Table of Contents

[Features of Java 2](#_Toc64611496)

[**1.** **Simple** 2](#_Toc64611497)

[**2.** **Object-oriented** 3](#_Toc64611498)

[**3.** **Platform Independent** 3](#_Toc64611499)

[**4. Secured** 4](#_Toc64611500)

[**5.Robust** 5](#_Toc64611501)

[**6.Architecture-neutral** 6](#_Toc64611502)

[**7.Portable** 6](#_Toc64611503)

[**8.High-performance** 6](#_Toc64611504)

[**9.Distributed** 6](#_Toc64611505)

[**10.Multi-threaded** 7](#_Toc64611506)

[**11.Dynamic** 7](#_Toc64611507)

[**12.Interpreted** 7](#_Toc64611508)

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.



# **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:

* Java syntax is based on C++ (so easier for programmers to learn it after C++).
* Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
* There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java. (no need of destructors like C++)
* Takes less memory and less execution time

# **Object-oriented**

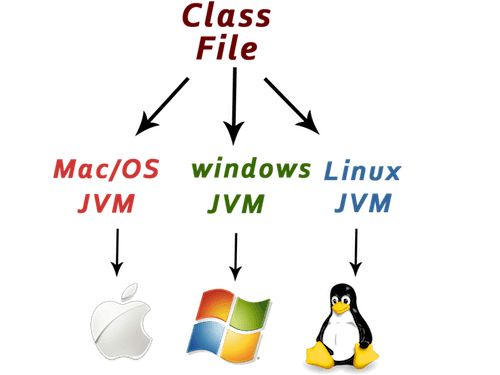
Java is an [object-oriented](https://www.javatpoint.com/java-oops-concepts) programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

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 allows to compile in one OS and execute the same in other OS.

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:

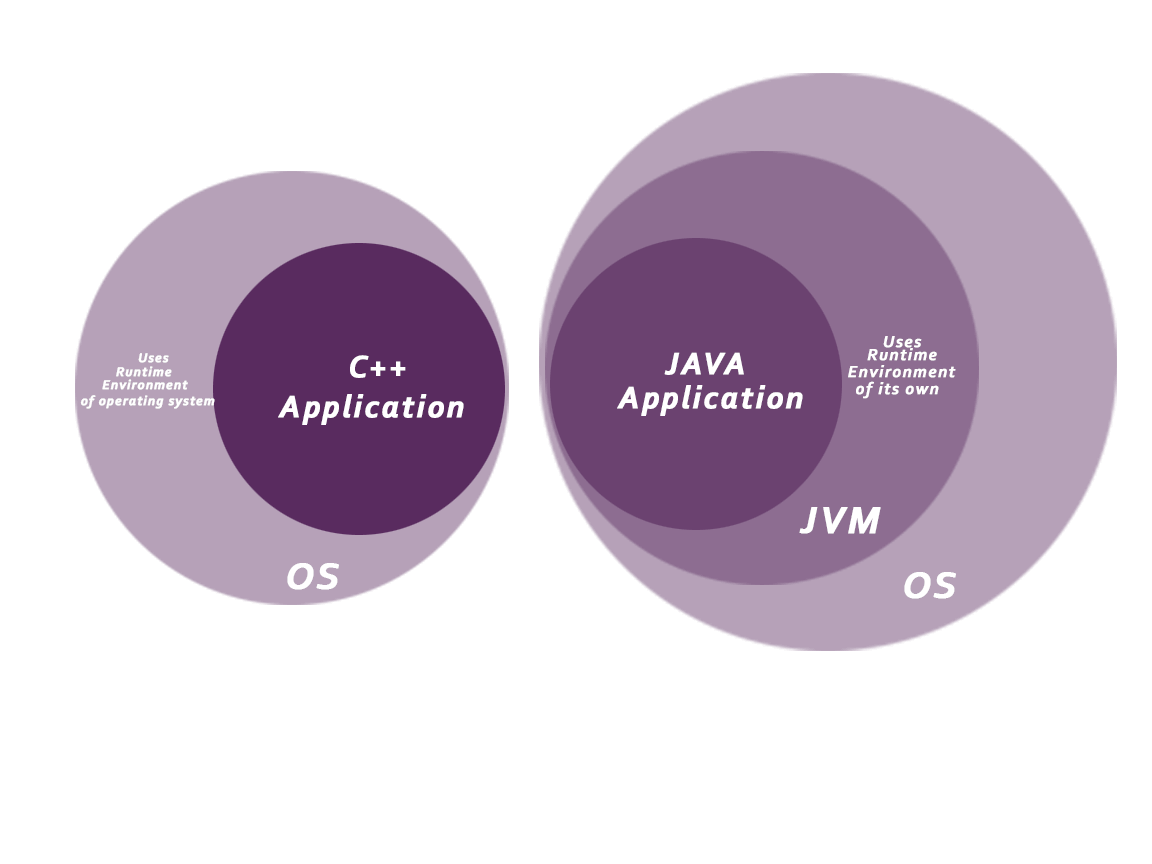
1. Runtime Environment
2. API(Application Programming Interface)

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).

# **4. Secured**

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

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



Inside JVM, we have separate security manager. It checks bytecode formation is good or not.

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

With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption

# **5.Robust**

Robust simply means strong. Java is robust because:

* It uses strong memory management. It uses heap memory management system. It’s a dynamic memory as it can allocate and deallocate memory for the objects at runtime as per application requirements.
* There is a lack of pointers that avoids security problems.
* 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.
* There are exception handling and the type checking mechanism in Java. Java library provides most of the frequently occurring exceptions.

All these points make Java robust.

# **6.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.

# **7.Portable**

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

# **8.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.

# **9.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.

# **10.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.

# **11.Dynamic**

Java is a dynamic programming language. It allows memory allocation for primitive data types at runtime (not at compile time)

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).

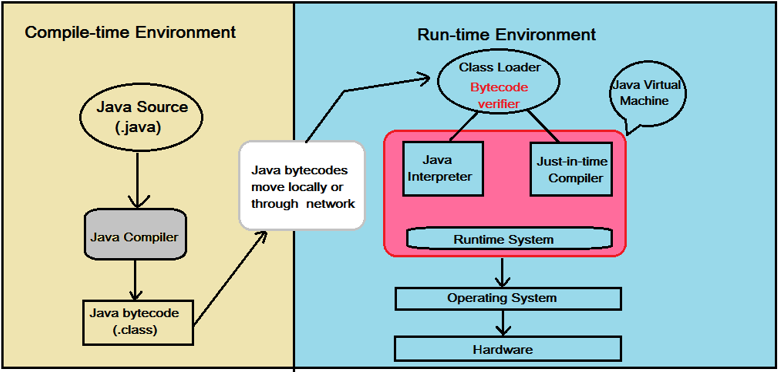
# **12.Interpreted**

Java is a compiled and interpreted programming language

Java is compiled to bytecode, which then goes into the Java VM, which interprets it

Java is compiled programming language, because to translate a program from high level representation to low level representation and to check developers mistakes in java application before execution, we need compilation.

To execute java application, we need an interpreter inside JVM, so Java is an interpreted programming language.



Code written in Java is:

* **First compiled** to bytecode by a program called *javac* as shown in the left section of the image above;
* Then, as shown in the right section of the above image, another program called *java* starts the Java runtime environment and it may **compile and/or interpret** the bytecode by using the Java Interpreter/JIT Compiler.

**When does java interpret the bytecode and when does it compile it?** The application code is initially interpreted, but the JVM monitors which sequences of bytecode are frequently executed and translates them to machine code for direct execution on the hardware. For bytecode which is executed only a few times, this saves the compilation time and reduces the initial latency; for frequently executed bytecode, JIT compilation is used to run at high speed, after an initial phase of slow interpretation. Additionally, since a program spends most time executing a minority of its code, the reduced compilation time is significant. Finally, during the initial code interpretation, execution statistics can be collected before compilation, which helps to perform better optimization.