4. Introduction to Java.

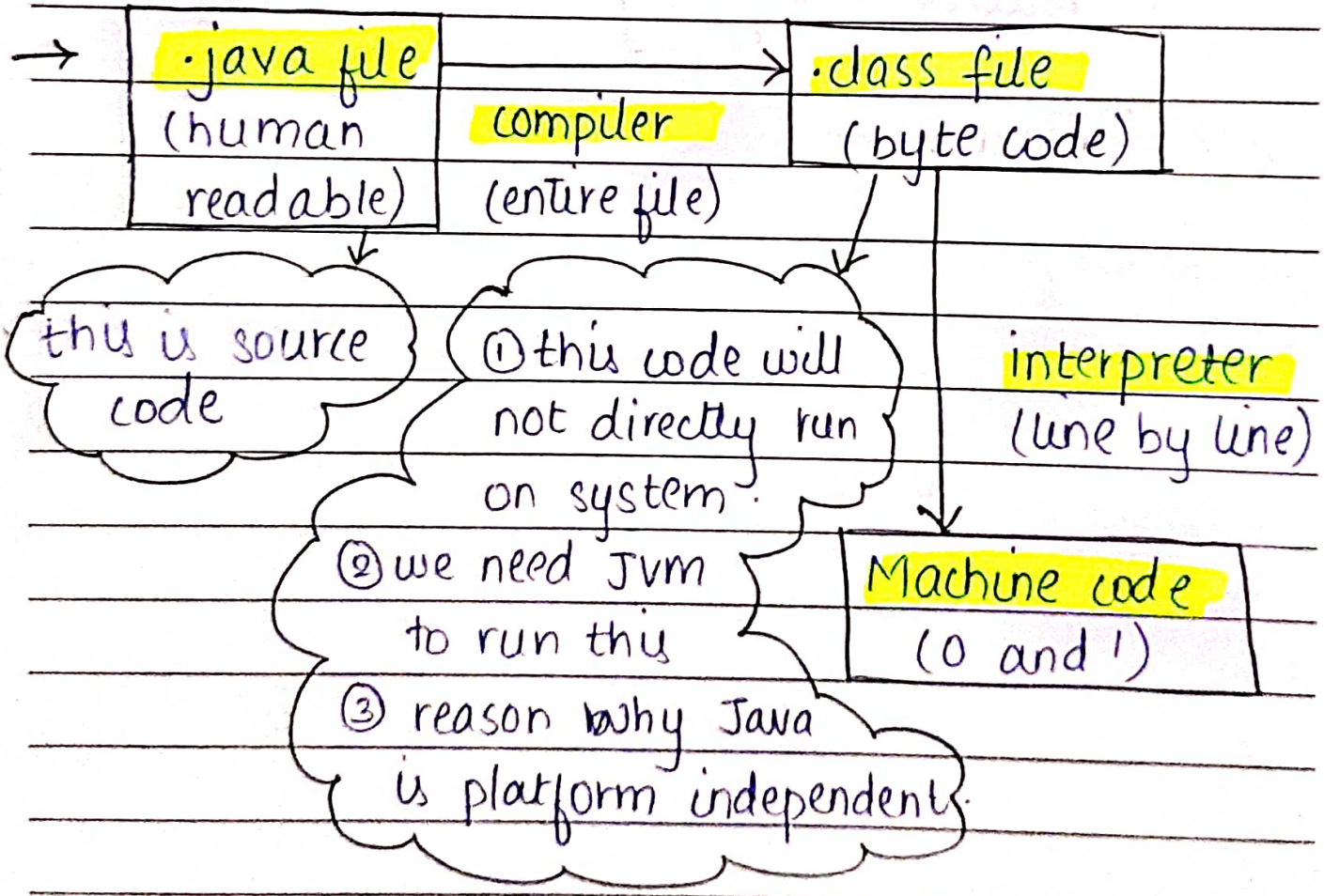
03/08/21

Q. Why programming languages are used ?

→ Machine only understands machine code which is in zeros & one. To make code more readable to humans, programming languages are used.

Extension of java file : .java

Q. How Java code executes :



__/__/__

- Byte code is some intermediate language of Java.

- In C++, byte code conversion part is skipped and human readable code is directly converted to machine code.

* **About Platform Independence:**

① Byte code would run on all OS.
② Need to convert source code to machine code so computer can understand that.
③ Compiler helps by turning into executable code.
④ Executable file (i.e. .exe file) is a set of instruction for computer which depends on architecture of computer indirectly platform dependent. But in Java because of byte code, it becomes platform independant.
⑤ JVM (Java Virtual Machine) converts byte code to machine code.
⑥ Java → Platform-Independent.
JVM → Platform-Dependant (because of OS).

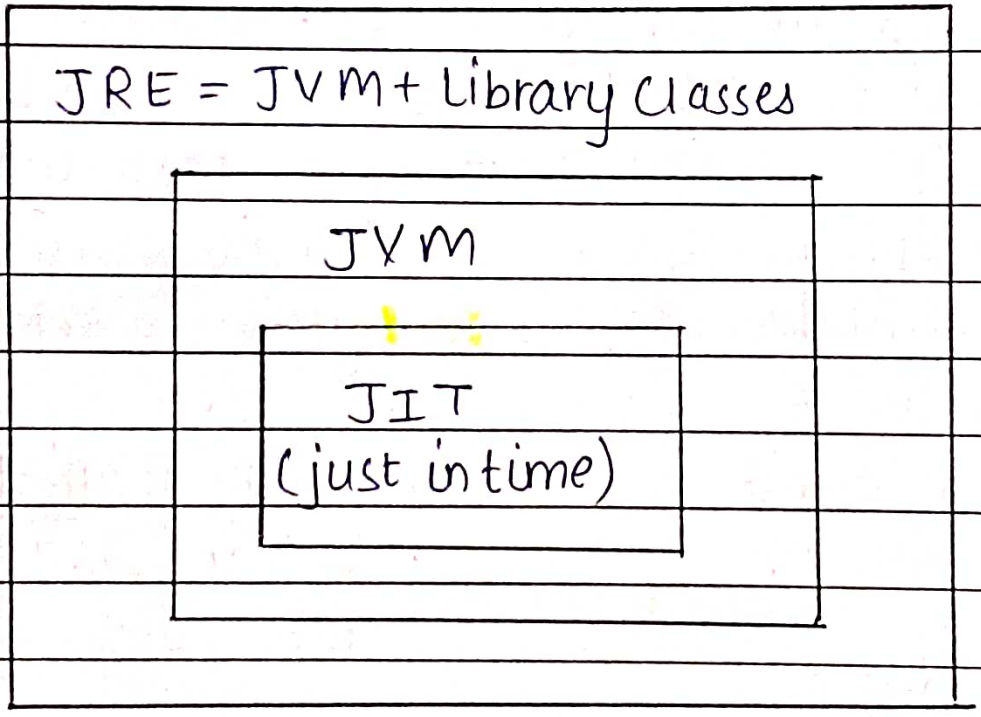
__/__/__

* Architecture of Java :

JDK - Java Development Kit.
JRE - Java Runtime Environment.
JVM - Java Virtual Machine.
JIT - Just In Time.

JDK vs JRE vs JVM vs JIT

JDK = JRE + Development Tools.



__/__/__

① JDK - Java Development Kit.

- required to develop & run Java Programs.

- it is a package that includes:
development tools, JRE, a compiler (javac) archiever (jar), docs generator (javadoc), interpreter/loader.

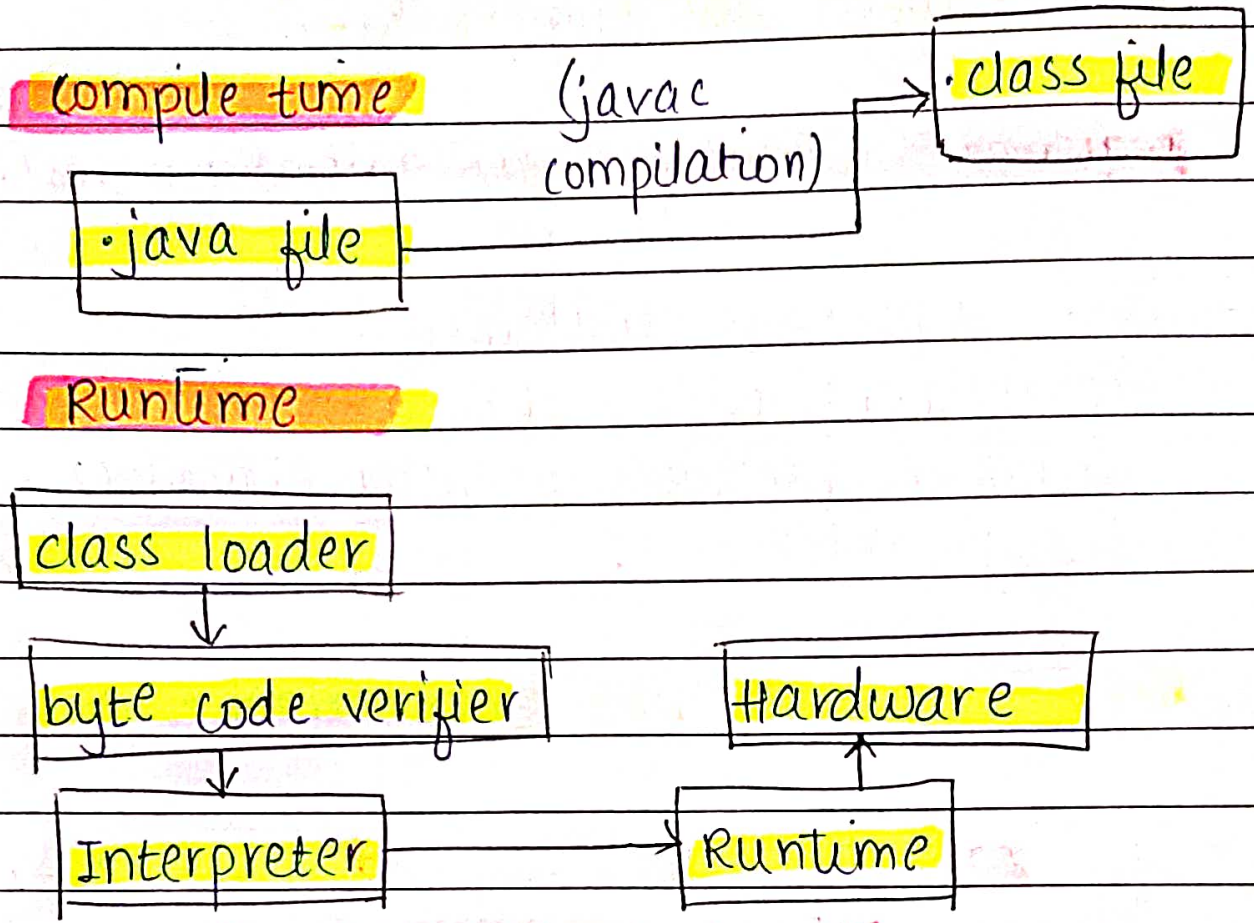
② JRE - Jave RunTime Environment.

- you cannot create programs using this, you can only run your programs.
It is inside JDK.

- consists of:
deployment techs, UI toolkits, integration libs, base libs, JVM.

- after getting byte code (.class file), things that happen at runtime are:
(1) All classes are loaded by class loader
↓
needed to execute program.
(2) Format of code is checked by JVM by sending it to byte code verifier.

* Entire Explaination :



- How JVM works :- Class loader .

① Loading :
- reads class files & generates binary data.
- an object of this class is created in heaps.

② Linking :
- JVM verifies .class files.

__/__/__

- allocates memory for class vars & default values.
- replaces symbolic references from the type with direct references. (will be explained in brief in functions)

③ Initialization :
- all static vars are assigned with their values defined in code & static block. (OOPS).

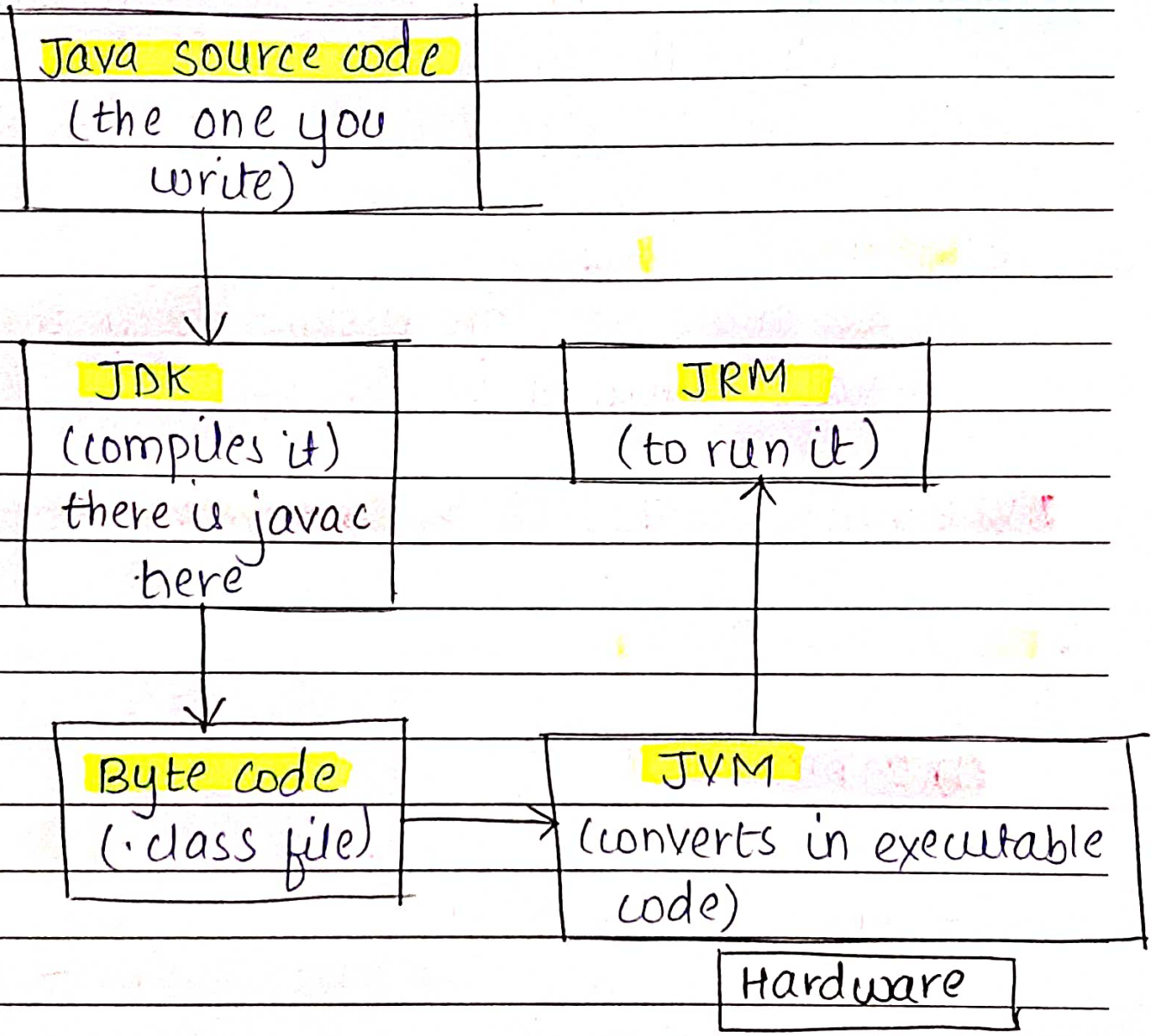
JVM contains stack & heap memory allocations

- JVM Execution.

Interpreter :
- line by line execution.
- when one method is called many times, it will interpret again and ~~help~~ again.

- JIT - Just in time :
- provides direct machine code so no reinterpretation is required.
- makes execution faster.
- garbage collector.

* How this entire thing works collectively :



JRE is like a box & JVM is actual content inside the box.

compiler is only in JDK.