## **What is 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 Oak name to Java.

# **History of Java**

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". [Java](https://www.javatpoint.com/java-tutorial) was developed by James Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s.

## Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. There are given significant points that describe the history of Java.

1) [**James Gosling**](https://www.javatpoint.com/james-gosling-father-of-java), Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.

2) Initially designed for small, [embedded systems](https://www.javatpoint.com/embedded-system-tutorial) in electronic appliances like set-top boxes.

3) Firstly, it was called "Greentalk" by James Gosling, and the file extension was .gt.

4) After that, it was called Oak and was developed as a part of the Green project.

### **Java Version History**

Many java versions have been released till now. The current stable release of Java is Java SE 10.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan 1996)
3. JDK 1.1 (19th Feb 1997)
4. J2SE 1.2 (8th Dec 1998)
5. J2SE 1.3 (8th May 2000)
6. J2SE 1.4 (6th Feb 2002)
7. J2SE 5.0 (30th Sep 2004)
8. Java SE 6 (11th Dec 2006)
9. Java SE 7 (28th July 2011)
10. Java SE 8 (18th Mar 2014)
11. Java SE 9 (21st Sep 2017)
12. Java SE 10 (20th Mar 2018)

## **Application**

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

## **Types of Java Applications**

There are mainly 4 types of applications that can be created using Java programming:

#### **1) Standalone Application**

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

#### **2) Web Application**

An application that runs on the server side and creates a dynamic page is called a web application. Currently, [Servlet](https://www.javatpoint.com/servlet-tutorial), [JSP](https://www.javatpoint.com/jsp-tutorial), [Struts](https://www.javatpoint.com/struts-2-tutorial), [Spring](https://www.javatpoint.com/spring-tutorial), [Hibernate](https://www.javatpoint.com/hibernate-tutorial), [JSF](https://www.javatpoint.com/jsf-tutorial), etc. technologies are used for creating web applications in Java.

#### **3) Enterprise Application**

An application that is distributed in nature, such as banking applications, etc. is called enterprise application. It has advantages of the high-level security, load balancing, and clustering. In Java, [EJB](https://www.javatpoint.com/ejb-tutorial) is used for creating enterprise applications.

#### **4) Mobile Application**

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications

# **Features of Java**

The primary objective of [Java programming](https://www.javatpoint.com/java-tutorial) language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as java buzzwords.

A list of most important features of Java language is given below.



1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

### **Simple**

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

1. Java syntax is based on C++ (so easier for programmers to learn it after C++).
2. Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
3. There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

### **Object-oriented**

Java is an [object-oriented](https://www.javatpoint.com/java-oops-concepts) programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. [Object](https://www.javatpoint.com/object-and-class-in-java)
2. Class
3. [Inheritance](https://www.javatpoint.com/inheritance-in-java)
4. [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
5. [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
6. [Encapsulation](https://www.javatpoint.com/encapsulation)

### **Platform Independent**



Java is platform independent because it is different from other languages like [C](https://www.javatpoint.com/c-programming-language-tutorial), [C++](https://www.javatpoint.com/cpp-tutorial), etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

* Runtime Environment
* API(Application Programming Interface)

Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).

### **Secured**

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

1. **No explicit pointer**
2. **Java Programs run inside a virtual machine sandbox**



1. **Classloader:** Classloader in Java is a part of the Java Runtime Environment(JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
2. **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access right to objects.
3. **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

### **Robust**

Robust simply means strong. Java is robust because:

1. It uses strong memory management.
2. There is a lack of pointers that avoids security problems.
3. There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
4. There are exception handling and the type checking mechanism in Java. All these points make Java robust.

### **Architecture-neutral**

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

### **Portable**

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

### **High-performance**

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

### **Distributed**

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

### **Multi-threaded**

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

### **Dynamic**

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

# **C++ vs Java**

There are many differences and similarities between the [C++ programming](https://www.javatpoint.com/cpp-tutorial) language and [Java](https://www.javatpoint.com/java-tutorial). A list of top differences between C++ and Java are given below:

|  |  |  |
| --- | --- | --- |
| **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 window, web-based, enterprise and mobile applications. |
| **Design Goal** | C++ was designed for systems and applications programming. It was an extension of [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 with a goal of being 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 [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 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 compiler and interpreter both. 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 comment. | 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 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++ creates a new inheritance tree always. | Java uses a single inheritance tree always because all classes are the child of 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 C language, 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. |

### **Note**

* Java doesn't support default arguments like C++.
* Java does not support header files like C++. Java uses the import keyword to include different classes and methods.

# **First Java Program | Hello World Example**

In this page, we will learn how to write the simple program of java. We can write a simple hello java program easily after installing the JDK.

To create a simple java program, you need to create a class that contains the main method. Let's understand the requirement first.

### **The requirement for Java Hello World Example**

|  |
| --- |
| For executing any java program, you need to1. Install the JDK if you don't have installed it, [download the JDK](http://www.oracle.com/technetwork/java/javase/downloads/index.html) and install it.
2. Set path of the jdk/bin directory. <http://www.javatpoint.com/how-to-set-path-in-java>
3. Create the java program
4. Compile and run the java program
 |

### **Creating Hello World Example**

Let's create the hello java program:

**class** Simple

{

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

   System.out.println("Hello Java");

 }

}

save this file as Simple.java

|  |  |
| --- | --- |
| To compile: | javac Simple.java |
| To execute: | java Simple |

Output:Hello Java

Compilation Flow:

When we compile Java program using javac tool, java compiler converts the source code into byte code.



## **Parameters used in First Java Program**

Let's see what is the meaning of class, public, static, void, main, String[], **System.out.println().**

* **class** keyword is used to declare a class in java.
* **public** keyword is an access modifier which represents visibility. It means it is visible to all.
* **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create an object to invoke the main method. So it saves memory.
* **void** is the return type of the method. It means it doesn't return any value.
* **main** represents the starting point of the program.
* **String[] args** is used for command line argument. We will learn it later.
* **System.out.println()** is used to print statement. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class. We will learn about the internal working of System.out.println statement later.

To write the simple program, you need to open notepad by **start menu -> All Programs -> Accessories -> notepad** and write a simple program as displayed below:



|  |
| --- |
| As displayed in the above diagram, write the simple program of java in notepad and saved it as Simple.java. To compile and run this program, you need to open the command prompt by **start menu -> All Programs -> Accessories -> command prompt**. |



|  |
| --- |
| To compile and run the above program, go to your current directory first; my current directory is c:\new. Write here: |

|  |  |
| --- | --- |
| **To compile**: | javac Simple.java |
| **To execute**: | java Simple |

## **What happens at compile time?**

At compile time, java file is compiled by Java Compiler (It does not interact with OS) and converts the java code into bytecode.



## **What happens at runtime?**

At runtime, following steps are performed:



**Classloader**: is the subsystem of JVM that is used to load class files.

**Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects.

**Interpreter**: read bytecode stream then execute the instructions.

# **Difference between JDK and JVM**

### **JVM**

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each [OS](https://www.javatpoint.com/os-tutorial) is different from each other. However, Java is platform independent. There are three notions of the JVM: specification, implementation, and instance.

The JVM performs the following main tasks:

1. Loads code
2. Verifies code
3. Executes code
4. Provides runtime environment

## **JVM Architecture**

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



### **1) Classloader**

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

1. Bootstrap ClassLoader: This is the first classloader which is the super class of Extension classloader. It loads the **rt.jar** file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.
2. Extension ClassLoader: This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside **$JAVA\_HOME/jre/lib/ext** directory.
3. System/Application ClassLoader: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

### **JDK**

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and [applets](https://www.javatpoint.com/java-applet). It physically exists. It contains JRE + development tools.

JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

1. Standard Edition Java Platform
2. Enterprise Edition Java Platform
3. Micro Edition Java Platform

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.

## **Components of JDK**

Following is a list of primary components of JDK:

|  |  |
| --- | --- |
| appletviewer: | This tool is used to run and debug Java applets without a web browser. |
| apt: | It is an annotation-processing tool. |
| extcheck: | it is a utility that detects JAR file conflicts. |
| idlj: | An IDL-to-Java compiler. This utility generates Java bindings from a given Java IDL file. |
| jabswitch: | It is a Java Access Bridge. Exposes assistive technologies on Microsoft Windows systems. |
| java: | The loader for Java applications. This tool is an interpreter and can interpret the class files generated by the javac compiler. Now a single launcher is used for both development and deployment. The old deployment launcher, jre, no longer comes with Sun JDK, and instead it has been replaced by this new java loader. |
| javac: | It specifies the Java compiler, which converts source code into Java bytecode. |
| javadoc: | The documentation generator, which automatically generates documentation from source code comments |
| jar: | The specifies the archiver, which packages related class libraries into a single JAR file. This tool also helps manage JAR files. |
| javafxpackager: | It is a tool to package and sign JavaFX applications. |
| jarsigner: | the jar signing and verification tool. |
| javah: | the C header and stub generator, used to write native methods. |
| javap: | the class file disassembler. |
| javaws: | the Java Web Start launcher for JNLP applications. |
| JConsole: | Java Monitoring and Management Console. |
| jdb: | the debugger. |
| jhat: | Java Heap Analysis Tool (experimental). |
| jinfo: | This utility gets configuration information from a running Java process or crash dump. |
| jmap: | Oracle jmap - Memory Map- This utility outputs the memory map for Java and can print shared object memory maps or heap memory details of a given process or core dump. |
| jmc: | Java Mission Control |
| jps: | Java Virtual Machine Process Status Tool lists the instrumented HotSpot Java Virtual Machines (JVMs) on the target system. |
| jrunscript: | Java command-line script shell. |
| jstack: | It is a utility that prints Java stack traces of Java threads (experimental). |
| jstat: | Java Virtual Machine statistics monitoring tool (experimental). |
| jstatd: | jstat daemon (experimental). |
| keytool: | It is a tool for manipulating the keystore. |
| pack200: | JAR compression tool. |
| Policytool: | It specifies the policy creation and management tool, which can determine policy for a Java runtime, specifying which permissions are available for code from various sources. |
| VisualVM: | It is a visual tool integrating several command-line JDK tools and lightweight [clarification needed] performance and memory profiling capabilities |
| wsimport: | It generates portable JAX-WS artifacts for invoking a web service. |
| xjc: | It is the part of the Java API for XML Binding (JAXB) API. It accepts an XML schema and generates Java classes. |