MODULE - 0

LOGIC BUILDING

what is LOGIC ?

the collections of well-defined activities to Performed to solve the problem.

Example :- Adding Two Numbers

① Get a 2 number ② use Addition operator (+) ③ save the result ④ Print the result)

=> Logic can be written in many ways like,

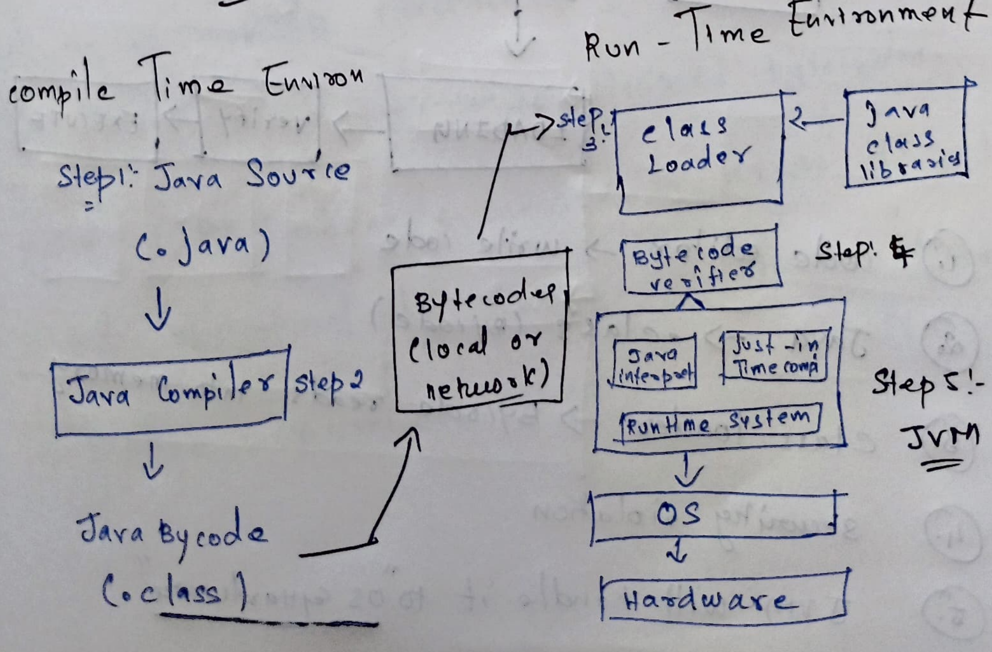
① Pseudo Code ② Flow chart

BUT ? How computer need to solve by this logic

LOGIC -> PROGRAM (CODE)
(Java, C, C++) etc...

TECH MODULE - 1

Java Architecture



Step 1:- Java Source code (.java)

Step 2:- source code -> Bycode code (.class)

Step 3:-

"CLASS LOADER" -> Reads both "user defined" & "library classes" into memory for execution

Step 4:

"BYTECODE". verifier -> validate all bytecodes

(Due to DO NOT VOLATILE JAVA SECURITY RESTRICTIONS)

Step 5:-

JVM reads bytecodes & "translates" into machine code for execution.

i.e -> while execution, the program will interact to the OS & hardware.

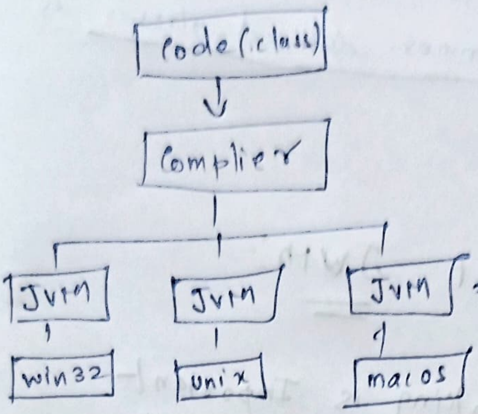
5 Phases

$$\begin{tikzcd}
\boxed{\text{EDIT}} \arrow[r] & \boxed{\text{Compile}} \arrow[d] & & \\
& \boxed{\text{LOADING}} \arrow[r] & \boxed{\text{verify}} \arrow[r] & \boxed{\text{EXECUTE}}
\end{tikzcd}$$

1. " code editor -> write code
2. Java -> .class (Bycode)
3. class loader -> Bycode read's into memory
4. security violation
5. JVM will handle it to "OS & Hardware".

Java Virtual Machine

- output of the "Compiler" is bytecode
- Bytecodes are executed by Jvm.



"Platform dependent is Jvm But code is platform independent"

- Jvm will differ to differ for Platform.
  - ↳ platform - specific.
- Interpreted code runs → "slow compare to execution"

Execution → Fast (Complilers)

Interpreted → slow (Bytecodes)

The Adaptive Optimizer

- Another type of execution engine
- virtual machine → starts interpret bytecodes.
- But if any "heavily used code areas" are not optimized by Jvm. so At the "time used".

Adaptive optimizer

The class loader (.class)

why?

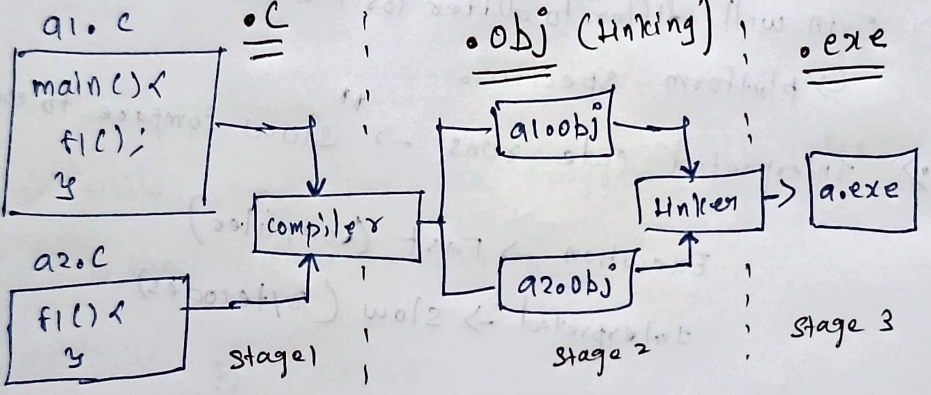
1.) A security stand point

2.) Network mobility

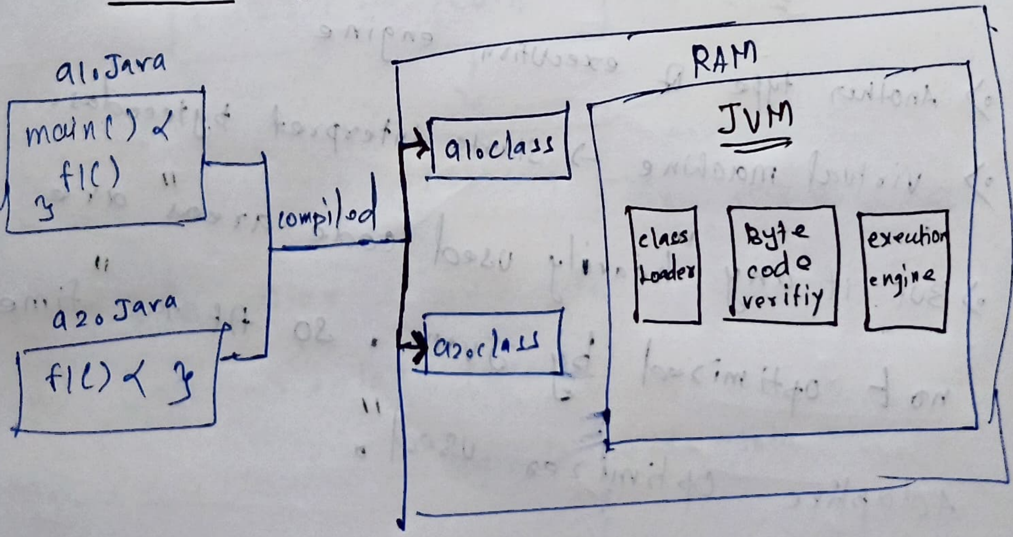
=> The bootStrap class loader is responsible for loading the classes, programmer defined classes as well as Java's API classes.

Complete Flow of J.W.M:-

In C :- → Linking is Important



In Java:- (NO LINKING)


a1.Java
main(){
f1()
}
a2.Java
f1(){ }
compiled
RAM
a1.class
a2.class
JVM
class Loader
Byte code verifiy
execution engine