JDK (Java Development Kit)

- The JDK is a **software development environment** used to develop Java applications and applets.
- It physically exists and includes the JRE (Java Runtime Environment) along with development tools.
- It is an implementation of one of the Java platforms:
 - Java SE (Standard Edition)
 - Java EE (Enterprise Edition)
 - Java ME (Micro Edition)
 - JavaFX

Components of JDK:

- JVM + JDK = Other resources, which include:
 - Interpreter/Loader (java)
 - o Compiler (javac)
 - Archiver (jar)
 - Document generator (Javadoc), etc.

JRE (Java Runtime Environment)

- JRE is a **set of software tools** used for developing Java applications.
- Provides the **runtime environment** for Java programs.
- Physically exists → it is the implementation of JVM.
- Consists of:
 - Libraries
 - o **Other files** used by the JVM at runtime.

JVM (Java Virtual Machine)

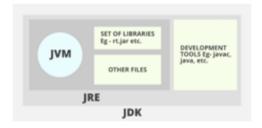
- JVM is an abstract machine → it doesn't physically exist.
- Provides a runtime environment for executing Java bytecode.
- Available for many hardware & software platforms.
- JDK, JRE, and JVM are platform-dependent because of differences in configuration.

3 Notions of JVM

- 1. Specification
- 2. Implementation
- 3. Instance

Main Tasks of JVM

- Loads code
- Executes code
- Verifies code
- · Provides runtime environment



Java File Processing

1. Compilation vs Execution

- Compilation → .java file → .class file
- Execution → .class file runs inside the JVM (which is custom-built for every OS).
- Java achieves platform independence because .class (bytecode) runs on JVM, not directly on OS.

2. Compilation Process (7 Steps)

Input: .java file → Output: .class file

i) Parse

- Java compiler reads the .java file.
- Breaks code into tokens and builds an Abstract Syntax Tree (AST) to represent the code.

ii) Enter

- Compiler records information about classes, methods, and variables in a symbol table.
- Works like a map for tracking definitions.

iii) Process Annotations

• If annotations exist (e.g., @Override, @Deprecated), the compiler processes them when needed.

iv) Attribute

- · Compiler checks and verifies:
 - Types
 - Naming conventions
 - o Other semantic rules

v) Flow

- Performs data flow analysis:
 - o Ensures variables are assigned before use.
 - Checks which parts of the code can actually be executed.

vi) Desugar

• Converts advanced syntax into simpler Java code.

Examples:

- o for-each loops → converted into simple for loops.
- Lambdas → converted into standard method calls.

vii) Generate

Generates .class file (bytecode) → which JVM can execute.

Execution in Java

- The .class files generated are **independent of machine and OS**, allowing them to run on any system.
- Execution process:
 - 1. The main class file (with main method) is passed to JVM.
 - 2. It goes through:
 - ClassLoader
 - Bytecode Verifier
 - JIT (Just-In-Time) Compiler

ClassLoader

- The main class is loaded into memory.
- All other classes referenced through the main class are loaded via the ClassLoader.
- The **ClassLoader** itself is an object that creates a flat namespace.

LoadClass Function Prototype

Class r = loadClass(String className, boolean resolveIt);

- className → Name of the class to be loaded.
- resolveIt → Flag to decide whether the referenced class should be loaded or not.

Types of ClassLoader

1. Primordial ClassLoader

- Also called Bootstrap ClassLoader.
- Loads core Java classes from the JDK (e.g., java.lang.String, java.util.List).
- It is embedded into all JVMs and acts as the default ClassLoader.
- Example: java.lang.Object

Types of ClassLoader

1. Non-Primordial (User-defined) ClassLoader

- A user-defined ClassLoader that can be coded to customize the class loading process.
- It is defined by the programmer and preferred over the default loader when custom behavior is required.

Bytecode Verifier

- Inspects the loaded class to ensure instructions do not perform harmful operations.
- Performs several checks:
 - Variables are initialized before use.
 - Method calls match the type of object references.
 - Rules for accessing private data & methods are followed.
 - Local variable access stays within the runtime limits.
 - o The runtime stack does not overflow.
 - o If any check fails → class is **not allowed to be loaded**.

Just-In-Time (JIT) Compiler

- Converts loaded bytecode into machine code.
- Bytecode is understandable by JVM, but not directly executable by the processor → JIT converts it into native machine code.
- Native code can then run directly on the computer's processor.
- Ensures faster program execution because conversion happens while the program is running.
- Triggered **only during execution**, not beforehand.