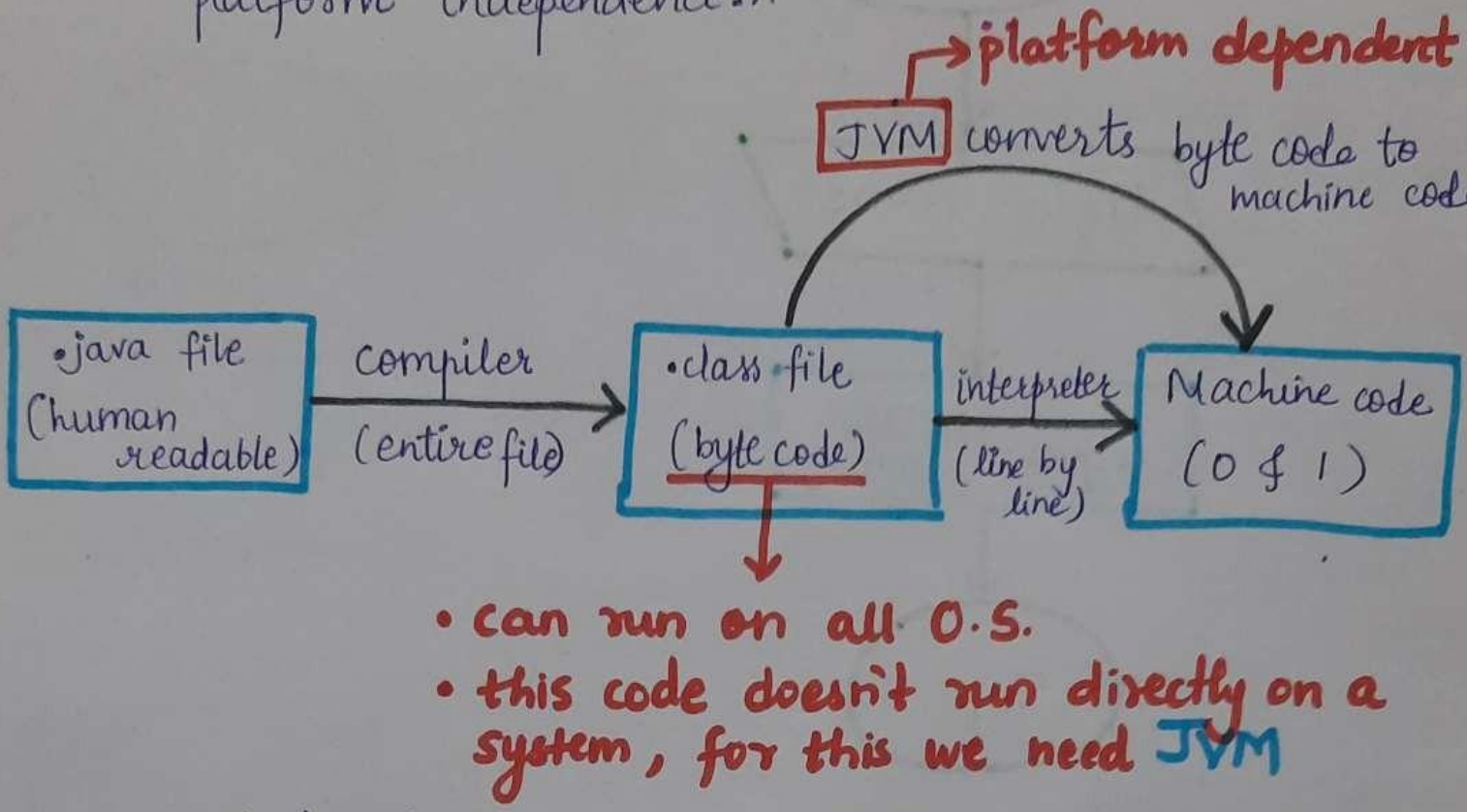
Introduction to Java ♥

★ How Java code executes and more information about platform Independence...



★ Therefore, Java is platform independent ★

⇒ We can provide this byte code to any system means we can compile the java code on any system.

⇒ But JVM is platform dependent means for every O.S. the executable file that we get, it has step by step set of instruction dependent on platform.

✯ JDK vs JRE vs JVM vs JIT

JDK [Java Development Kit]

↳ provides environment to develop & run Java program

JRE [Java Runtime Environment]

↳ provides environment to only run the program

- → deployment technologies
- → user interface toolkit
- → integration libraries
- → base libraries

etc.

JVM [Java Virtual Machine]

JIT ~~[Java]~~ [Just-in-time]

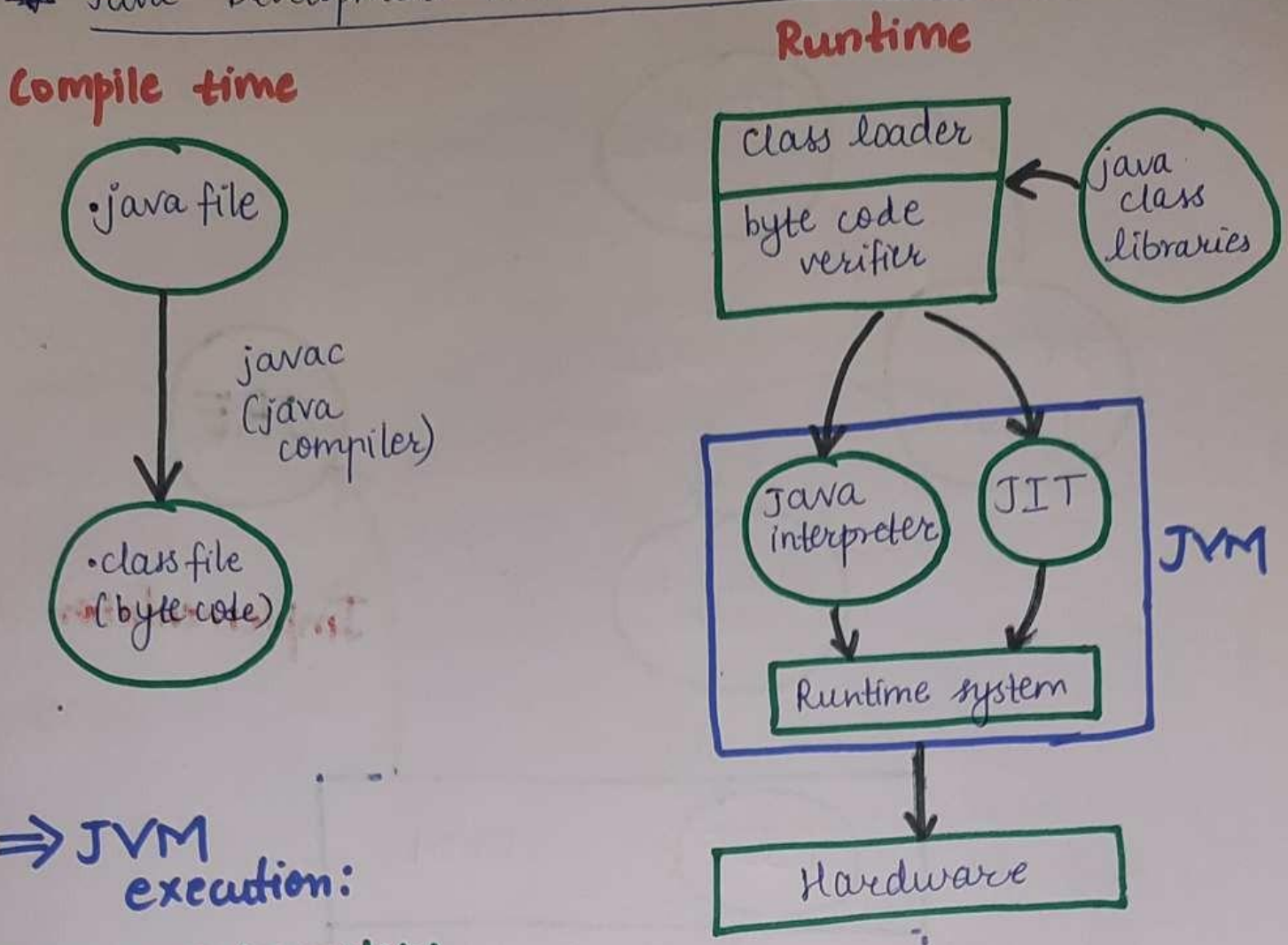
- → Java Interpreter
- → Garbage collector

etc.

- → development tools
- → javac → Java compiler
- → archiver → jar
- → docs generator ↳ javadoc
- → interpreter/loader

etc.

✱ Java Development and Runtime Environment



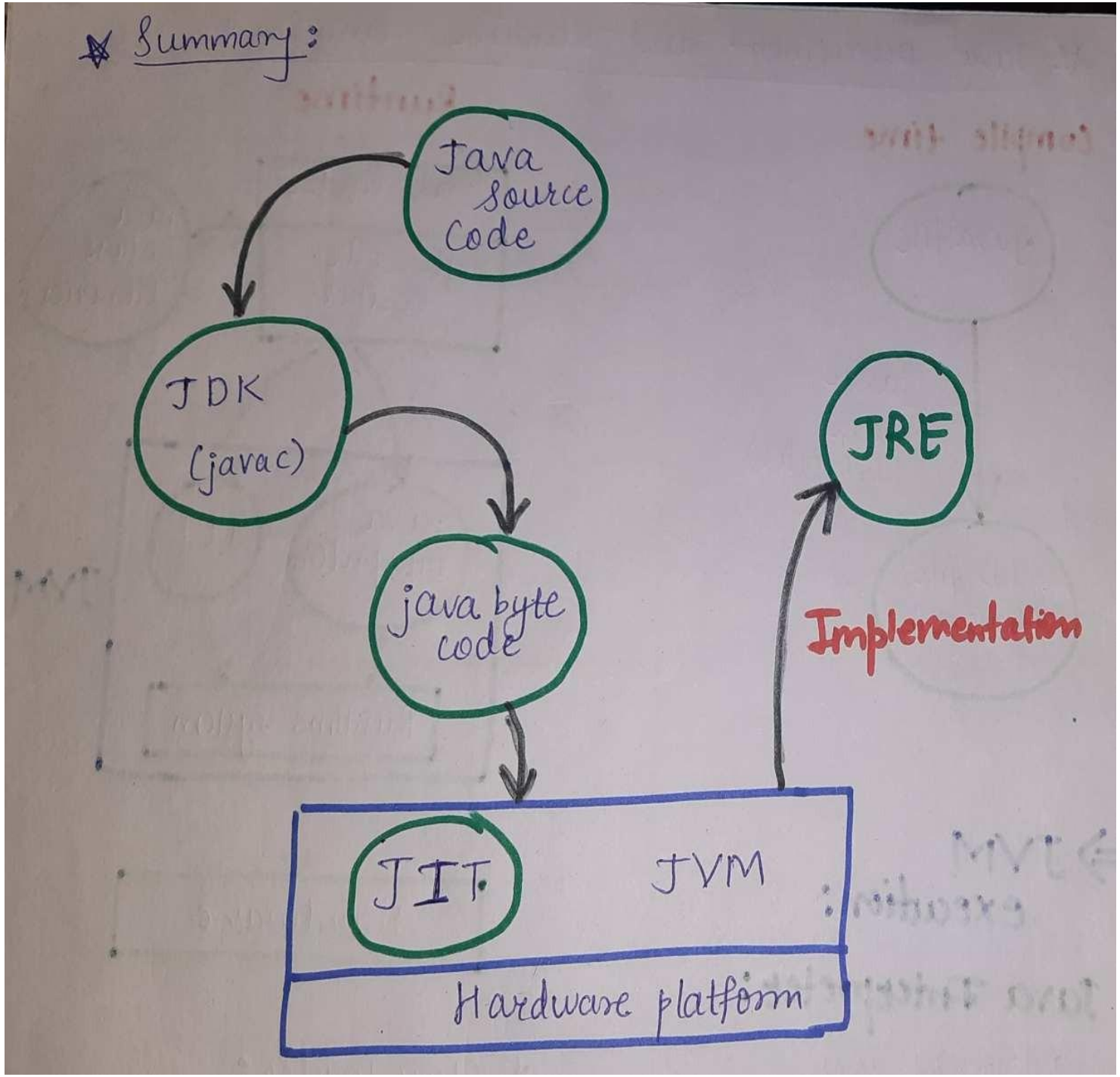
⇒ JVM execution:

- **Java Interpreter:**
  - → line by line execution
  - → when one method is called many times, it will interpret again & again
- **JIT:**
  - → methods that are repeated, JIT provides direct machine code so re-interpretation is not required
  - → makes execution faster
- **Garbage Collector**

✱ Class loader:

- **Loading**
  - → reads byte code file & generates binary data
  - → an object of this class is created in heap
- **Linking**
  - → JVM verifies .class file
  - → allocates memory for class variables & default values
  - → replace symbolic references from the type with direct references
- **Initialization**
  - → all static variable are assigned with their values defined in the code & static block

✯ Summary:


Java Source Code
JDK (javac)
java byte code
JRE
Implementation
JIT
JVM
Hardware platform

Questions

Introduction to Java – Interview Q&A

1. What is Java?

Answer: Java is a high-level, object-oriented, and platform-independent programming language. It follows the **"Write Once, Run Anywhere" (WORA)** principle, meaning compiled Java code can run on any operating system with a **Java Virtual Machine (JVM)**.

2. Why is Java called a platform-independent language?

Answer: Java is platform-independent because it does not compile directly to machine code. Instead, the Java compiler (javac) converts source code into **bytecode**, which can run on any system with a JVM, regardless of the underlying OS.

3. What are the key features of Java?

Answer:

1. **Platform Independent –** Runs on any OS with a JVM.
2. **Object-Oriented –** Uses objects and classes for better code organization.
3. **Secure –** No direct memory access, reducing security risks.
4. **Robust –** Includes garbage collection and strong memory management.
5. **Multithreading –** Supports concurrent execution of multiple tasks.
6. **High Performance –** Uses Just-In-Time (JIT) compiler for faster execution.

4. What do you mean by "Write Once, Run Anywhere" (WORA)?

Answer: This means Java code is written once and compiled into **bytecode**, which can run on any operating system with a JVM, making Java platform-independent.

5. What is bytecode in Java?

Answer: Bytecode is the intermediate machine-independent code generated by the Java compiler (javac). The JVM converts bytecode into machine code specific to the OS during execution.

6. What is JVM, JRE, and JDK?

Answer:

- **JVM (Java Virtual Machine):** Executes Java bytecode.
- **JRE (Java Runtime Environment):** Includes JVM + libraries to run Java applications.
- **JDK (Java Development Kit):** Includes JRE + development tools (compiler, debugger).

7. What is the difference between JDK, JRE, and JVM?

Component	Purpose
JVM	Runs Java bytecode.
JRE	JVM + libraries needed to run Java programs.
JDK	JRE + tools for developing Java applications (compiler, debugger).

8. What is the role of the Just-In-Time (JIT) compiler in Java?
Answer: The JIT compiler improves Java performance by converting frequently used bytecode into **native machine code** at runtime, reducing interpretation overhead.

9. Explain the Java program execution process.
Answer:

1. **Write Java Code** (.java file).
2. **Compile using javac**, generating **bytecode** (.class file).
3. **Run with JVM**, which interprets bytecode and executes it.

Flowchart:
Java Code (.java) → Compiler (javac) → Bytecode (.class) → JVM → Output

10. What is automatic garbage collection in Java?
Answer: Java's **Garbage Collector (GC)** automatically removes **unused objects** from memory, preventing memory leaks. This makes Java memory management more efficient than C/C++.

11. What are the different memory areas managed by JVM?
Answer:

1. **Heap Memory –** Stores objects.
2. **Stack Memory –** Stores local variables and method calls.
3. **Method Area –** Stores class metadata and static variables.
4. **PC Register –** Stores the current instruction address.
5. **Native Method Stack –** Stores data for native methods.

12. What is the main method in Java, and why is it needed?
Answer: The main() method is the entry point for Java applications. It is required for execution.

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

13. Why is the main() method static in Java?
Answer: The main() method is static because JVM calls it **without creating an object**. If it were non-static, JVM would need an object, but there would be no entry point to create one.

14. What happens if we remove static from the main() method?
Answer: The program **will not run**, and the JVM will throw an error:
Error: Main method is not static in class...
Since JVM calls main() directly, it **must be static**.

15. Is Java a purely object-oriented language?
Answer: No, Java is **not 100% object-oriented** because it supports **primitive data types** (int, float, char, etc.), which are not objects.

16. What are the four main principles of Object-Oriented Programming (OOP)?
Answer:

1. **Encapsulation –** Wrapping data and methods inside a class.
2. **Abstraction –** Hiding implementation details from the user.
3. **Inheritance –** Reusing properties from a parent class.
4. **Polymorphism –** A method behaving differently based on input.

17. Why is Java considered secure?
Answer: Java is secure because:

- It **does not use pointers**, preventing memory access vulnerabilities.
- It has **automatic memory management** using Garbage Collection.
- It runs inside a **JVM sandbox**, preventing unauthorized access.
- It includes **bytecode verification**, preventing malicious code execution.

18. Where is Java used in real-world applications?
Answer: Java is used in:

- **Web applications** (Spring, Hibernate).
- **Android development** (Kotlin is Java-based).
- **Enterprise software** (Banking, E-commerce).

- **Game development** (Minecraft is built in Java).

19. Why doesn’t Java support multiple inheritance?
Answer: Java does not support multiple inheritance to avoid **ambiguity** when two parent classes have the same method. Instead, Java supports **multiple inheritance using interfaces**.

20. What is the difference between Java and C++?

Feature	**Java**	**C++**
Platform-Independent	Yes (JVM)	No (compiled to OS-specific code)
Memory Management	Automatic (Garbage Collector)	Manual (pointers)
Multiple Inheritance	Not supported (uses interfaces)	Supported
Security	More secure (no pointers)	Less secure (uses pointers)
Performance	Slower (interpreted + compiled)	Faster (fully compiled)

21. What is the difference between Java and Python?

Feature	**Java**	**Python**
Speed	Faster (compiled + interpreted)	Slower (fully interpreted)
Syntax	Strict (curly braces, semicolons)	Simple (indentation-based)
Memory Management	Automatic (GC)	Automatic (GC)
Usage	Enterprise, Android, Web	Data Science, AI, Web