Unit 1. Introduction to Java

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) in the year 1995. James Gosling is known as the father of Java. Before Java, its name was Oak. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

**Platform**: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

**class** Simple{

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

     System.out.println("Hello Java");

    }

}

* 1. Properties of Java
  2. 1.2. Comparison of java with C++

|  |  |  |
| --- | --- | --- |
| **Comparison Index** | **C++** | **Java** |
| **Platform-independent** | C++ is platform-dependent. | Java is platform-independent. |
| **Mainly used for** | C++ is mainly used for system programming. | Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications. |
| **Design Goal** | C++ was designed for systems and applications programming. It was an extension of the [C programming language](https://www.javatpoint.com/c-programming-language-tutorial). | Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience. |
| **Goto** | C++ supports the [goto](https://www.javatpoint.com/cpp-goto-statement) statement. | Java doesn't support the goto statement. |
| **Multiple inheritance** | C++ supports multiple inheritance. | Java doesn't support multiple inheritance through class. It can be achieved by using [interfaces in java](https://www.javatpoint.com/interface-in-java). |
| **Operator Overloading** | C++ supports [operator overloading](https://www.javatpoint.com/cpp-overloading). | Java doesn't support operator overloading. |
| **Pointers** | C++ supports [pointers](https://www.javatpoint.com/cpp-pointers). You can write a pointer program in C++. | Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java. |
| **Compiler and Interpreter** | C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent. | Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent. |
| **Call by Value and Call by reference** | C++ supports both call by value and call by reference. | Java supports call by value only. There is no call by reference in java. |
| **Structure and Union** | C++ supports structures and unions. | Java doesn't support structures and unions. |
| **Thread Support** | C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support. | Java has built-in [thread](https://www.javatpoint.com/multithreading-in-java) support. |
| **Documentation comment** | C++ doesn't support documentation comments. | Java supports documentation comment (/\*\* ... \*/) to create documentation for java source code. |
| **Virtual Keyword** | C++ supports virtual keyword so that we can decide whether or not to override a function. | Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default. |
| **unsigned right shift >>>** | C++ doesn't support >>> operator. | Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator. |
| **Inheritance Tree** | C++ always creates a new inheritance tree. | Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the [inheritance](https://www.javatpoint.com/inheritance-in-java) tree in java. |
| **Hardware** | C++ is nearer to hardware. | Java is not so interactive with hardware. |
| **Object-oriented** | C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible. | Java is also an [object-oriented](https://www.javatpoint.com/java-oops-concepts) language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object. |

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main() {
4. cout << "Hello C++ Programming";
5. **return** 0;
6. }
7. **class** Simple{
8. **public** **static** **void** main(String args[]){
9. System.out.println("Hello Java");
10. }
11. }

1.3. Java Compiler

A **Java compiler** is a [compiler](https://en.wikipedia.org/wiki/Compiler) for the programming language [Java](https://en.wikipedia.org/wiki/Java_(programming_language)). The most common form of output from a Java compiler is [Java class files](https://en.wikipedia.org/wiki/Java_class_file) containing platform-neutral [Java bytecode](https://en.wikipedia.org/wiki/Java_bytecode),[[1]](https://en.wikipedia.org/wiki/Java_compiler#cite_note-1) but there are also compilers that output optimized [native machine code](https://en.wikipedia.org/wiki/Machine_code) for a particular hardware/[operating system](https://en.wikipedia.org/wiki/Operating_system) combination, most notably the now discontinued [GNU Compiler for Java](https://en.wikipedia.org/wiki/GNU_Compiler_for_Java).[[2]](https://en.wikipedia.org/wiki/Java_compiler#cite_note-2)

Most Java-to-bytecode compilers do virtually no [optimization](https://en.wikipedia.org/wiki/Optimization_(computer_science)), leaving this until [run time](https://en.wikipedia.org/wiki/Run_time_(program_lifecycle_phase)) to be done by the [Java virtual machine](https://en.wikipedia.org/wiki/Java_virtual_machine) (JVM).[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

The JVM loads the class files and either [interprets](https://en.wikipedia.org/wiki/Interpreter_(computing)) the [bytecode](https://en.wikipedia.org/wiki/Bytecode) or [just-in-time](https://en.wikipedia.org/wiki/Just-in-time_compilation) compiles it to [machine code](https://en.wikipedia.org/wiki/Machine_code) and then possibly optimizes it using [dynamic compilation](https://en.wikipedia.org/wiki/Dynamic_compilation).

A standard on how to interact with Java compilers programmatically was specified in [JSR](https://en.wikipedia.org/wiki/Java_Specification_Request) 199.

A Java compiler is a program that takes the text file work of a developer and [compiles](https://www.techtarget.com/whatis/definition/compiler) it into a platform-independent [Java](https://www.theserverside.com/definition/Java) file. Java compilers include the Java Programming Language Compiler (javac), the GNU Compiler for Java (GCJ), the [Eclipse](https://www.techtarget.com/searchapparchitecture/definition/Eclipse-Eclipse-Foundation) Compiler for Java (ECJ) and  [Jikes](https://www.theserverside.com/definition/Jikes).

Programmers typically write language statements in a given programming language one line at a time using a code editor or an integrated development environment (IDE). The resulting file contains what are called the source statements. The programmer then runs a compiler for the appropriate language, specifying the name of the file that contains the source statements.

At run time, the compiler first parses (analyzes) all of the language statements syntactically and then, in one or more successive stages or "passes,” builds the output code, making sure that statements that refer to other statements are referred to correctly in the final code.

Generally, Java compilers are run and pointed to a programmer’s code in a text file to produce a [class](https://www.techtarget.com/whatis/definition/class) file for use by the Java virtual machine ([JVM](https://www.theserverside.com/definition/Java-virtual-machine-JVM)) on different platforms. Jikes, for example, is an open source compiler that works in this way.

A just-in-time ([JIT](https://www.theserverside.com/definition/just-in-time-compiler-JIT)) compiler comes along with the Java VM. Its use is optional, and it is run on the platform-independent code. The JIT compiler then translates the code into the machine code for different hardware so that it is optimized for different architectures. Once the code has been (re-)compiled by the JIT compiler, it will usually run more quickly than the Java code that can only be executed one instruction at a time.

1.4 Java Interpreter

Java is a platform-independent programming language. It means that we can run Java on the platforms that have a **Java interpreter**. It is the reason that makes the Java platform-independent. The Java interpreter converts the Java bytecode (.class file) into the code understand by the operating system.

In this section, we will understand **what is an interpreter in Java, the features of the interpreter,** and **how does the Java interpreter work.** We will also see **how it is different from a compiler.**

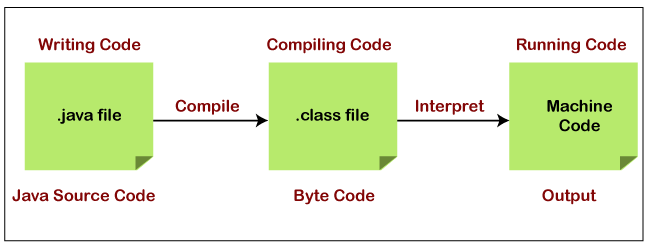
What is an interpreter in Java?

**Java interpreter** is a computer program (system software) that implements the JVM. It is responsible for reading and executing the program. It is designed in such a way that it can read the source program and translate the source code instruction by instruction. **It converts the high-level program into assembly language** (machine language).

How does the Java interpreter work?

To convert the byte code into machine code, we deploy the .class file on the [Java Virtual Machine (JVM)](https://www.javatpoint.com/jvm-java-virtual-machine)

. The JVM converts that code into machine code using the Java interpreter. The JVM uses the interpreter at runtime, after that it execute the code on the host machine.



As the Java compiler compiles the source code into the [Java bytecode](https://www.javatpoint.com/java-bytecode)

. In the same way, the Java interpreter converts or translates the bytecode into the machine-understandable format i.e. machine code, after that the machine code interacts with the operating system.

If the JVM is installed on any system it means that the platform is JVM enabled. The platform performs all the tasks of the [Java](https://www.javatpoint.com/java-tutorial)

run-time system. It loads the Java class file and interprets the compiled byte-code.

The [browsers](https://www.javatpoint.com/browsers)

, like [Google Chrome](https://www.javatpoint.com/google-chrome)

, Netscape, etc. are the popular example that contains the Java interpreter. It means these are Java-enabled browsers. It is used to run the Applet in the browser. The interpreter also serves as a specialized compiler in an implementation that supports dynamic or **just-in-time (JIT)** compilation which turns the Java bytecode into native machine instructions.

Let's see how an interpreter loads a Java program.

First, we specify the class by using the **java** command followed by the class name and options available for the interpreter, and command-line arguments if required. We use the following command to load the class:

1. % java [interpreter options] **class** name [arguments]

In the above command, the class name should be a fully qualified name (the name of the class that includes the package name, if any). Remember that, we do not write the .class extension at the end of the class name. For example:

1. java Product
2. java com.javatoint.product.Mobile

In the first command, **Product** is the class name. In the second command, **com.javatpint.product** is the name of the package in which the **Mobile** class is stored.

Once the class is loaded, Java follows a convention and searches for the class that contains the main() method. When the JVM founds the main() method, the interpreter starts the application by invoking the main() method. After executing the main() method, additional threads, and references other classes.

Features of Interpreter

It converts the source code into machine language, line by line at run time, without changing the sequence.

* An interpreter does not generate an intermediate machine code
* Each error of every line is displayed one by one
* When compared to a compiler, the program execution speed is slower
* Less amount of time is spent on analysing and processing the program
* In the following table, we have summarized the key differences between an interpreter and a compiler.
* Difference Between Interpreter and Compiler

|  |  |
| --- | --- |
| **Interpreter** | **Compiler** |
| It translates the code instruction by instruction. | It translates the entire program at once. |
| Its execution is slower. | Its execution is faster. |
| Its compile time is less. | It takes more time to compile the code. |
| It does not generate the intermediate object code. | It generates the intermediate object code. |
| It compiles the program until an error is found. | All the errors show once at the end of the compilation. |
| Python, PHP, Ruby, and Perl use an interpreter. | Java, C++, Scala, and C uses a compiler. |