What is Java?

Java is a popular programming language, created in 1995.

It is owned by Oracle, and more than **3 billion** devices run Java.

It is used for:

* Mobile applications (especially Android apps)
* Desktop applications
* Web applications
* Web servers and application servers
* Games
* Database connection
* And much, much more!

Why Use Java?

* Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
* It is one of the most popular programming language in the world
* It is easy to learn and simple to use
* It is open-source and free
* It is secure, fast and powerful
* It has a 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

## Java Install

Some PCs might have Java already installed.

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

If Java is installed, you will see something like this (depending on version):

java version "11.0.1" 2018-10-16 LTS  
Java(TM) SE Runtime Environment 18.9 (build 11.0.1+13-LTS)  
Java Hotspot(TM) 64-Bit Server VM 18.9 (build 11.0.1+13-LTS, mixed mode)

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

## Java QuickStart

In Java, every application begins with a class name, and that class must match the filename.

Let's create our first Java file, called MyClass.java, which can be done in any text editor (like Notepad).

The file should contain a "Hello World" message, which is written with the following code:

MyClass.java

public class MyClass {

public static void main(String[] args) {

System.out.println("Hello World");

}

}

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

C:\Users\*Your Name*>javac MyClass.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 MyClass" to run the file:

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

The output should read:

Hello World

## Java Syntax

In the previous chapter, we created a Java file called MyClass.java, and we used the following code to print "Hello World" to the screen:

MyClass.java

public class MyClass {

public static void main(String[] args) {

System.out.println("Hello World");

}

}

### **Example explained**

Every line of code that runs in Java must be inside a class. In our example, we named the class **MyClass**. A class should always start with an uppercase first letter.

**Note:** Java is case-sensitive: "MyClass" and "myclass" has different meaning.

The name of the java file **must match** the class name. When saving the file, save it using the class name and add ".java" to the end of the filename. To run the example above on your computer, make sure that Java is properly installed: Go to the [Get Started Chapter](https://www.w3schools.com/java/java_getstarted.asp) for how to install Java. The output should be:

Hello World

## The main Method

The main() method is required and you will see it in every Java program:

public static void **main**(String[] args)

Any code inside the main() method will be executed. You don't have to understand the keywords before and after main. You will get to know them bit by bit while reading this tutorial.

For now, just remember that every Java program has a class name which must match the filename, and that every program must contain the main() method.

## System.out.println()

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

public static void main(String[] args) {

**System.out.println("Hello World");**

}

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

**Note:** Each code statement must end with a semicolon.

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

## Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

### **Syntax**

*type variable = value;*

Where type is one of Java's types (such as int or String), and variable is the name of the variable (such as **x** or **name**). The **equal sign** is used to assign values to the variable.

To create a variable that should store text, look at the following example:

### **Example**

Create a variable called **name** of type String and assign it the value "**John**":

String name = "John";

System.out.println(name);

### **Example**

Create a variable called **myNum** of type int and assign it the value **15**:

int myNum = 15;

System.out.println(myNum);

### **Example**

Change the value of myNum from 15 to 20:

int myNum = 15;

myNum = 20; // myNum is now 20

System.out.println(myNum);

## Final Variables

However, you can add the final keyword if you don't want others (or yourself) to overwrite existing values (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";

## Display Variables

The println() method is often used to display variables.

To combine both text and a variable, use the + character:

### **Example**

String name = "John";

System.out.println("Hello " + name);

### **Example**

String firstName = "John ";

String lastName = "Doe";

String fullName = firstName + lastName;

System.out.println(fullName);

For numeric values, the + character works as a mathematical operator (notice that we use int (integer) variables here):

### **Example**

int x = 5;

int y = 6;

System.out.println(x + y); // Print the value of x + y

From the example above, you can expect:

* x stores the value 5
* y stores the value 6
* Then we use the println() method to display the value of x + y, which is **11**

## Declare Many Variables

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

### **Example**

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

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

## Java Identifiers

All Java **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

**Note:** It is recommended to use descriptive names in order to create understandable and maintainable code:

### **Example**

// Good

int minutesPerHour = 60;

// OK, but not so easy to understand what **m** actually is

int m = 60;

The general rules for constructing names for variables (unique identifiers) are:

* Names can contain letters, digits, underscores, and dollar signs
* Names must begin with a letter
* Names should start with a lowercase letter and it 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

As explained in the previous chapter, a variable in Java must be a specified data type:

### **Example**

int myNum = 5; // Integer (whole number)

float myFloatNum = 5.99f; // Floating point number

char myLetter = 'D'; // Character

boolean myBool = true; // Boolean

String myText = "Hello"; // String

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**

byte myNum = 100;

System.out.println(myNum);

### **Short**

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

### **Example**

short myNum = 5000;

System.out.println(myNum);

### **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);

### **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);

## 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.

### **Float**

The float data type can store fractional numbers from 3.4e−038 to 3.4e+038. Note that you should end the value with an "f":

### **Example**

float myNum = 5.75f;

System.out.println(myNum);

### **Double**

The double data type can store fractional numbers from 1.7e−308 to 1.7e+308. Note that you should end the value with a "d":

### **Example**

double myNum = 19.99d;

System.out.println(myNum)

## Booleans

A boolean data type is declared with the boolean keyword and can only take the values true or false:

### **Example**

boolean isJavaFun = true;

boolean isFishTasty = false;

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

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

Boolean values are mostly used for conditional testing, which you will learn more about in a later chapter.

## Characters

The char data type is used to store a **single** character. The character must be surrounded by single quotes, like 'A' or 'c':

### **Example**

char myGrade = 'B';

System.out.println(myGrade);

Alternatively, you can use ASCII values to display certain characters:

### **Example**

char a = 65, b = 66, c = 67;

System.out.println(a);

System.out.println(b);

System.out.println(c);

## 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);

## ava 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 MyClass {

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 MyClass {

public static void main(String[] args) {

double myDouble = 9.78;

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.

In the example below, we use the + **operator** to add together two values:

### **Example**

int x = 100 + 50;

Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

### **Example**

int sum1 = 100 + 50; // 150 (100 + 50)

int sum2 = sum1 + 250; // 400 (150 + 250)

int sum3 = sum2 + sum2; // 800 (400 + 400)

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

## Java Assignment Operators

Assignment operators are used to assign values to variables.

In the example below, we use the **assignment** operator (=) to assign the value **10** to a variable called **x**:

### **Example**

int x = 10;

The **addition assignment** operator (+=) adds a value to a variable:

### **Example**

int x = 10;

x += 5;