Java Programming Fundamentals

1- What is Java?

Java is a popular programming language, created in 1995. It is a powerful general-purpose programming language. According to Oracle, the company that owns Java, Java runs on 3 billion devices worldwide, which makes Java one of the most popular programming languages.

It is used for:

- Mobile applications (specially Android apps) and desktop applications.
- Web applications
- Games
- Database connection and big data processing
- And much, much more!

Our Java tutorial will guide you to learn Java one step at a time.

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

Java Programming Fundamentals

3- 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. The file should contain a "Hello World" message and print it to the screen, which is written with the following code:

MyClass.java

```
public class MyClass {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

When you run the program, the output will be:

```
Hello World
```

Example explained

```
Public class MyClass { ... }
```

In Java, every application begins with a class definition. Every line of code 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.

```
public static void main(String[] args) { ... }
```

This is the main method. Every application in Java must contain the main method.

The Java compiler starts executing the code from the main method. The main method must be inside the class definition.

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

The following code prints the string inside quotation marks `Hello World` to standard output (your screen). Notice, this statement is inside the main function, which is inside the class definition.

4- Java Variables and (Primitive) Data Types

In this tutorial, you will learn about variables, how to create them, and different data types that Java programming language supports for creating variables.

Java Variables

A variable is a location in memory (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name (identifier).

How to declare variables in Java?

Here's an example to declare a variable in Java.

```
int price = 80;
```

Here, *price* is a variable of *int* data type and is assigned value 80. Meaning, the *price* variable can store integer values. You will learn about Java data types in detail later in the article.

In the example, we have assigned value to the variable during declaration. However, it's not mandatory. You can declare variables without assigning the value, and later you can store the value as you wish. For example,

```
int price;
price = 80;
```

The value of a variable can be changed in the program, hence the name 'variable'. For example,

```
int price = 80;
... .. ...
price = 90;
```

Java is a statically-typed language. It means that all variables must be declared before they can be used.

Also, you cannot change the data type of a variable in Java within the same scope. So, you cannot do something like this.

```
int price = 80;
... .. ...
float price;
```

Java Primitive Data Types

In Java all variables must be declared before they can be used.

```
int price;
```

Here, *price* is a variable, and the data type of the variable is `int`. The *int* data type determines that the *price* variable can only contain integers.

In simple terms, a variable's data type determines the values a variable can store. There are 8 data types predefined in Java programming language, known as primitive data types.

Primitive Data Types

boolean

- The `boolean` data type has two possible values, either `true` or `false`.
- Default value: `false`.
- They are usually used for true/false conditions. For example,

```
class BooleanExample {
    public static void main(String[] args) {

        boolean flag = true;
        System.out.println(flag);
    }
}
```

Output: true

byte

- The `byte` data type can have values from -128 to 127 (1 byte)
- It's used instead of `int` or other integer data types to save memory if it's certain that the value of a variable will be within [-128, 127].
- Default value: 0
- Example:

```
class ByteExample {
    public static void main(String[] args) {

        byte range;
        range = 124;
        System.out.println(range);
    }
}
```

Output: 124

short

- The `short` data type can have values from -32768 to 32767 (2 bytes)
- It's used instead of other integer data types to save memory if it's certain that the value of the variable will be within [-32768, 32767].
- Default value: 0
- Example:

```
class ShortExample {
    public static void main(String[] args) {

        short temperature;
        temperature = -200;
        System.out.println(temperature);
    }
}
```

Output: -200

int

- The `int` data type can have values from -2,147,483,648 to 2,147,483,647 (4 bytes)
- Default value: 0
- Example:

```
class IntExample {
    public static void main(String[] args) {

        int range = -4250000;
        System.out.println(range);
    }
}
```

Output: -4250000

long

- The `long` data type can have values from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (8 bytes).
- Default value: 0

- Example:

```
class LongExample {
    public static void main(String[] args) {

        long range = -42332200000L;
        System.out.println(range);
    }
}
```

Output: -42332200000

float

- Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits (4 bytes)
- Default value: 0.0 (0.0f)
- Example:

```
class FloatExample {
    public static void main(String[] args) {

        float number = -42.3f;
        System.out.println(number);
    }
}
```

Output: `-42.3`

Notice that, we have used `-42.3f` instead of `-42.3`. It's because `-42.3` is a `double` literal. To tell the compiler to treat `-42.3` as `float` rather than `double`, you need to use *f* or *F*.

double

- Stores fractional numbers. Sufficient for storing 15 decimal digits (8 bytes)
- Default value: 0.0 (0.0d)
- Example:

```
class DoubleExample {
    public static void main(String[] args) {

        double number = -42.3;
        System.out.println(number);
    }
}
```

Output: - 42.3

char

- It's a 16-bit Unicode character (2 bytes)
- The minimum value of the char data type is `'\u0000'` (0). The maximum value of the char data type is `'\uffff'`.
- Stores a single character/letter or ASCII values
- Example:

```
class CharExample {
    public static void main(String[] args) {

        char letter = '\u0051';
        System.out.println(letter);
    }
}
```

Output:`Q`

You get the output *Q* because the Unicode value of *Q* is `'\u0051'`.

Here is another example:

```
class CharExample {
    public static void main(String[] args) {

        char letter1 = '9';
        System.out.println(letter1);

        char letter2 = 65;
        System.out.println(letter2);

    }
}
```

Output:
```
9
A
```

When you print *letter1*, you will get *9* because *letter1* is assigned character `'9'`.

When you print *letter2*, you get *A* because the ASCII value of `'A'` is 65.

String

Java also provides support for character strings via `java.lang.String` class. Here's how you can create a String object in Java:

```
myString = "Programming is awesome";
```

5- Java Operators

Operators are special symbols (characters) that carry out operations on operands (variables and values). For example, `+` is an operator that performs addition.

Java divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Relational 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 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;
```

A list of all assignment operators:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Java Relational (Comparison) Operators

Comparison operators are used to compare two values. It determines the relationship between the two operands. Depending on the relationship, it is evaluated to either `true` or `false`.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Java Logical Operators

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

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

6- Java Basic Input and Output

In Java, you can simply use

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

System.out.print(); or

System.out.printf();
```

Java Programming Fundamentals

Difference between println(), print() and printf()

- `print()` - It prints string inside the quotes.
- `println()` - It prints string inside the quotes similar like `print()` method. Then the cursor moves to the beginning of the next line.
- `printf()` - Tt provides string formatting (similar to printf in C/C++ programming).

Example: print() and println()

```
class Output {
    public static void main(String[] args) {

        System.out.println("1. println ");
        System.out.println("2. println ");

        System.out.print("1. print ");
        System.out.print("2. print");
    }
}
```

Output:

```
1. println
2. println
1. print 2. print
```

Example: Print Concatenated Strings

```
class PrintVariables {
    public static void main(String[] args) {

        Double number = -10.6;

        System.out.println("I am " + "awesome.");
        System.out.println("Number = " + number);
    }
}
```

Output:

```
I am awesome.
Number = -10.6
```

In the above example, notice the line,

```
System.out.println("I am " + "awesome.");
```

Here, we have used the `+` operator to concatenate (join) the two strings: *"I am "* and *"awesome."*.

And also, the line,

```
System.out.println("Number = " + number);
```

Here, first the value of variable *number* is evaluated. Then, the value is concatenated to the string: *"Number = "*

Java Input

in this tutorial, you will learn to get input from user using the object of `Scanner` class.

In order to use the object of `Scanner`, we need to import `java.util.Scanner` package.

```
import java.util.Scanner;
```

Then, we need to create an object of the `Scanner` class. We can use the object to take input from the user.

```
// create an object of Scanner
Scanner input = new Scanner(System.in);

// take input from the user
int number = input.nextInt();
```

Example: Get Integer Input From the User

```
import java.util.Scanner;

class Input {
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        System.out.print("Enter an integer: ");
        int number = input.nextInt();
        System.out.println("You entered " + number);

        // closing the scanner object
        input.close();
    }
}
```

Output:

```
Enter an integer: 23
You entered 23
```

In the above example, we have created an object named *input* of the `Scanner` class. We then call the `nextInt()` method of the `Scanner` class to get an integer input from the user.

Similarly, we can use `nextLong()`, `nextFloat()`, `nextDouble()`, and `next()` methods to get `long`, `float`, `double`, and `string` input respectively from the user.

Example: Get float, double and String Input

```
import java.util.Scanner;

class Input {
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        // Getting float input
        System.out.print("Enter float: ");
        float myFloat = input.nextFloat();
        System.out.println("Float entered = " + myFloat);

        // Getting double input
        System.out.print("Enter double: ");
        double myDouble = input.nextDouble();
        System.out.println("Double entered = " + myDouble);

        // Getting String input
        System.out.print("Enter text: ");
        String myString = input.next();
        System.out.println("Text entered = " + myString);
    }
}
```

Output:

```
Enter float: 2.343
Float entered = 2.343
Enter double: -23.4
Double entered = -23.4
Enter text: Hey!
Text entered = Hey!
```

7- 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"
```

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
```

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));
```

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.";
```

he sequence `\\` inserts a single backslash in a string:

Example

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

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";
int y = 20;
String z = x + y;    // z will be 1020 (a String)
```

8- 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 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");
```

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");
```

9- Java if, if...else Statement

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");
}
```

We can also test variables:

Example

```
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`.

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
}
```

Example

```
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`.

Java Programming Fundamentals

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
}
```

Example

```
int time = 22;
if (time < 10) {
  System.out.println("Good morning.");
} else if (time < 20) {
  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 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."

Short Hand If...Else (Ternary Operator)

There 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:

Syntax

```
variable = (condition) ? expressionTrue :
expressionFalse;
```

Instead of writing:

Example

```
int time = 20;
if (time < 18) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
```

You can simply write:

Example

```
int time = 20;
String result = (time < 18) ? "Good day." : "Good evening.";
System.out.println(result);
```

10- 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
 }
```

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:

Example

```
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.

Another Example

This example will only print even values between 0 and 10:

Example

```
for (int i = 0; i <= 10; i = i + 2) {
  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**:

Syntax

```
for (type variableName : arrayName) {
  // code block to be executed
}
```

The following example outputs all elements in the **cars** array, using a "**for-each**" loop:

Example

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

Output:

Volvo
BMW
Ford
Mazda

11- 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
}
```

In the example below, the code in the loop will run, over and over again, as long as a variable (i) is less than 5:

Example

```
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:

Example

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

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