How Java Actually Runs

A .java file becomes OS-neutral bytecode; the JVM runs it on your laptop and on Jenkins the same way -- which is why JDK, JRE, and JVM show up in every automation interview.

Automation Foundations · ashishautomationlabs.store

Before you install anything, know what you are installing. Most tutorials skip this and jump to "download the JDK." That is why thousands of automation engineers can write Java and then go quiet when an interviewer asks: "So, what is the JVM?"

Five minutes here means you will never be that person -- and it explains the suite mystery you already hate: passes on your laptop, fails on Jenkins.

The problem Java was built to solve

Your laptop may run Windows. The Jenkins machine that will run your future regression suite almost certainly runs Linux. A colleague writes tests on a Mac. Those operating systems understand different low-level machine languages. A program built for Windows does not simply run on Linux.

Your test suite must give the same results everywhere. A test that passes locally and fails in CI is the famous "works on my machine" problem -- a serious risk for automation, not a joke.

Java's solution is a middle step. Your instructions go through two translations instead of one.

Two steps: compile, then execute

Step 1 -- You compile. A tool called the compiler (`javac`) reads your `.java` file -- human-readable source -- and translates it into a middle language called **bytecode**, stored in a `.class` file. Bytecode is not for Windows, Linux, or Mac. It is for one standard, imaginary machine.

Step 2 -- The JVM executes. That imaginary machine is made real by the **Java Virtual Machine (JVM)**. There is a JVM for Windows, one for Linux, one for Mac -- and every one of them reads the same bytecode and runs it the same way.

Your `.class` file is like a test case written in standard English: you write it once, and any trained executor anywhere can run it. That is Java's promise -- _write once, run anywhere_ -- and for you it is not marketing. It gives your Java code a common execution model across operating systems -- while your automation environment can still introduce its own differences.

> *Predict from memory: is a* `.class` *file built for one specific OS? Does the same bytecode run on the Windows JVM and the Linux JVM? Does* `javac` *execute bytecode?*

>

> *Answers: no; yes; no --* `javac` *compiles, the JVM executes.*

The compiler is a defect reporter

Before the compiler produces bytecode, it checks every line. If anything is unclear or badly formed, it refuses to build and gives you a report: which file, which line, what is wrong.

Beginners feel that as rejection. Hear this early: **a compiler error is a defect report raised against your code, and you are the developer it is assigned to.** You have spent years writing defect reports and wishing developers would take them well. Now you are on the other side -- with one improvement. This reporter is never wrong about the line number, and it never marks anything "cannot reproduce."

You will cause two of those reports on purpose in a later lesson, and read them like the defect reports they are.

JDK, JVM, JRE -- the interview question

Three short names, endlessly confused, asked again and again. They are nested, not rivals:

- **JVM (Java Virtual Machine)** -- the executor. It reads bytecode and runs it. It does

not help you write Java; it only runs the finished product.

- **JDK (Java Development Kit)** -- the full toolbox. It contains the JVM, the compiler

`javac`, and other development tools. When you install "Java" to _write_ Java, you install a JDK.

- **JRE (Java Runtime Environment)** -- an old package that contained only enough to _run_

Java programs, not write them. Since Java 11 there is no separate JRE download -- you install the JDK. The word still appears in interviews and old blog posts; the product itself is history.

One-line answer for the room: the JVM runs bytecode; the JDK is the JVM plus the tools that create bytecode; the JRE was the run-only package, and it stopped shipping separately after Java 11.

> *Q: If the JRE is gone, why do interviewers still ask about it?*

>

> *Because millions of machines and posts still use the word. Knowing it marks you as someone who understands the history, not just the current installer.*

Which Java version? A thirty-second decision

Java releases every six months. Every two years or so, one version is marked **LTS** (Long-Term Support). Companies standardize on those.

As of 2026, the LTS versions you will meet are 8, 11, 17, 21, and 25. **This course uses Java 17 (LTS).** Java 11+ is common in Selenium ecosystems; companies may standardize on 11, 17, or newer LTS releases.

Selenium needs Java 11 as a minimum as of 2026; most company frameworks sit on 11, 17, or 21. Nobody will ask why you are not on last week's release. Worrying about versions is a trap that keeps beginners shopping instead of coding. You have now escaped it.

You now know what happens when you run a `.java` file, and you can answer the opener that starts most automation interviews. The next lesson is the practical half: install the JDK, install an IDE, and create the project you will type your first check into.