manual casting: double to int

System.out.println(myDouble); // Outputs 9.78

System.out.println(myInt); // Outputs 9

}

}

# **Java Operators**

Operators are used to perform operations on variables and values.

Java divides the operators into the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Bitwise operators

## Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Example** | **Try it** |
| + | Addition | Adds together two values | x + y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_add) |
| - | Subtraction | Subtracts one value from another | x - y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_sub) |
| \* | Multiplication | Multiplies two values | x \* y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_mult) |
| / | Division | Divides one value by another | x / y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_div) |
| % | Modulus | Returns the division remainder | x % y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_mod) |
| ++ | Increment | Increases the value of a variable by 1 | ++x | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_inc) |
| -- | Decrement | Decreases the value of a variable by 1 | --x | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_dec) |

## Java Assignment Operators

Assignment operators are used to assign values to variables.

A list of all assignment operators:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Example** | **Same As** | **Try it** |
| = | x = 5 | x = 5 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass1) |
| += | x += 3 | x = x + 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass2) |
| -= | x -= 3 | x = x - 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass3) |
| \*= | x \*= 3 | x = x \* 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass4) |
| /= | x /= 3 | x = x / 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass5) |
| %= | x %= 3 | x = x % 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass6) |
| &= | x &= 3 | x = x & 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass7) |
| |= | x |= 3 | x = x | 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass8) |
| ^= | x ^= 3 | x = x ^ 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass9) |
| >>= | x >>= 3 | x = x >> 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass10) |
| <<= | x <<= 3 | x = x << 3 |  |

## Java Comparison Operators

Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.

The return value of a comparison is either true or false. These values are known as Boolean values, and you will learn more about them in the [Booleans](https://www.w3schools.com/java/java_booleans.asp) and [If..Else](https://www.w3schools.com/java/java_conditions.asp) chapter.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| == | Equal to | x == y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_compare1) |
| != | Not equal | x != y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_compare2) |
| > | Greater than | x > y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_compare3) |
| < | Less than | x < y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_compare4) |
| >= | Greater than or equal to | x >= y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_compare5) |
| <= | Less than or equal to | x <= y |  |

## Java Logical Operators

You can also test for true or false values with logical operators.

Logical operators are used to determine the logic between variables or values:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Example** | **Try it** |
| && | Logical and | Returns true if both statements are true | x < 5 &&  x < 10 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_logical1) |
| || | Logical or | Returns true if one of the statements is true | x < 5 || x < 4 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_logical2) |
| ! | Logical not | Reverse the result, returns false if the result is true | !(x < 5 && x < 10) |  |

## Java Strings

Strings are used for storing text.

A String variable contains a collection of characters surrounded by double quotes:

## String Length

A String in Java is actually an object, which contain methods that can perform certain operations on strings. For example, the length of a string can be found with the length() method:

String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

System.out.println("The length of the txt string is: " + txt.length());

## More String Methods

There are many string methods available, for example toUpperCase() and toLowerCase():

### **Example**

String txt = "Hello World";

System.out.println(txt.toUpperCase()); // Outputs "HELLO WORLD"

System.out.println(txt.toLowerCase()); // Outputs "hello world"

## Finding a Character in a String

The indexOf() method returns the **index** (the position) of the first occurrence of a specified text in a string (including whitespace):

### **Example**

String txt = "Please locate where 'locate' occurs!";

System.out.println(txt.indexOf("locate")); // Outputs 7

Java counts positions from zero.  
0 is the first position in a string, 1 is the second, 2 is the third ...

## String Concatenation

The + operator can be used between strings to combine them. This is called **concatenation**:

String firstName = "John";

String lastName = "Doe";

System.out.println(firstName + " " + lastName);

Note that we have added an empty text (" ") to create a space between firstName and lastName on print.

You can also use the concat() method to concatenate two strings:

String firstName = "John ";

String lastName = "Doe";

System.out.println(firstName.concat(lastName));

## Adding Numbers and Strings

WARNING!

Java uses the + operator for both addition and concatenation.

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

### **Example**

int x = 10;

int y = 20;

int z = x + y; // z will be 30 (an integer/number)

If you add two strings, the result will be a string concatenation:

### **Example**

String x = "10";

String y = "20";

String z = x + y; // z will be 1020 (a String)

If you add a number and a string, the result will be a string concatenation:

### **Example**

String x = "10";

int y = 20;

String z = x + y; // z will be 1020 (a String)

# **Java Special Characters**

## Strings - Special Characters

Because strings must be written within quotes, Java will misunderstand this string, and generate an error:

String txt = "We are the so-called "Vikings" from the north.";

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (\) escape character turns special characters into string characters:

|  |  |  |
| --- | --- | --- |
| **Escape character** | **Result** | **Description** |
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash |

The sequence \"  inserts a double quote in a string:

### **Example**

String txt = "We are the so-called \"Vikings\" from the north.";

The sequence \'  inserts a single quote in a string:

### **Example**

String txt = "It\'s alright.";

The sequence \\  inserts a single backslash in a string:

### **Example**

String txt = "The character \\ is called backslash.";

Other common escape sequences that are valid in Java are:

|  |  |  |
| --- | --- | --- |
| **Code** | **Result** | **Try it** |
| \n | New Line | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_newline) |
| \r | Carriage Return | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_r) |
| \t | Tab | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_tab) |
| \b | Backspace | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_b) |
| \f | Form Feed |  |

# **Java Math**

The Java Math class has many methods that allows you to perform mathematical tasks on numbers.

## Math.max(x,y)

The Math.max(x,y) method can be used to find the highest value of x and y: Math.max(5, 10);

## Math.min(x,y)

The Math.min(x,y) method can be used to find the lowest value of x and y:

Math.min(5, 10);

## Math.sqrt(x)

The Math.sqrt(x) method returns the square root of x:

Math.sqrt(64);

## Math.abs(x)

The Math.abs(x) method returns the absolute (positive) value of x:

Math.abs(-4.7);

## Random Numbers

Math.random() returns a random number between 0.0 (inclusive), and 1.0 (exclusive):

Math.random();

To get more control over the random number, for example, if you only want a random number between 0 and 100, you can use the following formula:

int randomNum = (int)(Math.random() \* 101); // 0 to 100

## Java Booleans

Very often, in programming, you will need a data type that can only have one of two values, like:

* YES / NO
* ON / OFF
* TRUE / FALSE

For this, Java has a boolean data type, which can store true or false values.

boolean isJavaFun = true;

boolean isFishTasty = false;

System.out.println(isJavaFun); // Outputs true

System.out.println(isFishTasty); // Outputs false

int x = 10;

int y = 9;

System.out.println(x > y); // returns true, because 10 is higher than 9

In the examples below, we use the **equal to** (==) operator to evaluate an expression:

System.out.println(10 == 15); // returns false, because 10 is not equal to 15

Let's think of a "real life example" where we need to find out if a person is old enough to vote.

In the example below, we use the >= comparison operator to find out if the age (25) is **greater than** OR **equal to** the voting age limit, which is set to 18:

int myAge = 25;

int votingAge = 18;

System.out.println(myAge >= votingAge);

# **Java If ... Else**

## Java Conditions and If Statements

You already know that Java supports the usual logical conditions from mathematics:

* Less than: a < b
* Less than or equal to: a <= b
* Greater than: a > b
* Greater than or equal to: a >= b
* Equal to a == b
* Not Equal to: a != b

You can use these conditions to perform different actions for different decisions.

Java has the following conditional statements:

* Use if to specify a block of code to be executed, if a specified condition is true
* Use else to specify a block of code to be executed, if the same condition is false
* Use else if to specify a new condition to test, if the first condition is false
* Use switch to specify many alternative blocks of code to be executed

## The if Statement

Use the if statement to specify a block of Java code to be executed if a condition is true.

if (condition) {

// block of code to be executed if the condition is true

}

Note that if is in lowercase letters. Uppercase letters (If or IF) will generate an error.

if (20 > 18) {

System.out.println("20 is greater than 18");

}

int x = 20;

int y = 18;

if (x > y) {

System.out.println("x is greater than y");

}

## The else Statement

Use the else statement to specify a block of code to be executed if the condition is false.

if (condition) {

// block of code to be executed if the condition is true

} else {

// block of code to be executed if the condition is false

}

int time = 20;

if (time < 18) {

System.out.println("Good day.");

} else {

System.out.println("Good evening.");

}

// Outputs "Good evening."

In the example above, time (20) is greater than 18, so the condition is false. Because of this, we move on to the else condition and print to the screen "Good evening". If the time was less than 18, the program would print "Good day".

int time = 22;

if (time < 10) {

System.out.println("Good morning.");

} else if (time < 18) {

System.out.println("Good day.");

} else {

System.out.println("Good evening.");

}

// Outputs "Good evening."In the example above, time (22) is greater than 10, so the **first condition** is false. The next condition, in the else

#### **Example explained**

if statement, is also false, so we move on to the else condition since **condition1** and **condition2** is both false - and print to the screen "Good evening".

However, if the time was 14, our program would print "Good day."

**ternary operator:** It can be used to replace multiple lines of code with a single line, and is most often used to replace simple if else statements:

**Syntax** variable *= (*condition*) ?* expressionTrue *:*  expressionFalse*;*

Instead of writing: int time = 20;

if (time < 18) {

System.out.println("Good day.");

} else {

System.out.println("Good evening.");

}

You can simply write: int time = 20;

String result = (time < 18) ? "Good day." : "Good evening.";

System.out.println(result);

## Java Switch Statements

Instead of writing **many** if..else statements, you can use the switch statement.

The switch statement selects one of many code blocks to be executed:

switch(expression) {

case x:

// code block

break;

case y:

// code block

break;

default:

// code block

}

This is how it works:

* The switch expression is evaluated once.
* The value of the expression is compared with the values of each case.
* If there is a match, the associated block of code is executed.
* The break and default keywords are optional, and will be described later in this chapter

The example below uses the weekday number to calculate the weekday name:

int day = 4;

switch (day) {

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

case 3:

System.out.println("Wednesday");

break;

case 4:

System.out.println("Thursday");

break;

case 5:

System.out.println("Friday");

break;

case 6:

System.out.println("Saturday");

break;

case 7:

System.out.println("Sunday");

break;

}

// Outputs "Thursday" (day 4)

## The break Keyword

When Java reaches a break keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match is found, and the job is done, it's time for a break. There is no need for more testing.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

## The default Keyword

The default keyword specifies some code to run if there is no case match:

int day = 4;

switch (day) {

case 6:

System.out.println("Today is Saturday");

break;

case 7:

System.out.println("Today is Sunday");

break;

default:

System.out.println("Looking forward to the Weekend");

}

// Outputs "Looking forward to the Weekend"

Note that if the default statement is used as the last statement in a switch block, it does not need a break.

## Loops

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

## Java While Loop

The while loop loops through a block of code as long as a specified condition is true:

while (condition) {

*// code block to be executed*

}

int i = 0;

while (i < 5) {

System.out.println(i);

i++;

}**Note:** Do not forget to increase the variable used in the condition, otherwise the loop will never end!

## The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

### **Syntax**

do {

*// code block to be executed*

}

while (condition);

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

int i = 0;  
do {

System.out.println(i);

i++;

}

while (i < 5);

## Java For Loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

for (*statement 1*; *statement 2*; *statement 3*) {

*// code block to be executed*

}

**Statement 1** is executed (one time) before the execution of the code block.

**Statement 2** defines the condition for executing the code block.

**Statement 3** is executed (every time) after the code block has been executed.

The example below will print the numbers 0 to 4:

for (int i = 0; i < 5; i++) {

System.out.println(i);

}

#### **Example explained**

Statement 1 sets a variable before the loop starts (int i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5). If the condition is true, the loop will start over again, if it is false, the loop will end.

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

## Nested Loops

It is also possible to place a loop inside another loop. This is called a **nested loop**.

The "inner loop" will be executed one time for each iteration of the "outer loop":

for (int i = 1; i <= 2; i++) {

System.out.println("Outer: " + i); // Executes 2 times

// Inner loop

for (int j = 1; j <= 3; j++) {

System.out.println(" Inner: " + j); // Executes 6 times (2 \* 3)

}

}

## For-Each Loop

There is also a "**for-each**" loop, which is used exclusively to loop through elements in an [**array**](https://www.w3schools.com/java/java_arrays.asp):

### **Syntax**

for (type variableName : arrayName) {

*// code block to be executed*

}

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars) {

System.out.println(i);

}