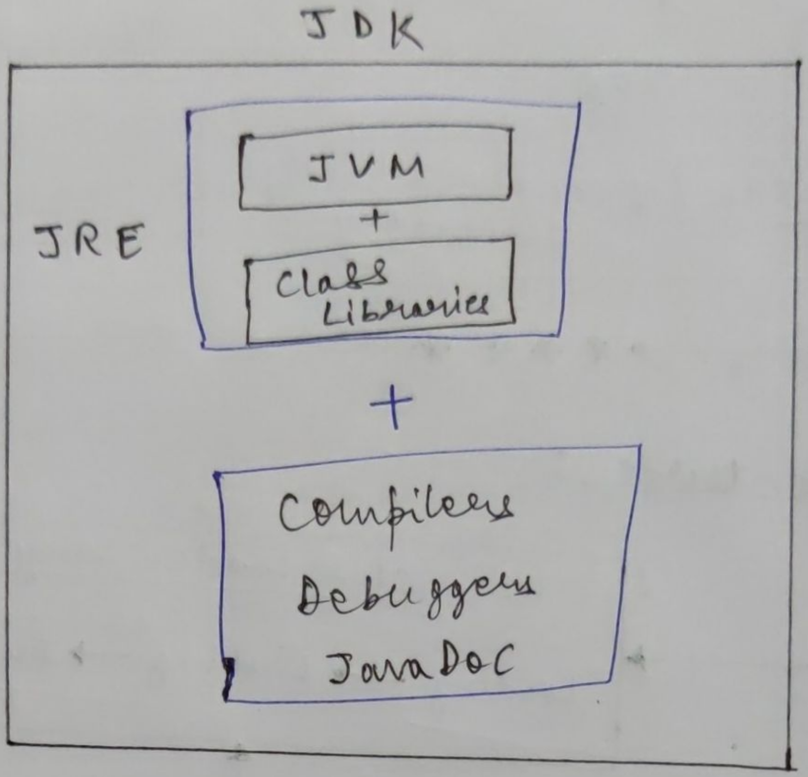
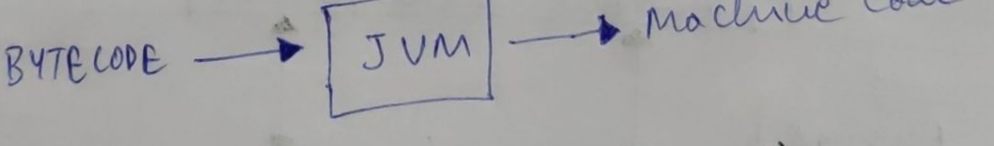
JDK vs JRE vs JVM



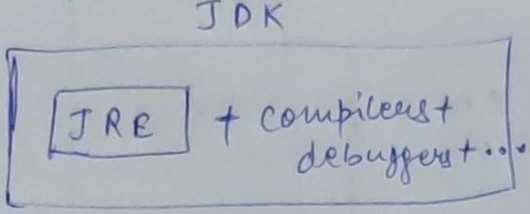
- **JVM** :- (Java virtual Machine)
  - ↳ JVM makes java codes/programs platform independent.
  - ↳ JVM converts Bytecode to Machine code



- **JRE** :- (Java Runtime Environment)
  - ↳ If we need to run a Java program but, not develop them, JRE is what we need.
  - ↳ It is a Software package that provides Java Class Libraries + JVM + and other components to run Java applicat^ns. ~~to run~~

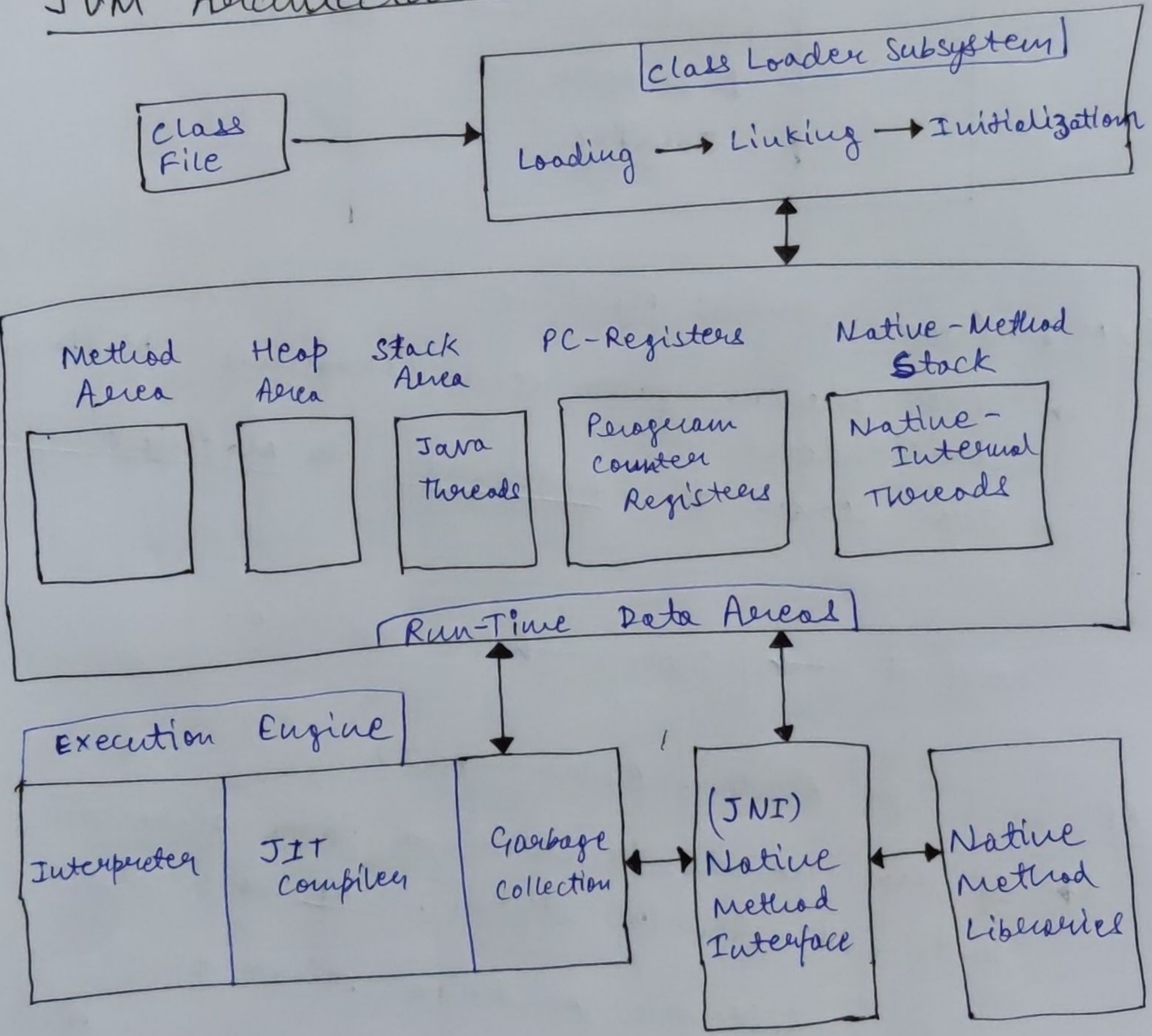
• <u>JDK</u> :- (Java development kit)

↳ A software development kit required to develop applications in java.



* * * * *

JVM Architecture :-



The JVM is divided into 3-main subsystems :-

1) Classloader subsystems
2) Runtime Data Area
3) Execution Engine

1) Classloader Subsystem :-

- Loading
  - Bootstrap classloader : Highest priority
  - Extension classloader :
  - Application classloader :

- Linking
  - verify : Bytecode is verified. (gives verification error)
  - prepare : For all static variables memory will be allocated & assigned with default values.
  - Resolve :

- Initialization
  - Final phase, here (all static variables will be assigned with original values.,)
    &
    (Static block will be executed.)

2) Run-Time Data Areas:

Method Area

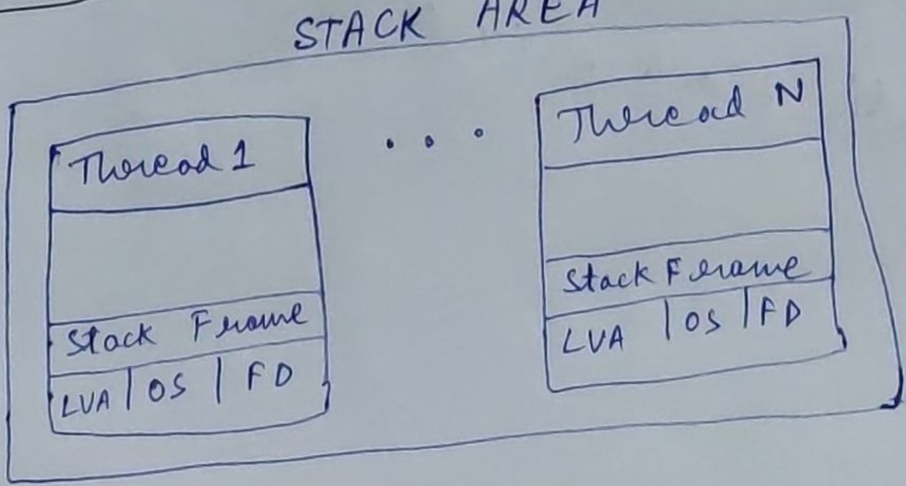
- All class level data will be stored here, including static variables.
- 1-Method area / JVM.
- It is a shared resource.
- Not Thread safe.

Heap Area

- All "objects", their corresponding "instance variables" & "Arrays" will be stored here.
- 1-Heap area / JVM.
- It is also a shared resource.
- It is "Not Thread safe";

Since both method & Heap area share memory for multiple threads.

Stack Area



→ For every thread, a seperate runtime Stack will be created.

→ For every method call, one entry will be made in the stack memory which is called "STACK FRAME" (it is pushed on top of the Thread's stack)

→ It is "Thread-Safe".
since it is not a shared memory.

→ Stack Frame is divided into 3 - subentities;

(i) Local Variable Array
↳ all local variables related to the method and their corresponding values will be stored here.

(ii) Operand Stack
↳ If any intermediate operation is required, Operand Stack acts as runtime workspace to perform the operation.

(iii) Frame Data
↳ in case of Exception, the catch block information will be maintained in the frame data.

PC Registers -

- Each thread will have seperate PC-Registers.
- It holds the address of current executing instruction.
- once instruct^n is executed, PC-Register will be updated with the next instruction

Native Method Stack

- It holds "Native Method" informat^n. For every thread, a separate native method stack will be created.

Native keyword :-

- it is applied to a method to indicate method is implemented in native code using JNI (Java native Interface).
- methods implemented in C, C++ are called native methods.
- such method usually have platform-dependent code.

- if we are running JVM on windows, it will contain windows-related info. if on Linux, it will have all the Linux related info.

3) Execution Engine :

- The bytecode, which is assigned to Run Time data area, will be executed by the Execution Engine.
- It reads Bytecode & executes it piece by piece.

(i) Interpreter

- interpreets bytecode faster but executes slowly.
- Disadvantage : when one method is called multiple times, everytime a new interpre-tation is required.

(ii) JIT compiler → (converts Bytecode to Machine Code at Runtime)

- It neutralizes the disadvantage of the intrepreter.
- For repeated code JIT compiler compiles the entire bytecode & changes it to Native code.
- This Native code is used directly for repeated method calls.

4 - Parts

- Intermediate code generator
- Code optimizer
- Target code generatore → (generates Machine code / Native code)
- Profiler → (responsible for finding hotspots. whether method is called multiple Times or not)

NOTE : Java is both Compiled & Interpereted

(iii) Garbage Collector

↳ collects & Removes unreferenced objects.

Java Native Interface (JNI)

↳ it interacts with the Native Method Libraries & provides the Libraries required for the execution Engine.

Native Method Liberaries

↳ collection of Native Libraries, required for Execution Engine.