Java for beginners:

Java Introduction

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 (specially 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 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++ and C#, it makes it easy for programmers to switch to Java or vice versa

Java Syntax

Main.java

```java
public class Main {

  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 **Main**. 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 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:

```java
public static void main(String[] args)
```

Any code inside the `main()` method will be executed. Don't worry about 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");

}
```

Java Output / Print

Print Text

You learned from the previous chapter that you can use the `println()` method to output values or print text in Java:

Example

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

You can add as many `println()` methods as you want. Note that it will add a new line for each method:

Example

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

System.out.println("I am learning Java.");

System.out.println("It is awesome!");
```

Double Quotes

When you are working with text, it must be wrapped inside double quotations marks `""`.

If you forget the double quotes, an error occurs:

Example

```
System.out.println("This sentence will work!");

System.out.println(This sentence will produce an error);
```

The Print() Method

There is also a `print()` method, which is similar to `println()`.

The only difference is that it does not insert a new line at the end of the output:

Example

```
System.out.print("Hello World! ");

System.out.print("I will print on the same line.");
```

Java Output / Print

Print Text

You learned from the previous chapter that you can use the `println()` method to output values or print text in Java:

Example

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

You can add as many `println()` methods as you want. Note that it will add a new line for each method:

Example

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

System.out.println("I am learning Java.");
```

```
System.out.println("It is awesome!");
```

Double Quotes

When you are working with text, it must be wrapped inside double quotations marks "".

If you forget the double quotes, an error occurs:

Example

```
System.out.println("This sentence will work!");

System.out.println(This sentence will produce an error);
```

The Print() Method

There is also a `print()` method, which is similar to `println()`.

The only difference is that it does not insert a new line at the end of the output:

Example

```
System.out.print("Hello World! ");

System.out.print("I will print on the same line.");
```

Java Output Numbers

Print Numbers

You can also use the `println()` method to print numbers.

However, unlike text, we don't put numbers inside double quotes:

Example

```
System.out.println(3);

System.out.println(358);

System.out.println(50000);
```

You can also perform mathematical calculations inside the `println()` method:

Example

```
System.out.println(3 + 3);
```

Example

```
System.out.println(2 * 5);
```

Java Comments

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

Single-line Comments

Single-line comments start with two forward slashes (`//`).

Any text between `//` and the end of the line is ignored by Java (will not be executed).

This example uses a single-line comment before a line of code:

Example

```
// This is a comment

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

This example uses a single-line comment at the end of a line of code:

Example

```
System.out.println("Hello World"); // This is a comment
```

Java Multi-line Comments

Multi-line comments start with `/*` and ends with `*/`.

Any text between `/*` and `*/` will be ignored by Java.

This example uses a multi-line comment (a comment block) to explain the code:

Example

```
/* The code below will print the words Hello World

to the screen, and it is amazing */

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

Java Variables

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 variableName = value;
```

Where *type* is one of Java's types (such as `int` or `String`), and *variableName* 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);
```

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

Example

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

```
int myNum = 15;

System.out.println(myNum);
```

You can also declare a variable without assigning the value, and assign the value later:

Example

```
int myNum;

myNum = 15;
```

```
System.out.println(myNum);
```

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

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

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";
```

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);
```

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

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
```

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`, Arrays and Classes (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 9,223,3
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digit
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

Java Numbers

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`.

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

```
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.

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);
```

Double Example

```
double myNum = 19.99d;

System.out.println(myNum);
```

Boolean Types

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 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
```

Java Characters

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, 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);
```

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)
```

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
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Java Strings

Strings are used for storing text.

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

Example

Create a variable of type `String` and assign it a value:

```
String greeting = "Hello";
```

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:

Example

```
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"
```

String Concatenation

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

Example

```
String firstName = "John";

String lastName = "Doe";

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

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

Example

```
String firstName = "John ";

String lastName = "Doe";

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

Java Special Characters

❮ PreviousNext ❯

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
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
