JVM, JDK & JRE and JSE, JEE &JME Explained

made by vijay singh

1.What is JVM?

JVM = Java Virtual Machine

JVM is a **software that runs Java programs**.
It acts like a **fake computer inside your real computer**.

2.Why JVM Exists (Big Problem It Solves)

Different computers are different:

- Windows
- Linux
- Mac
- Intel CPU
- ARM CPU

Machine code for each is different.

Without JVM:

You must compile Java separately for every OS and processor.

With JVM:

Write Java once → Run anywhere.

3.Java Program Execution Flow (Full Internal Steps)

Step 1: You Write Java Code

```
class Hello {

  public static void main(String[] args) {

    System.out.println("Hello");

  }

}
```

This is **human-readable code**.

Step 2: Java Compiler Converts to Bytecode

Command:

javac Hello.java

Output:

Hello.class

This file contains **Bytecode**, NOT machine code.

Bytecode = universal language for JVM

Same on Windows, Linux, Mac.

Step 3: JVM Starts

When you run:

java Hello

JVM starts and loads Hello.class

4.Inside JVM (Main Internal Parts)

1.Class Loader (Loads Code)

Class Loader loads .class files into memory.

It loads:

- Your class
- Java libraries (System, String, etc.)

Types:

1. Bootstrap Class Loader (loads core Java classes)
2. Extension Class Loader
3. Application Class Loader (your code)

2.Bytecode Verifier (Security Guard)

Before running, JVM checks:

- No illegal memory access

- No fake bytecode
- Stack not overflow
- Code is safe

This is why Java is **secure**.

3.Runtime Data Areas (Memory Inside JVM)

JVM divides memory like this:

Heap

- Stores objects and arrays
- Garbage Collector works here

Stack

- Stores function calls
- Local variables
- Each thread has its own stack

Method Area

- Stores class info, bytecode, static variables

PC Register

- Stores next instruction address

Native Method Stack

- For C/C++ code called from Java (JNI)

Step 4: Execution Engine (Real Brain)

Execution Engine runs bytecode.

Interpreter

- Reads bytecode line by line
- Converts to machine code
- Slow but starts fast

JIT Compiler (Just-In-Time)

When code runs many times:

JVM converts bytecode to **native machine code**
Stores it in cache

So next time:

- Runs super fast
- No conversion again

This is why Java becomes fast after warm-up.

Step 5: Garbage Collector (Auto Memory Cleaner)

In C/C++ you manually free memory.
In Java:

Garbage Collector automatically deletes unused objects.

Example:

new Student(); // no reference later

GC removes it from Heap.

Step 6: Native Interface (JNI)

If Java needs OS features:

Calls C/C++ libraries using JNI.

Example:

- File system
- OS calls
- Graphics
- Network stack

Full Internal Flow Diagram

Java Code (.java)

↓

Java Compiler (javac)

↓

Bytecode (.class)

↓

JVM

├── Class Loader

├── Bytecode Verifier

├── Memory Areas

├── Execution Engine

│ ├ Interpreter

│ └ JIT Compiler

└── Garbage Collector

↓

Machine Code → CPU

Why Java is Platform Independent?

Because:

Java does NOT convert to Windows/Linux machine code
It converts to Bytecode
JVM converts Bytecode to machine code

So:

Same Bytecode + Different JVMs = Run Everywhere

Real-Life Example (Girlfriend WhatsApp Style)

Think:

- You speak **Hindi**
- Girlfriend speaks **English**

Bytecode = Common Translator Language

JVM = Translator for each person

You speak once → Translator converts for everyone.

JVM vs JRE vs JDK (Confusing Part)

JVM

- Only runs Java programs

JRE (Java Runtime Environment)

- JVM + Libraries

JDK (Java Development Kit)

- JRE + Compiler + Tools (for coding)

Why JVM is Powerful?

Platform independent
Secure
Automatic memory management
Optimized with JIT
Multithreading support

Why C++ is Faster Sometimes?

Because:

- C++ → Direct Machine Code
- Java → Bytecode → JVM → Machine Code

Extra layer = little overhead (but JIT reduces it).

Is Java a Compiler or Interpreter?

Java is BOTH: Compiler + Interpreter

1.Java is a Compiler

When you write:

Hello.java

You run:

javac Hello.java

This **compiles** Java code into **Bytecode**

Output:

Hello.class

So:

Java source code → Bytecode

This is compilation.

2.Java is also an Interpreter (Inside JVM)

When you run:

java Hello

JVM does:

- Reads bytecode line by line
- Converts it to machine code
- Executes it

This is interpretation.

3.Java is ALSO JIT Compiler (Extra)

Modern JVM uses:

JIT (Just-In-Time Compiler)

- Converts frequently used bytecode into **native machine code**
- Stores it in cache
- Runs fast next time

So Java becomes **very fast after warm-up**.

Java Code → Compiler → Bytecode

Bytecode → Interpreter/JIT → Machine Code → CPU

What is JRE?

JRE = Java Runtime Environment

Meaning:
JRE is the software needed to **RUN Java programs**.

If you only want to **run Java programs (not write them)**, you need JRE.

What is Inside JRE?

JRE is made of:

1.JVM (Java Virtual Machine)

- Runs bytecode
- Converts bytecode → machine code
- Manages memory, threads, garbage collection

2.Java Libraries (API)

Pre-written classes like:

- System
- String
- Math
- ArrayList
- File handling

These help Java programs work.

3.Supporting Files

- Configuration files
- Native OS libraries
- Tools for runtime

Relationship Between JVM, JRE, JDK

JVM

Only runs bytecode.

JRE

JVM + Libraries (to run Java programs)

JDK

JRE + Compiler + Tools (to DEVELOP Java programs)

Easy Diagram

JDK

└── JRE

└── JVM

Real-Life Example

JVM = Engine

JRE = Engine + Fuel + Parts (to drive)

JDK = Workshop tools to build the car

When You Need JRE

You need JRE when:

- You downloaded a Java software
- You want to run Minecraft (old versions)
- You want to run a Java application

No coding needed.

When You Need JDK

You need JDK when:

- You want to write Java code
- You use javac compiler
- You are a developer

JRE provides the runtime environment required to execute Java applications. It includes the JVM and core libraries.

- **JVM = Runs bytecode**
- **JRE = JVM + Libraries (Run Java apps)**
- **JDK = JRE + Compiler (Develop Java apps)**

What is JDK?

JDK = Java Development Kit
JDK is the software needed to **WRITE, COMPILE, DEBUG, and RUN Java programs**.

If you want to **be a Java developer**, you must install **JDK**.

What is Inside JDK?

JDK contains **everything** needed for Java development.

1.JRE (Java Runtime Environment)

To run Java programs.

2.JVM (Java Virtual Machine)

To execute bytecode.

3.Java Compiler (javac)

Converts:

Hello.java → Hello.class

4.Development Tools

Some important tools:

Tool	Work
javac	Compiler
java	Run program
jdb	Debugger
javadoc	Create documentation
jar	Package files
jshell	Interactive Java shell

Relationship (Very Important)

JDK

└── JRE

└── JVM

Real-Life Example

JVM = Car engine

JRE = Car (engine + fuel + parts)

JDK = Factory tools to build the car

When You Need JDK

You need JDK if:

- You write Java code
- You are learning Java
- You want to compile Java programs
- You use IDE like VS Code, IntelliJ, Eclipse

JDK is for developers, JRE is for users, JVM is for running bytecode.

JDK is a software development kit that provides tools, compiler, JRE, and libraries to develop and run Java applications.

- **JVM → Executes bytecode**
- **JRE → Runs Java apps**
- **JDK → Develop + Run Java apps**

Java Has 3 Main Editions

Java is divided into **three editions** (like three types of Java for different purposes):

1. **Java SE**
2. **Java EE (now Jakarta EE)**
3. **Java ME**

1.Java SE (Java Standard Edition)

Java SE = Core Java

This is the **basic Java** used for:

- Core programming
- Data Structures
- Desktop applications
- Learning Java

What Java SE Includes

- JVM
- JRE
- JDK
- Core libraries
- OOP concepts
- Collections
- Multithreading
- File handling

Where Java SE is Used

- Desktop apps
- Command line programs
- DSA coding
- Core backend logic

Example

```
class Test {

  public static void main(String[] args) {

    System.out.println("Hello");

  }

}
```

This is Java SE.

2.Java EE (Java Enterprise Edition)

Java EE = Big Company Backend Java

Now called:
Jakarta EE

Java EE is used to **build large web applications and enterprise systems**.

What Java EE Includes

Built on top of Java SE + extra tools:

- Servlets
- JSP
- EJB (Enterprise Java Beans)
- JPA (Database ORM)

- REST APIs
- Web servers
- Security
- Distributed systems

Where Java EE is Used

- Banking systems
- E-commerce websites
- Government portals
- Big company backend servers

Real Example

- Amazon backend
- Banking apps
- Enterprise ERP systems

Mostly replaced by **Spring Boot today**, but concept is same.

3.Java ME (Java Micro Edition)

Java ME = Small Devices Java

Java ME is for **small devices with low memory and power**.

Used In

- Old mobile phones
- Smart cards
- IoT devices
- Embedded systems
- TVs
- SIM cards

Example

Old Nokia phones running Java games (Snake, Jar apps).

Edition	Full Name	Used For
Java SE	Standard Edition	Core Java, desktop, learning
Java EE	Enterprise Edition	Big web apps, backend servers
Java ME	Micro Edition	Small devices, IoT

Simple Real-Life Example

Think Java like vehicles:

Java SE = Car

Java EE = Bus (big system)

Java ME = Bike (small device)

JSE:

Java Standard Edition is the core Java platform providing basic libraries and JVM.

JEE:

Java Enterprise Edition is used for enterprise-level web and distributed applications.

JME:

Java Micro Edition is designed for embedded and resource-constrained devices.

Important Modern Industry Truth

- Java EE is now **Jakarta EE**
- Spring Boot is more popular than Java EE
- Java ME is rarely used today
- Java SE is mandatory for everyone

SE = Core Java

EE = Web & Enterprise Java

ME = Small Device Java