**Java** *is* a **class-based**, **object-oriented** **programming language** that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let application developers write once, run anywhere (WORA),[17] meaning that *compiled* Java code can run on all platforms that support Java without the need for recompilation.[18] Java applications are typically compiled to **bytecode** that can run on any Java virtual machine (**JVM**) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but has fewer **low-level** facilities than either of them. The Java runtime provides dynamic capabilities (such as reflection and runtime code modification) that are typically not available in traditional compiled languages. As of 2019, Java was one of the most popular programming languages in use according to GitHub,[19][20] particularly for client-server web applications, with a reported 9 million developers.[21]

The latest version is **Java 15**, released in September 2020, with Java 11, a currently supported long-term support (LTS) version, released on September 25, 2018; Oracle released for the legacy Java 8 LTS the last zero-cost public update in January 2019 for commercial use, although it will otherwise still support Java 8 with public updates for personal use up to at least December 2020. Other vendors have begun to offer zero-cost builds of OpenJDK 8 and 11 that are still receiving security and other upgrades.

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

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

**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 start with two forward slashes (**//**).

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

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

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

**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
* 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:

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):

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)

**Operators** are used to *perform operations* on variables and **values**.

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

Java divides the operators into the following groups:

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

**Assignment** operators are used to assign values to variables.

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 take the values **true** or **false**.

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
* Use the if statement to specify a block of Java code to be executed if a condition is true.
* n the example above we use two variables, **x** and **y**, to test whether x is greater than y (using the > operator). As x is 20, and y is 18, and we know that 20 is greater than 18, we print to the screen that "x is greater than y".
* Use the else statement to specify a block of code to be executed if the condition is false.
* here is also a short-hand if else, which is known as the **ternary operator** because it consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements:
* variable *= (*condition*) ?* expressionTrue *:*  expressionFalse*;*
* **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**.
* The **while**loop loops through a block of code as long as a specified condition is true:

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.

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:

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:

**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:

There is also a "**for-each**" loop, which is used exclusively to loop through elements in an **array**:

You have already seen the **break** statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a **loop**.

The **continue** statement breaks one **iteration** (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

String[] cars;

We have now declared a variable that holds an array of strings. To insert values to it, we can use an array literal - *place* the values in a comma-separated list, inside curly braces:

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

To create an array of integers, you could write:

int[] myNum = {10, 20, 30, 40};

You *access* an array element by referring to the index number.

To *change* the value of a specific element, refer to the index number:

cars[0] = "Opel";

## Array Length

To find out how many elements an array has, use the length property:

### **Example**

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

System.out.println(cars.length);

// Outputs 4

## Loop Through an Array

You can loop through the array elements with the for loop and use the length property to specify how many times the loop should run.

## Loop Through an Array with For-Each

If you compare the for loop and **for-each** loop, you will see that the **for-each** method is easier to write, it does not require a **counter** (using the length property), and it is more readable.

## Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

To access the elements of the **myNumbers** array, specify two indexes: one for the array, and one for the element inside that array.

We can also use a for loop inside another for loop to get the elements of a two-dimensional array (we still have to point to the two **indexes**)