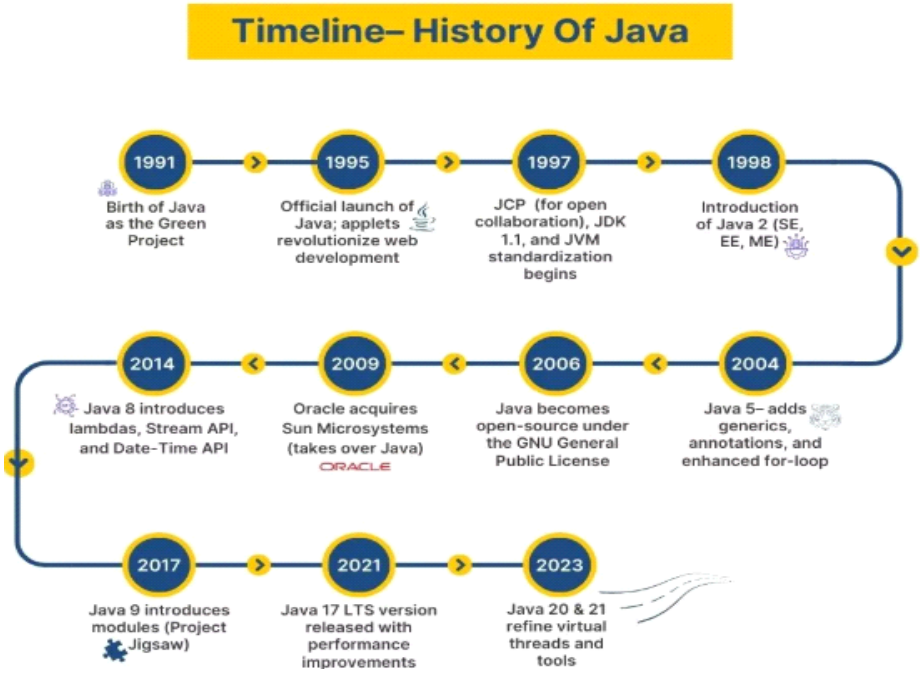
Java

05 July 2025 11:07

History of Java

- **Java is an Object-Oriented Programming Language** created by **James Gosling** in the **early 1990s**.
- It was **originally named "Oak"** after an **oak tree** outside Gosling's office.
- Later, the name was changed to **"Java"**, inspired by **Java coffee beans**, while Gosling was having coffee near his office.
- Java became popular due to its **platform independence**, known as **"Write Once, Run Anywhere"** – meaning code can run on any system with a compatible JVM.
- In **2010, Oracle Corporation acquired Sun Microsystems**, the original developer of Java.
- Oracle now **owns and manages Java**, including its official version – **Oracle Java Standard Edition (SE)**.



Features of Java :

◇ Object Oriented

Java follows the object-oriented programming (OOP) paradigm. This means everything in Java is represented as an object, which makes code modular, reusable, and easier to maintain.

◇ Simple

- **High-level language**: Java uses English-like syntax which is easy to read and write.
- **Easy to learn**: Java removes complex features like pointers and operator overloading

(found in C++), making it easier for beginners.

◇ **Robust**

- **Strong type checking**: Java checks data types strictly during compile time to reduce errors.
- **Automatic Memory Management**: Java has a garbage collector that automatically removes unused objects from memory, preventing memory leaks and crashes.

◇ **Architecture Neutral**

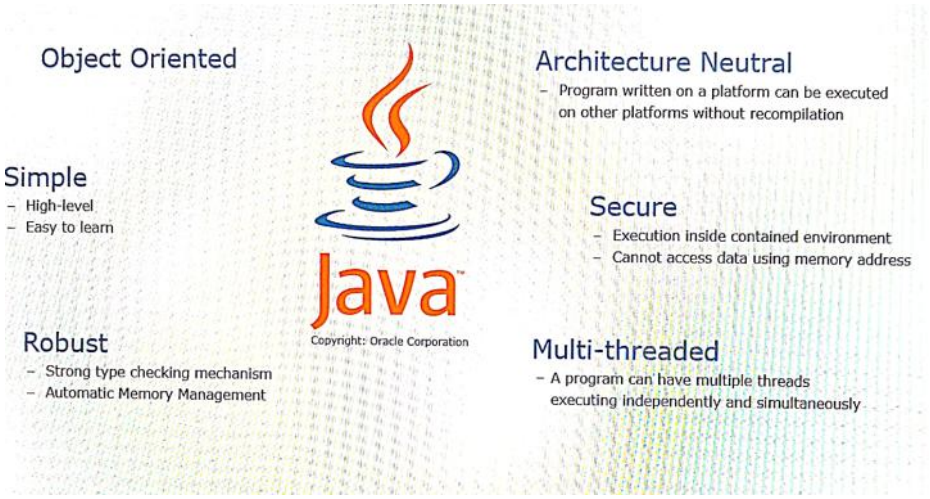
- Programs written in Java are compiled into bytecode.
- This bytecode can be executed on any platform (Windows, Linux, Mac, etc.) without recompilation, as long as the JVM is installed.

◇ **Secure**

- **Runs in a controlled environment**: Java programs run inside the Java Runtime Environment (JRE) which isolates them from system resources.
- **No direct memory access**: Java doesn't allow accessing memory using pointers, making it hard for malicious code to harm the system.

◇ **Multi-threaded**

- Java supports multithreading, which means you can create programs that perform multiple tasks at the same time.
- For example, a game can play background music while responding to user inputs simultaneously.



Industry Usage of Java :

1. Web Application Development

- Java is widely used to build dynamic, secure, and scalable web applications.
- Technologies used: **Servlets**, **JSP**, **Spring Framework**, **Hibernate**, **Spring Boot**.
- Examples: E-commerce sites, enterprise portals, CRMs.

2. Enterprise Applications

- Java is the **backbone of enterprise software**, especially with Java EE (Jakarta EE) platforms.
- Used in banks, insurance, telecoms for large-scale backend systems.
- Features like **multi-threading**, **security**, and **distributed computing** make it ideal.

3. Android App Development

- Java was the official language for Android development before Kotlin.
- Still widely used to create native Android apps using **Android Studio**.

4. Financial Services

- Used in banking and investment sectors for:
  - Transaction management
  - Online banking systems
  - Risk and portfolio analysis

5. Big Data and Analytics

- Java works with big data technologies like:
  - **Apache Hadoop**
  - **Apache Spark**
- Used for processing large datasets and analytics.

6. Embedded Systems

- Java runs well on small devices due to **Java ME (Micro Edition)**.
- Used in TVs, set-top boxes, smartcards, etc.

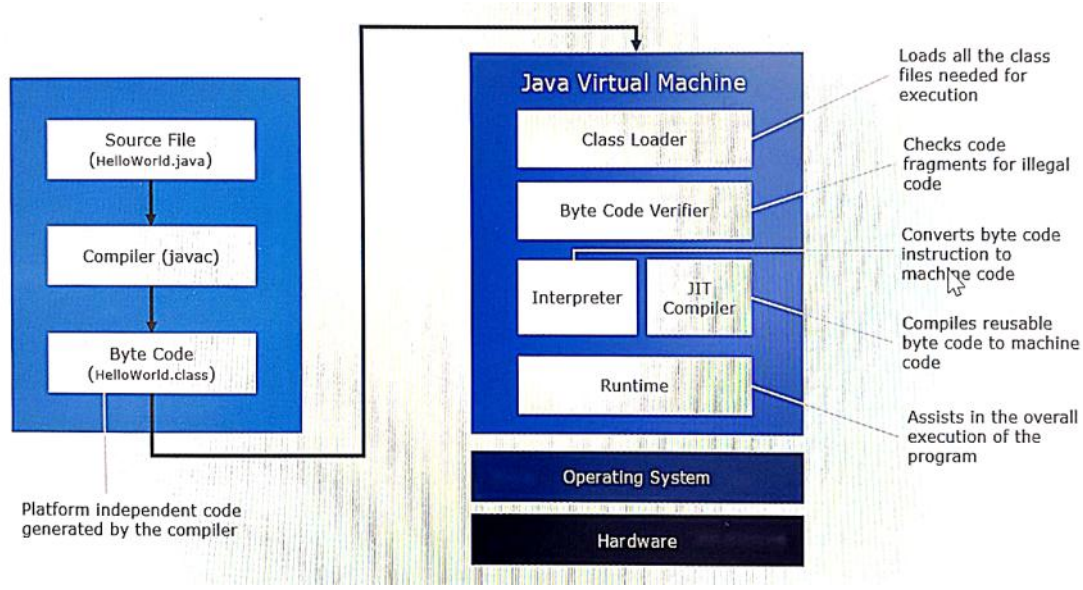
Compiling and Running of simple java program

Compilation of C-program

- The compiler in C program **translates the entire source code into machine code** also called **binary or executable code(.exe)**. that is specific to the operating system and hardware.

- This means C programs are **platform-dependent** – the compiled code won't run on a different system unless recompiled for that system.

Compilation of java program

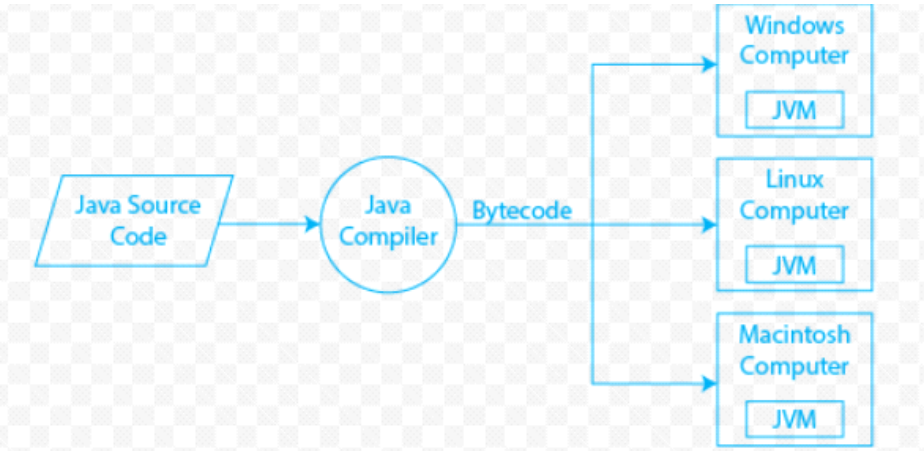


- Instead of directly converting to machine code, the **Java compiler (javac) converts it into an intermediate form called bytecode** (stored in a .class file).
- This bytecode is **not tied to any specific machine or OS**.
- Then, the **Java Virtual Machine (JVM)** on the target system interprets or compiles this bytecode into **machine code at runtime** using JIT (Just-In-Time compiler).
- This process makes Java **platform-independent**, meaning the same .class file can run on any device that has a JVM.

Bytecode, JVM ,JRE ,JDK

Bytecode :

- **Bytecode** is an intermediate, platform-independent code generated by the **Java compiler (javac)** after compiling a .java file.
- It is **not machine code**, but also **not source code**.
- Bytecode can **run on any platform** that has a compatible JVM, which makes Java **platform-independent**.



JVM(Java Virtual Machine) :

- The **Java Virtual Machine (JVM)** is a part of the Java Runtime Environment (JRE) that enables Java programs to run **platform-independently**.

- It acts like a **virtual computer** that reads and executes Java **bytecode**, no matter what underlying hardware or operating system you're using.

Components in JVM :

1. Class Loader
- **Role**: Loads .class files (bytecode) into memory.
- **Function**: It reads the class file and brings it into the runtime environment.

2. Bytecode Verifier
- **Role**: Checks the bytecode for **security and correctness**.
- **Function**: Verifies that no illegal code or access is present (e.g., no stack overflow, no memory violations).

3. Interpreter
- **Role**: Reads bytecode instructions **line by line** and converts them to machine code for immediate execution.
- **Function**: Useful for quick execution, but slower performance.

4. JIT (Just-In-Time) Compiler
- **Role**: Compiles **repeated bytecode** into **machine code** for faster performance.
- **Function**: Improves efficiency by compiling chunks of bytecode only when needed.

5. Runtime
- **Role**: Manages the entire execution of the program.

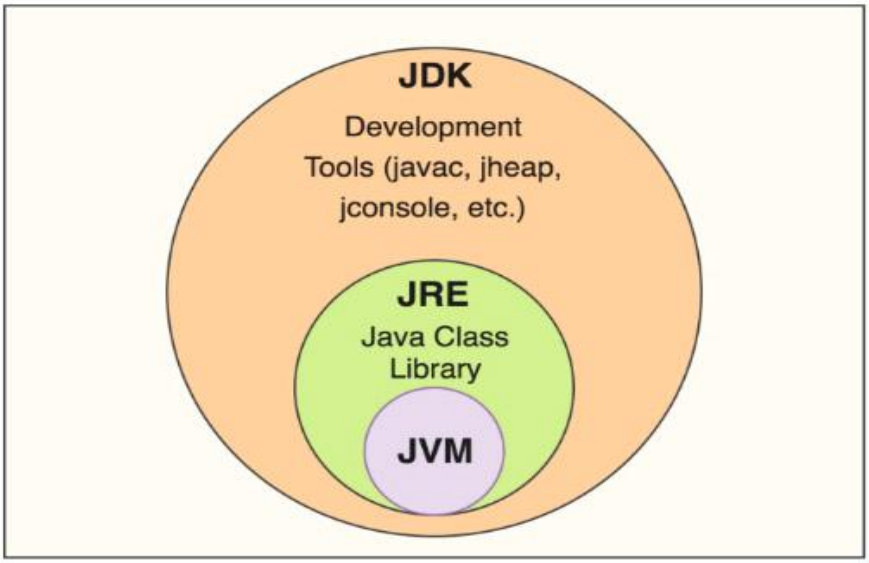
- **Function**: Provides the environment and services the program needs to run (e.g., memory allocation, garbage collection).

JRE(Java Runtime Environment) :

- The **Java Runtime Environment (JRE)** is a software package that provides everything needed to **run** Java applications — but **not to develop them**.
- **JRE does NOT contain the Java compiler (javac)**.
- You cannot **write and compile** Java programs with just JRE — only **run** them.

JDK(Java Development Kit) :

- The **Java Development Kit (JDK)** is a **complete software development package** that allows you to **write, compile, debug, and run** Java programs.
- It is the **core toolset for Java developers** and includes everything you need to build Java applications.
- JDK = JRE + Development Tools



Simple java program

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

public class HelloWorld

- class: Keyword used to define a **class** in Java.
- HelloWorld: The **name of the class**. It must match the filename (HelloWorld.java).
- public: Access modifier that allows the class to be accessed from anywhere.

public static void main(String[] args)
This is the **main method**, where the program **starts execution**.

- public: JVM can access this method from anywhere.
- static: No need to create an object to call this method.
- void: It **returns nothing**.
- main: Entry point of the Java program.
- String[] args: This is used to **accept command-line arguments**.

System.out.println("Hello, World!");
This line prints the message on the console.

- System: A built-in Java class from the **java.lang package**.
- out: A static object of **PrintStream**, used to display output.
- println(): A method to print a line of text, followed by a newline.