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

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 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 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

Java Syntax

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

Main.java

```
public class Main {
  public static void main(String[] args) {
    System.out.println("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. 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.

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).

You should also note that each code statement must end with a semicolon (;).

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

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

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

We 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.");
```

Print Numbers

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

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

ExampleGet your own Java Server

```
System.out.println(3);
System.out.println(358);
System.out.println(50000);
```

Java Comments

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:

ExampleGet your own Java Server

```
// 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");
```

Declaring (Creating) Variables

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

Syntax

```
type variableName = value;
```

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

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:

```
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.

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

The if Statement

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

Syntax

```
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.

In the example below, we test two values to find out if 20 is greater than 18. If the condition is true, print some text:

Example

```
if (20 > 18) {
  System.out.println("20 is greater than 18");
}
```

The else Statement

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

Syntax

```
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."
```

The else if Statement

Use the else if statement to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

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:

```
public class Main {
  public static void main(String[] args) {
   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;
   }
  }
```

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:

Syntax

```
while (condition) {
  // code block to be executed
}
```

Example

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;
}
```

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);
```

```
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:

Syntax

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

```
for (int i = 0; i < 5; i++) {
  System.out.println(i);
}
```

For-Each Loop

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

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

Java User Input

The Scanner class is used to get user input, and it is found in the java.util package.

To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read Strings:

Example

```
import java.util.Scanner;  // Import the Scanner class

class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username");

    String userName = myObj.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user input
  }
}
```

Input Types

In the example above, we used the nextLine() method, which is used to read Strings. To read other types, look at the table below:

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

```
import java.util.Scanner;

class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);

    System.out.println("Enter name, age and salary:");

    // String input
    String name = myObj.nextLine();

    // Numerical input
    int age = myObj.nextInt();
    double salary = myObj.nextDouble();

    // Output input by user
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Salary: " + salary);
  }
}
```