
Java Hello World Program

In this tutorial, you will learn to write "Hello World" program in Java.

A "Hello, World!" is a simple program that outputs `Hello, World!` on the screen. Since it's a very simple program, it's often used to introduce a new programming language to a newbie.

Let's explore how Java "Hello, World!" program works.

> **Note:** You can use our online Java compiler to run Java programs.

Java "Hello, World!" Program

```
// Your First Program

class HelloWorld {

    public static void main(String[] args) {

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

    }

}
```

Output

```
Hello, World!
```

How Java "Hello, World!" Program Works?

1. `// Your First Program`

In Java, any line starting with `//` is a comment. Comments are intended for users reading the code to understand the intent and functionality of the program. It is completely ignored by the Java compiler (an application that translates Java program to Java bytecode that computer can execute). To learn more, visit Java comments.

---

2. `class HelloWorld { ... }`

   In Java, every application begins with a class definition. In the program, `HelloWorld` is the name of the class, and the class definition is:

```
class HelloWorld {

... .. ...

}
```

3. For now, just remember that every Java application has a class definition, and the name of the class should match the filename in Java.

4. `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.

   How does it work? Good question. However, we will not discuss it in this article. After all, it's a basic program to introduce Java programming language to a newbie. We will learn the meaning of `public`, `static`, `void`, and how methods work? in later chapters.

   For now, just remember that the main function is the entry point of your Java application, and it's mandatory in a Java program. The signature of the main method in Java is:

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

... .. ...

}
```

5.

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

   The code above is a print statement. It prints the text `Hello, World!` to standard output (your screen). The text inside the quotation marks is called String in Java.

7.

8.

   Notice the print statement is inside the main function, which is inside the class definition.

9.

Things to take away

- Every valid Java Application must have a class definition (that matches the filename).
- The main method must be inside the class definition.
- The compiler executes the codes starting from the main function.

This is a valid Java program that does nothing.

```
public class HelloWorld {

    public static void main(String[] args) {

        // Write your code here

    }

}
```

Don't worry if you don't understand the meaning of `class`, `static`, methods, and so on for now.

Java JDK, JRE and JVM

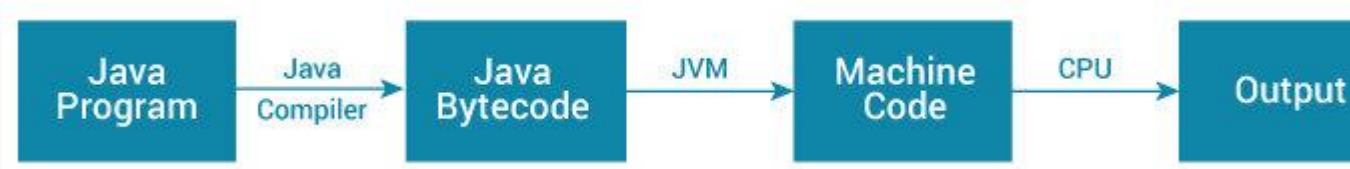
In this tutorial, you will learn about JDK, JRE, and JVM. You will also learn the key differences between them.

What is JVM?

JVM (Java Virtual Machine) is an abstract machine that enables your computer to run a Java program.

When you run the Java program, Java compiler first compiles your Java code to bytecode. Then, the JVM translates bytecode into native machine code (set of instructions that a computer's CPU executes directly).

Java is a platform-independent language. It's because when you write Java code, it's ultimately written for JVM but not your physical machine (computer). Since JVM executes the Java bytecode which is platform-independent, Java is platform-independent.



Working of Java Program

If you are interested in learning about JVM Architecture, visit The JVM Architecture Explained.

What is JRE?

JRE (Java Runtime Environment) is a software package that provides Java class libraries, Java Virtual Machine (JVM), and other components that are required to run Java applications.

JRE is the superset of JVM.

Java Runtime Environment

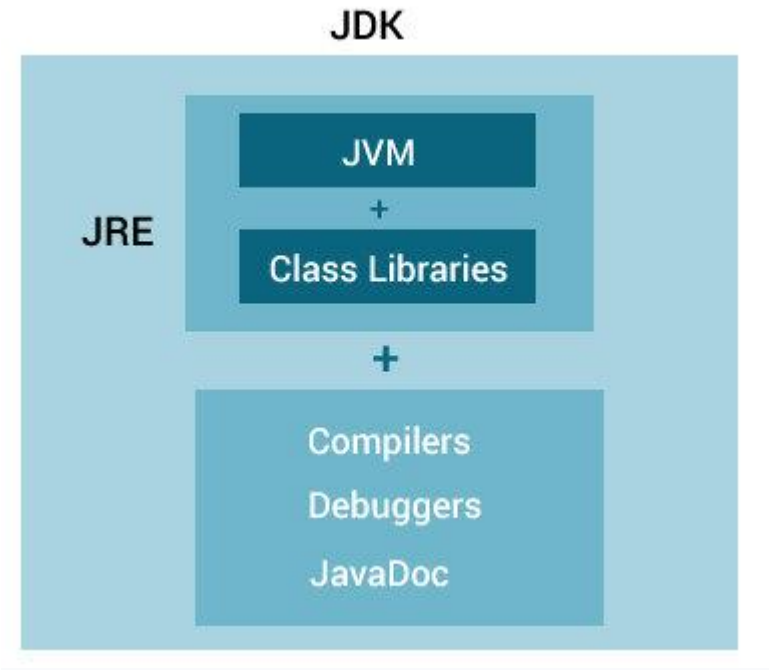
What is JDK?

JDK (Java Development Kit) is a software development kit required to develop applications in Java. When you download JDK, JRE is also downloaded with it.

In addition to JRE, JDK also contains a number of development tools (compilers, JavaDoc, Java Debugger, etc).

Java Development Kit

Relationship between JVM, JRE, and JDK.



Relationship between JVM, JRE, and JDK

Java Basic Input and Output

In this tutorial, you will learn simple ways to display output to users and take input from users in Java.

Java Output

In Java, you can simply use

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


System.out.print(); or


System.out.printf();
```

to send output to standard output (screen).

Here,

- `System` is a class
- `out` is a `public` `static` field: it accepts output data.

Don't worry if you don't understand it. We will discuss `class`, `public`, and `static` in later chapters.

Let's take an example to output a line.

```
class AssignmentOperator {

    public static void main(String[] args) {


        System.out.println("Java programming is interesting.");

    }

}
```

Output:

```
Java programming is interesting.
```

Here, we have used the `println()` method to display the string.

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()` - It 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
```

In the above example, we have shown the working of the `print()` and `println()` methods. To learn about the `printf()` method, visit Java printf().

Example: Printing Variables and Literals

```
class Variables {

    public static void main(String[] args) {



        Double number = -10.6;
```

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

        System.out.println(number);

    }

}
```

When you run the program, the output will be:

```
5

-10.6
```

Here, you can see that we have not used the quotation marks. It is because to display integers, variables and so on, we don't use quotation marks.

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

Java provides different ways to get input from the user. However, 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;
```

To learn more about importing packages in Java, visit Java Import Packages.

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 userint 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.

> **Note**: We have used the `close()` method to close the object. It is recommended to close the scanner object once the input is taken.

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!
```

Java Expressions, Statements and Blocks

In this tutorial, you will learn about Java expressions, Java statements, difference between expression and statement, and Java blocks with the help of examples.

In previous chapters, we have used expressions, statements, and blocks without much explaining about them. Now that you know about variables, operators, and literals, it will be easier to understand these concepts.

Java Expressions

A Java expression consists of variables, operators, literals, and method calls. To know more about method calls. For example,

```
int score;

score = 90;
```

Here, `score = 90` is an expression that returns an `int`. Consider another example,

```
Double a = 2.2, b = 3.4, result;

result = a + b - 3.4;
```

Here, `a + b - 3.4` is an expression.

```
if (number1 == number2)

    System.out.println("Number 1 is larger than number 2");
```

Here, `number1 == number2` is an expression that returns a boolean value. Similarly, `"Number 1 is larger than number 2"` is a string expression.

Java Statements

In Java, each statement is a complete unit of execution. For example,

```
int score = 9*5;
```

Here, we have a statement. The complete execution of this statement involves multiplying integers `9` and `5` and then assigning the result to the variable `score`.

In the above statement, we have an expression `9 * 5`. In Java, expressions are part of statements.

Expression statements

We can convert an expression into a statement by terminating the expression with a `;`. These are known as expression statements. For example,

```
// expression

number = 10// statement

number = 10;
```

In the above example, we have an expression `number = 10`. Here, by adding a semicolon (`;`), we have converted the expression into a statement (`number = 10;`).

Consider another example,

```
// expression

++number// statement

++number;
```

Similarly, `++number` is an expression whereas `++number;` is a statement.

Declaration Statements

In Java, declaration statements are used for declaring variables. For example,

```
Double tax = 9.5;
```

The statement above declares a variable `tax` which is initialized to `9.5`.

> **Note**: There are control flow statements that are used in decision making and looping in Java. You will learn about control flow statements in later chapters.

Java Blocks

A block is a group of statements (zero or more) that is enclosed in curly braces `{ }`. For example,

```
class Main {

    public static void main(String[] args) {


        String band = "Beatles";


        if (band == "Beatles") { // start of block

            System.out.print("Hey ");

            System.out.print("Jude!");

        } // end of block

    }

}
```

Output:

```
Hey Jude!
```

In the above example, we have a block `if {....}`.

Here, inside the block we have two statements:

- `System.out.print("Hey ");`
- `System.out.print("Jude!");`

However, a block may not have any statements. Consider the following examples,

```
class Main {

    public static void main(String[] args) {


        if (10 > 5) { // start of block
```

```
        } // end of block

    }

}
```

This is a valid Java program. Here, we have a block `if {...}`. However, there is no any statement inside this block.

```
class AssignmentOperator {

    public static void main(String[] args) {  // start of block



    } // end of block

}
```

Here, we have block `public static void main() {...}`. However, similar to the above example, this block does not have any statement.

Java Comments

In this tutorial, you will learn about Java comments, why we use them, and how to use comments in right way.

In computer programming, comments are a portion of the program that are completely ignored by Java compilers. They are mainly used to help programmers to understand the code. For example,

```
// declare and initialize two variablesint a =1;int b = 3;

// print the output

System.out.println("This is output");
```

Here, we have used the following comments,

- declare and initialize two variables
- print the output

Types of Comments in Java

In Java, there are two types of comments:

- single-line comment
- multi-line comment

Single-line Comment

A single-line comment starts and ends in the same line. To write a single-line comment, we can use the `//` symbol. For example,

```
// "Hello, World!" program example

 class Main {

    public static void main(String[] args) {

    {

        // prints "Hello, World!"

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

    }

}
```

Output:

```
Hello, World!
```

Here, we have used two single-line comments:

- `"Hello, World!" program example`
- `prints "Hello World!"`

The Java compiler ignores everything from `//` to the end of line. Hence, it is also known as **End of Line** comment.

Multi-line Comment

When we want to write comments in multiple lines, we can use the multi-line comment. To write multi-line comments, we can use the /*....*/ symbol. For example,

```
/* This is an example of  multi-line comment.
 * The program prints "Hello, World!" to the standard output.
 */
class HelloWorld {
    public static void main(String[] args) {
    {
        System.out.println("Hello, World!");
    }
}
```

Output:

```
Hello, World!
```

Here, we have used the multi-line comment:

```
/* This is an example of multi-line comment.
* The program prints "Hello, World!" to the standard output.
*/
```

This type of comment is also known as **Traditional Comment**. In this type of comment, the Java compiler ignores everything from `/*` to `*/`.

Use Comments the Right Way

One thing you should always consider that comments shouldn't be the substitute for a way to explain poorly written code in English. You should always write well structured and self explaining code. And, then use comments.

Some believe that code should be self-describing and comments should be rarely used. However, in my personal opinion, there is nothing wrong with using comments. We can

---

---

use comments to explain complex algorithms, regex or scenarios where we have to choose one technique among different technique to solve problems.

> **Note**: In most cases, always use comments to explain '**why**' rather than '**how**' and you are good to go.
