

## JAVA:





it is an **high-level**(human readable), **object-oriented** (code is based on objects and classes...#encapsulation,#inheritance),**portable**(it can be moved from one os to another without any changes #no os specific dependencies), **platform-independent** (using jvm it run on any os without os specific dependencies)

## JDK:(JAVA DEVELOPMENT KIT):

It is an **complete software development kit** which is used for developing the java applications .....when we install the JDK on our local machines it is used to **write, compile, run, debug**.

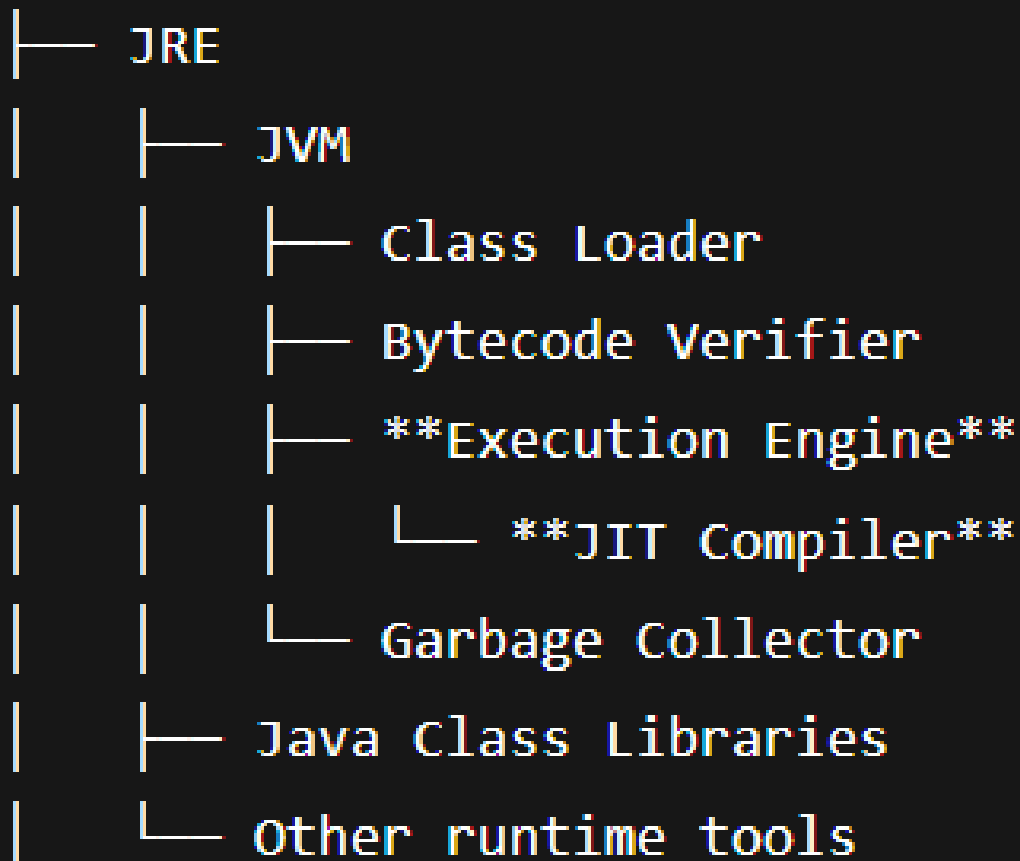
why it is important to install the jdk kit?

In software companies, **JDK is the foundation for Java application development**. It provides:

1.  Tools to **compile** Java code (e.g., **javac**)
2.  Tools to **run** Java programs (e.g., **java**)
3.  **Libraries (Java API)** used in enterprise apps
4.  Utilities for packaging (jar), documenting (javadoc), and debugging (jdb)

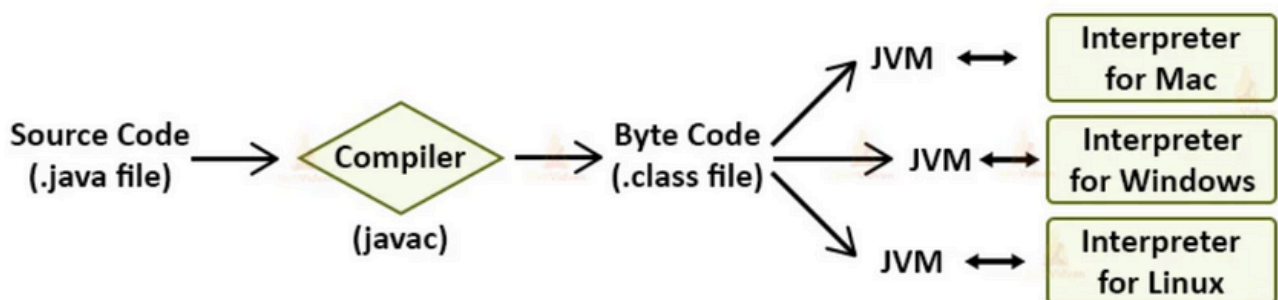
Without JDK, development tools like **Eclipse**, **IntelliJ IDEA**, or **Visual Studio Code** cannot compile Java code.

## JDK

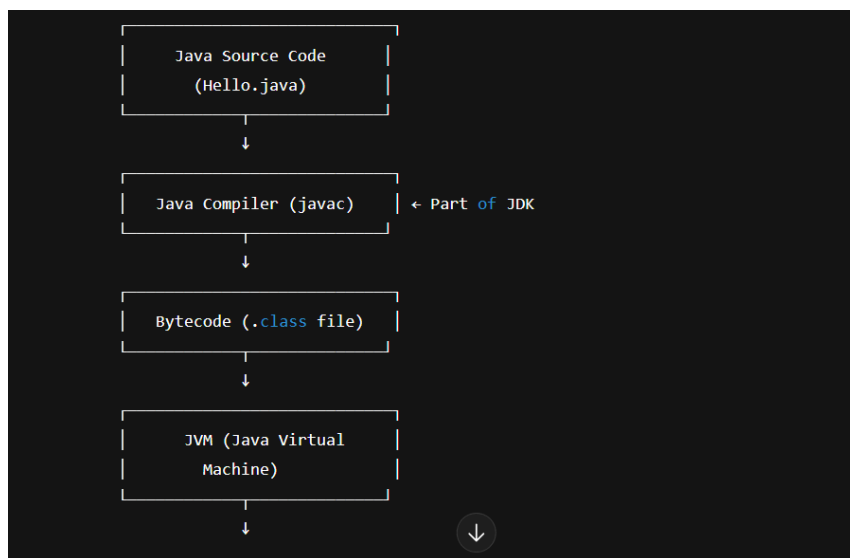


### JVM (JAVA VIRTUAL MACHINE):

It is necessary to compile the code or interpret the code but in java it uses both concept....



JVM IS THE **ENGINE** THAT RUNS THE JAVA PROGRAM. It takes the **compiled javacode** and then convert into **byte code** by **interpreting** it line by line and then converted it into **machine** code related to their system os.



JVM Internal Parts
1. <b>**Class Loader Subsystem**</b> <ul style="list-style-type: none"><li>- Loads .class files into JVM</li><li>- Handles loading, linking, and initialization</li></ul>
2. <b>**Bytecode Verifier**</b> <ul style="list-style-type: none"><li>- Checks code for security &amp; correctness</li></ul>
3. <b>**Runtime Data Areas**</b> <ul style="list-style-type: none"><li>- Heap (Objects), Stack (Methods), etc.</li></ul>
4. <b>**Execution Engine**</b> <ul style="list-style-type: none"><li>- Interpreter (line-by-line)</li><li>- JIT Compiler (compiles hot code for speed)</li></ul>
5. <b>**Native Interface &amp; Libraries**</b> <ul style="list-style-type: none"><li>- Connects with OS/Hardware</li></ul>

## JIT [JUST IN TIME]:

✓ Simple Explanation:

When you run a Java program:

1. Java code ➡ is compiled into **Bytecode** (.class file).
2. JVM reads this bytecode and **interprets** it **line by line** (slow).
3. 🔥 **JIT comes in** and says:
4. “Hey! This code is running again and again. Let me compile it into **machine code** (native CPU code) so the system can run it **faster next time!**”
5. It **boosts** the performance

6. It **avoids the repetition**

## ✓ Why So Many Components in JVM? Why is it Complex?

You're asking an *excellent* software engineering question.

### ◆ Reason 1: Platform Independence (Write Once, Run Anywhere)

- Other languages like C/C++ compile to **machine code** for one OS.
- Java compiles to **bytecode**, which is **OS-independent**.
- JVM makes this work by translating bytecode into machine code **during runtime**, which requires more steps — but it adds **massive flexibility**.

### ◆ Reason 2: Built-in Security

- JVM verifies all bytecode before execution.
- Prevents viruses, memory corruption, or illegal access.
- That's why Java is trusted in **banking, enterprise, Android, and government-level applications**.

Compare this with C: if you misuse memory (like accessing an array out of bounds), it crashes the system. Java catches this at the bytecode verification level.

### ◆ Reason 3: Automatic Memory Management

- Languages like C/C++ require developers to manually allocate and free memory.
- Java handles memory **automatically** using **Garbage Collection**.
- The JVM has **multiple types of collectors** like:
  - G1 GC (low-latency)
  - ZGC (scalable GC for large heaps)
  - Parallel GC (throughput focused)

That's why it has **Runtime Data Areas** and **GC threads** — adding complexity but making life easier for developers.

### ◆ Reason 4: Performance Optimization (JIT Compiler)

- JVM is **dynamic** — it watches which parts of the code run often, then **compiles only that part to native code** (called "hotspot compilation").
- This **makes Java perform nearly as fast as C/C++**, especially in long-running systems (like web servers).

Other interpreted languages like Python or Ruby don't do this. That's why Java performs better in enterprise environments.

### ◆ Reason 5: Multithreading and Concurrency Support

- JVM supports **true multithreading** at the language and VM level.
- It manages **thread scheduling, synchronization, and thread-local memory**.

- This makes Java suitable for building **servers, chat apps, game engines**, etc.

## **JAVA RUNTIME ENVIRONMENT (JRE) :**

It provides the environment to run the java program.....but not to develop them.....

It includes the **JVM** and **standard libraries**, but it does not contain development tools like the compiler. It's mainly used by end-users who want to execute Java programs.