Gemini
Sunfoundry

11 + 11 + 10 + 11 + 10 + 12

classmate
Date
Page

Assignment - 1 03/09/24

* A short history of Java -

- Java was first developed in 1991 by team of engineers at Sun microsystem.
- Java was developed by Sun microsystems, technology company.
- Green Project - it was internal project at Sun microsystems, aimed at finding new opportunities in the Consumere electronics industry.
- Java was originally created as a part of this project to run on this devices.

- Green Team -
[James Gosling, Patrick Naughton, mike sheridan
↑
"Father of Java"

* The original language named Oak, was later rename to java in 1995
- The name java was inspired by Java Coffee

* The "*7" Device (1992) -
In 1992 the team developing java created a device called "*7" to show how their new technology could work in smart gadgets.
* Failure of "*7" - Time-warner declined to use java for their set-top boxes & video-on-demand service.

* Breakthrough with the web (1994) –

In 19994, java gained significant attention when team developed a web browser called webRunner.

This browser had unique ability to run applets, → [applets - is a small Java program], that could be embedded in web pages & run directly within the browser. It was major breakthrough.

* First public Implimentation – Java 1.0 in 1996

Java 1.0 was the first official version of java made available to the public in 1996.

* Acquisition of java – In 2010, oracle Corporation, a major software & hardware Company, acquired Sun microsystems.

* Slogan → "Write once, Run Anywhere".

Java's slogan highlights its most powerful features → Platform independant

It means Java program run of one platform (like windows) it can run on any other platform (like linux or macOS) without needing modification, as long as platform has JVM installed

* Java Virtual Machine [JVM]

When we write code, it gets compiled into bytecode, which the JVM can interpret & execute on any device & OS. This makes java platform-independent.

* Automatic memory management - Java handles ~~process~~ memory automatically through process called 'garbage collection.

* Java Editions -

1) Java SE [Standard Edition] =

The core version of java that includes the basic libraries & tools needed for general-purpose programming.

Key components → Java appln Programming Interface (API) & JVM

* Used for → Developing desktop applications, Command line tools, small to medium-sized applications.

2) Java EE → Extended version of Java SE, with additional libraries & frameworks designed for building large-scale, web-based application.

Key components → Servlets, JSP, Filter, JPA, JTA, JMS, EJB, Jsf, Java mail

@) Used for - Developing enterprise-level applications such as online banking systems, e-commerce platforms & large-scale web services.

3) Java ME (Micro Edition)

A version of java designed for mobile & embedded devices with limited resources

Used for - Developing appln for smartphones, tablets & small appliances like IoT devices.

Java SE - Basic Java Programming
Java EE - Enterprise level application
Java ME - mobile & embedded devices
Java fx - rich, modern user interfaces.
It is used for building rich, modern user interfaces for desktop appl
Java Card - used for developing appl for Smart Card & secure IoT devices.

1) Java lan is both, technology as well as platform.
⇒ Is not only programing language is a platform that provides runtime environment (the JVM) where java program can be executed.

2) Java's standardization is managed through the java community process. (JCP)

3) Extension of java source file .java
→ These files are written by developers & are compiled into bytecode by the Java Compiler.

4) Java is object oriented but also support procedural & functional programming paradigms.

5) Java is statically typed lang. It means type checking is done at Compile time.
6) Java is Case-sensitive lang.
7) Java, Kotlin, Scala, Groovy are some are JVM based languages these languages are desined to run on JVM. They can use java libraries & tools

What is API -

API - Appn programming Interface.
It is set of tools & protocol that allow different software appn to communicate with each other.

Java version →

1) Java 8 - Long term support

Java 11
Java 17
Java 21

* SDK → Software development kit →
it is collection of tools, libraries, documen & sample code that developers use to create appn for specific platform.
It includes every thing needed to write, compile, test & debug software making it easier & faster to develop appn.

Q. 1) Explain Components of JDK -

* JDK - (Java development kit) -

It provides the tools needed to write compile & run java code.
Writing code - JDK includes text editor & other tools to write java code
Compiling code - The java compiler (javac) converts your written code into byte code, ~~which can be run~~

Running Programs - The JDK includes the java runtime environment (JRE) + JVM which allow you to run java programs.

Java runtime Enviroment (JRE)

→ - To execute / run java appl^n on developer's machine / client's machine, we require Java Runtime Enviroment (JRE).

- The JRE comes with the JDK by default, so developers do not need to download.

- on client machine, we must first download & install the JRE.

* Components of Java Runtime environment -
  1) Java class Library (rt. jar)
  2) Java virtual machine (JVM)

↳ rt.jar is the core library file in JRE. It contains standard java classes libraries. necessary for running java application. This includes packages → java.lang, java.util, java.io

rt.jar provides fundamental classes that the JVM needs to execute java programs.

src.zip	vs	rt.jar	vs	Java api docs
Source Code of Java API		Compiled code of Java API		Documentation of Java API
src.zip		rt.jar		jdk-8u***-docs-all
Contains .java files		Contains .class files		Contains .html files

* Java program execution flow

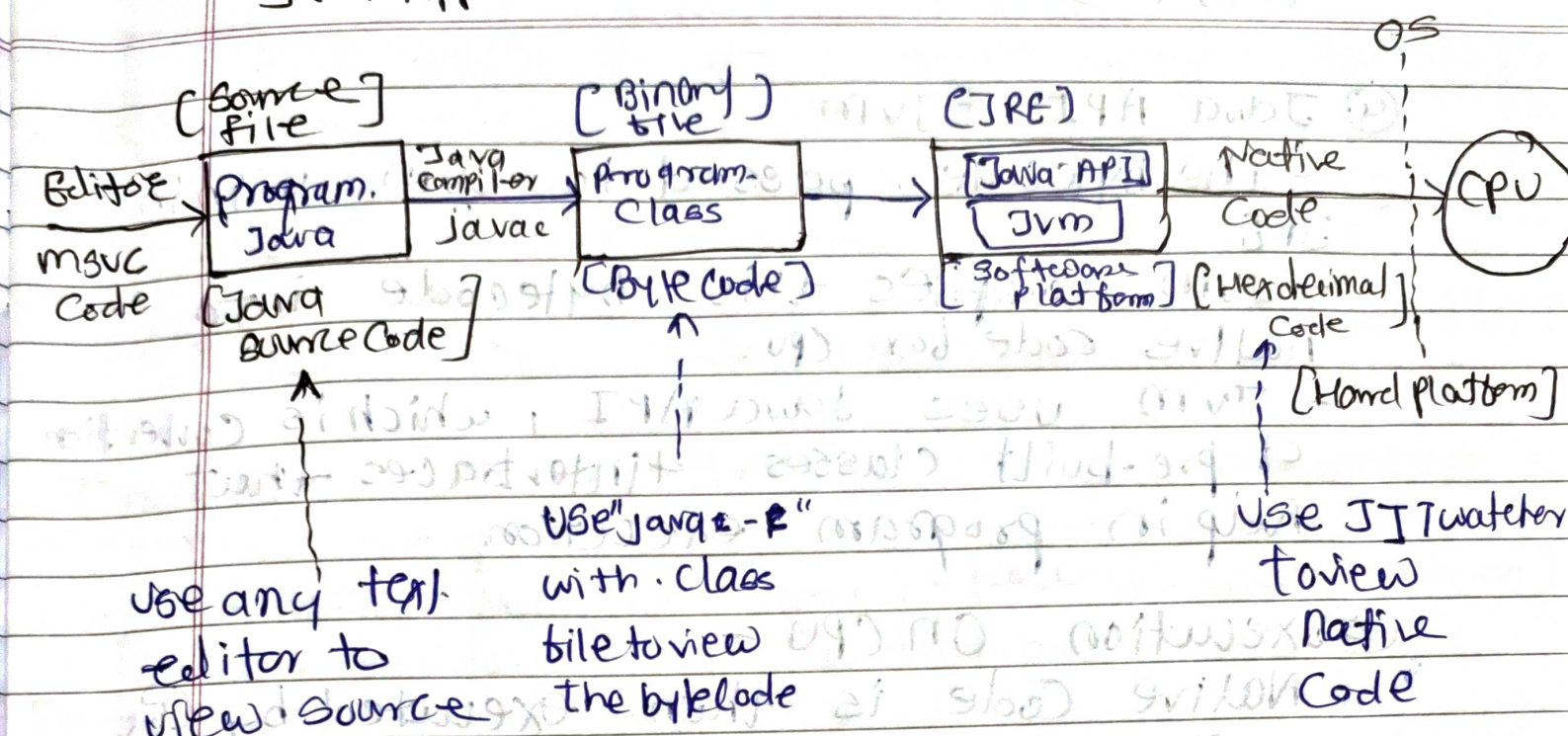
- Java is platform-independent but JVM is platform dependent.
- JRE → JVM + Library classes + Development tools
- JDK = JRE + Development tool
- JRE = JVM + Library

Windows\Program Files\Java\jee\lib

Class Loaders

- Bootstrap Class loaders
- Extension class loader
- System/Appn classloaders

Java Application Execution Flow -



① Source file (Program.java)

- we write java source code in text editor [like msvs code]
- The file is saved with a .java extension.

② Java Compiler (Javac) -

- The source file passed to java compiler which translate java code to bytecode
- output is binary file with .class extension, known as binarycode file.

③ Binary file (Program.class) -

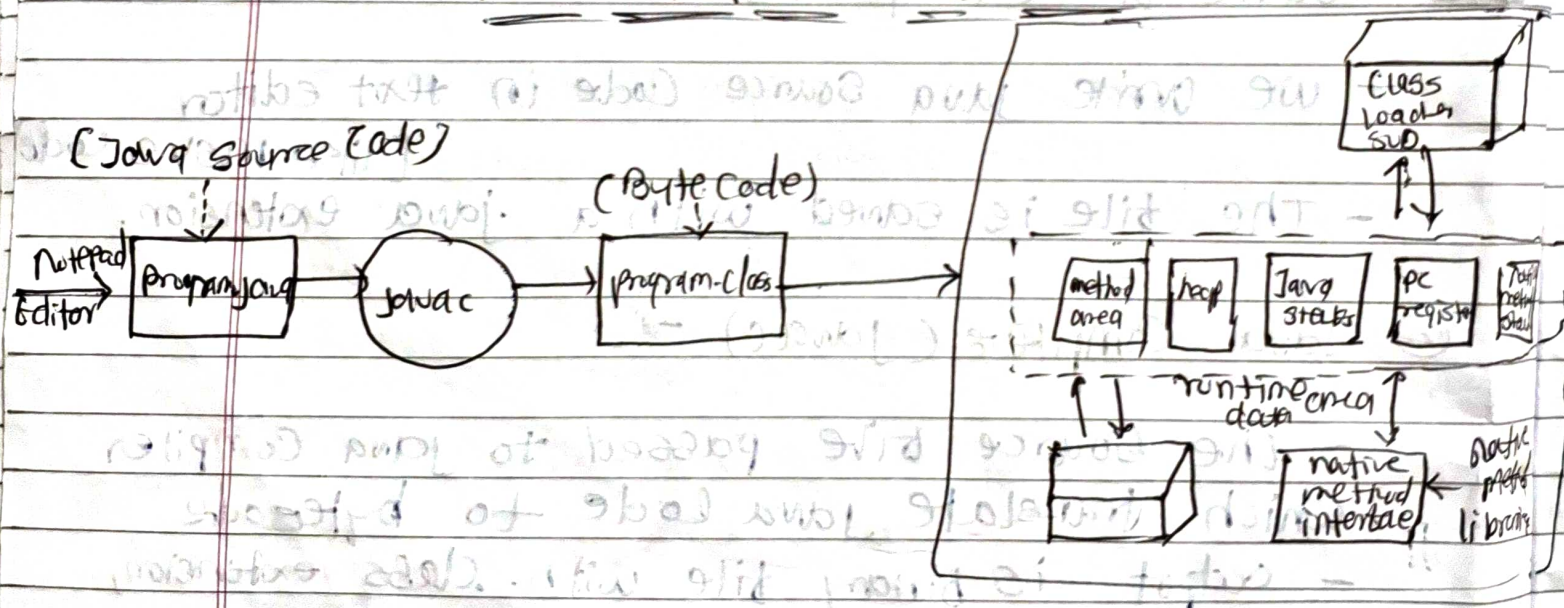
- This bytecode is platform independent can be run on any system that has JRE.
- view bytecode using javap -c

Q. 6) Describe architecture of JVM.

④ Java API & JVM
- The bytecode passed to JVM with in JRE
- JVM compiles this bytecode into native code for CPU.
- JVM uses Java API, which is collection of pre-built classes & interfaces that help in program execution.

⑤ Execution on CPU -
Native code is then executed by CPU
- JITwatcher tool use to view Native Code

* Architecture of JVM *



[जिस folder में rt.jar file रखी है उसको Bootstrap class Path.

1) Bootstrap class loader → Responsible for loading classes from bootstrap path, Nothing but rt.jar. Highest priority will be given to this loader.
2) Extension class loader → Responsible for loading classes which are inside the ext folder (jre/lib).
3) Appln class loader → Resp. for loading appln level classpath Path mentioned Environment variable etc

JVM - Divide in 3 parts → Class Loader

Hello.class - - - - - - - - - - - -

Class Loader → Memory Area → Execution Engine

* Class Loader subsystem -

Loading	Linking	Initialization
Bootstrap class Loader Extension Application	Verify Prepare Resolve	1. All static variable are assign with value 2. static block will be executed. from top to bottom.

↓ Memory Area

↓ Execution Engine

→ machine code

(i) Loading →

I) Bootstrap Class Loader → [It is first class loader that is responsible for loading the core java libraries located in the '<java_Home>/jre/lib/*.jar'.

II) Extension class loader →
- It is child class of bootstrap class loader.
- It is responsible of loading all classes from the extension class path in java.
* Extension classpath is - '<java_Home>/jre/lib/ext

III) Appn class loader - child class of extension class.
- It is resp of loding classes from the appn classpath.

Linking →

1) Verify → verification
- It checks whether the ".class" file is generated by valid compiler or not.
- If verification fails, it throws java.lang.VerifyError

In interview

2) Prepare -
JVM allocates memory for class variable & initializes them with default values.

3) Resolve -
ex- public class Main {

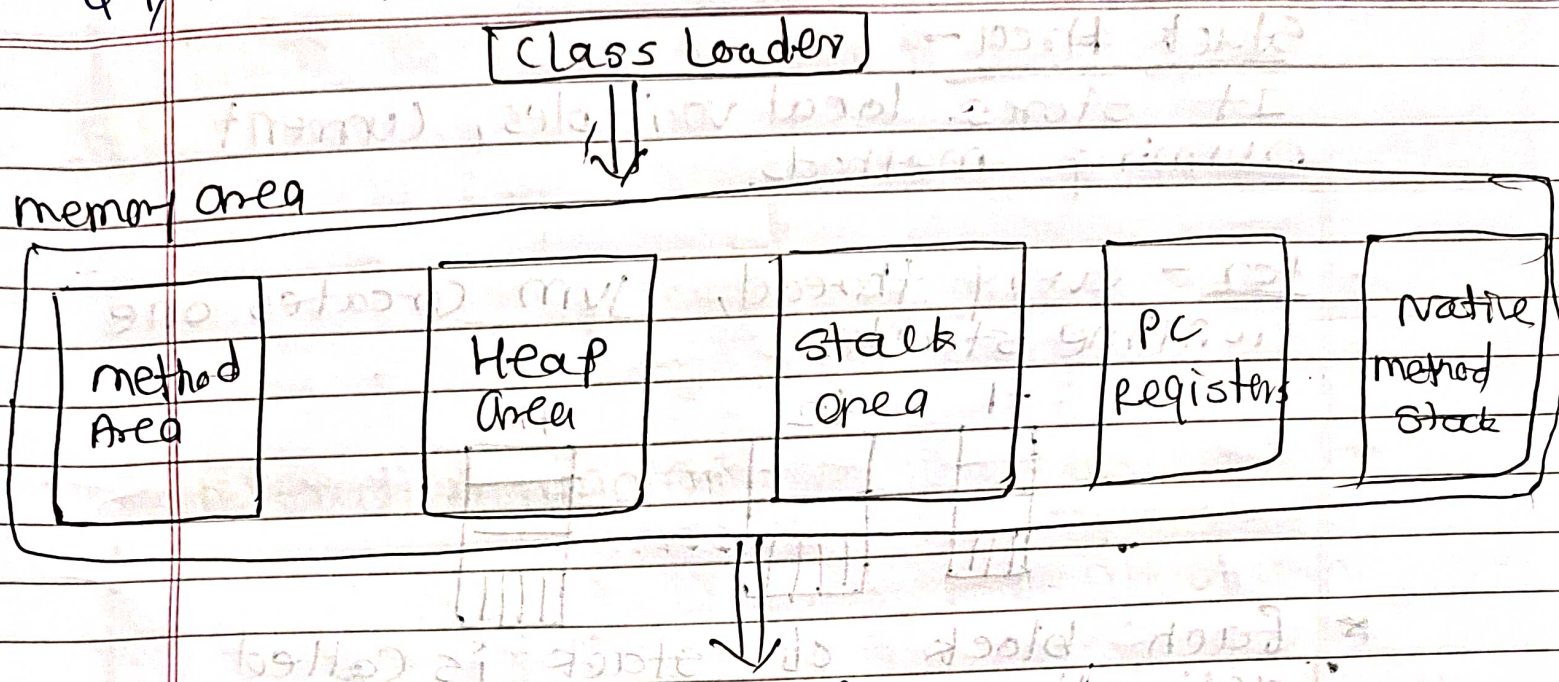
```
    psvm(String[] args) {
        Helper helper = new Helper();
        helper.performTask();
    }
}
```

The reference to helper & its method performTask in main class is symbolic. The actual memory allocations are not known at compile time.

During linking phase, JVM loads in main class. At this point, symbolic references to helper & its method need to be resolved.

The resolution step finds actual memory addresses for the helper class & its performTask method. It ensures that Main class can call correct method on correct object.

Q.4) Memory management system of JVM

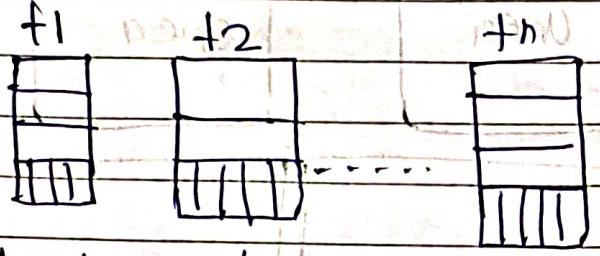


* Method Area → It stores class level into like, class name, method & variable into.
- static variables
- The method area is shared among all the threads running in the JVM (Not thread safe)
- Only one method area per JVM.

* Heap Area → It stores arrays, objects, instance variable.
- The heap area is shared among all the threads running in the JVM (Not threads safe)
- Only one heap area per JVM

Stack Area -
It stores local variables, current running methods.

For - every thread, JVM creates one runtime stack.



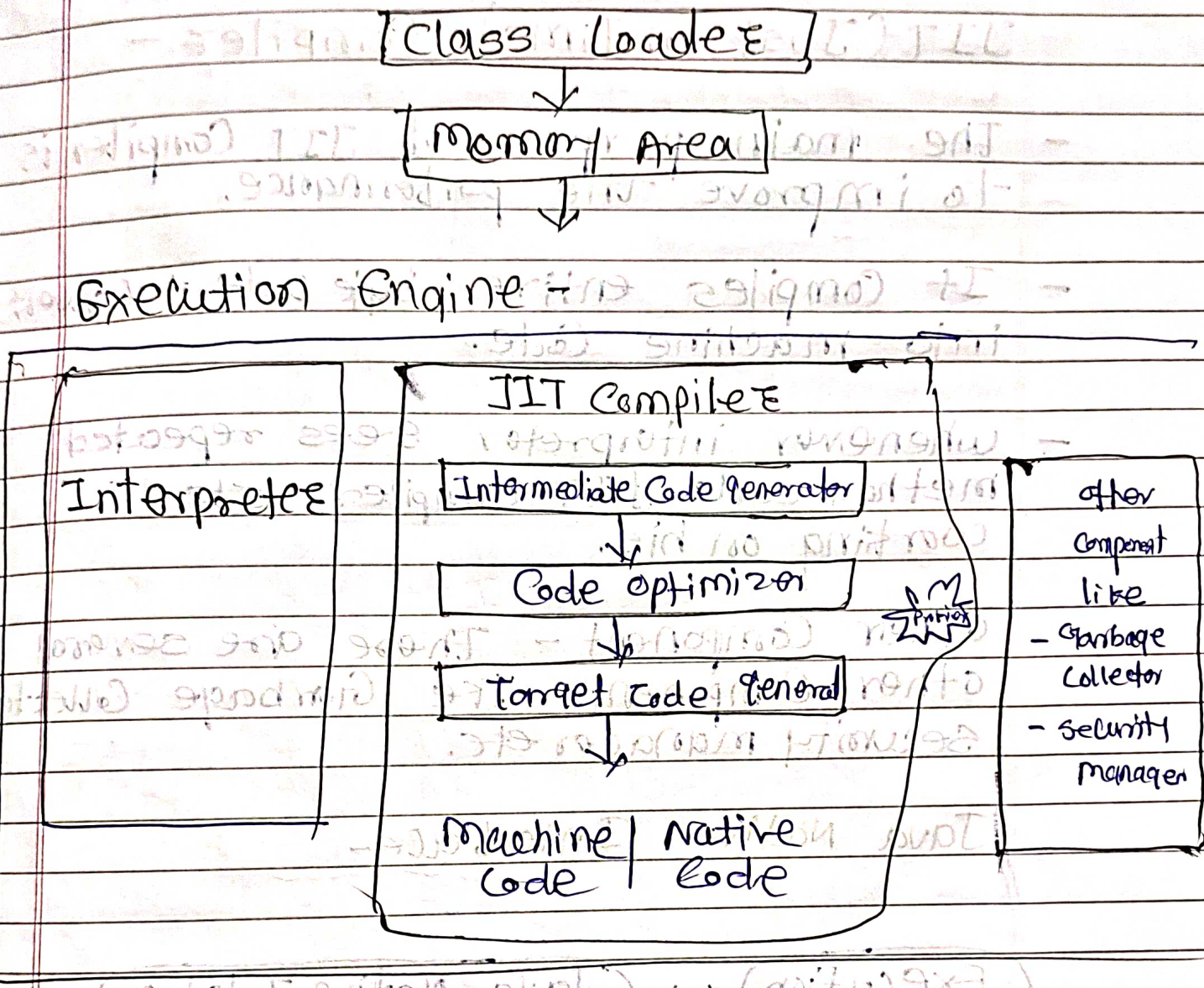
* Each block of stack is called activation record / stack frame.

* Each frame contains - local variable, frame data & operand stack.

PC Registers - It stores current execution instruction, once it completes, Automatically update the nxt PC register
- Each thread has separate PC Registers

Native Method stack - Memory used for native method execution.
- It is separate from the java stack to handle native (non-java) code.
- For every thread separate native stack is created.

Jvm Execution Engine -



Interpreter -

- It is responsible to read byte code & interprets into machine code line by line.

- The problem with interpreter is, it interprets every time even if repeated method called. Which effects the performance

- To overcome this problem JIT Compiler introduced in 1.1 version.

Q.5) what are JIT Compiler & its role in JVM.
what is bytecode & why it is imp. to in java

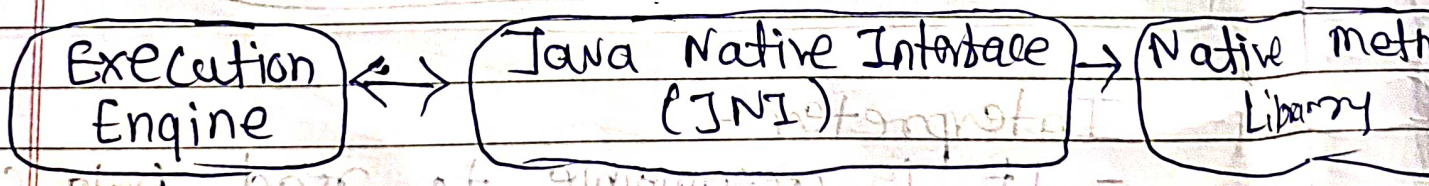
JVM - Execution engine

JIT (Just in Time) Compiler -

- The main purpose of JIT Compiler is to improve the performance.
- It Compiles entire byte code & Convert into machine Code.
- whenever interpreter sees repeated method calls JIT Compiler starts coorking on his.

other Component - There are several other Component like Garbage Collector, security manager etc.

Java Native Interface -



- JNI interact with native method library & provides the native method library to Execution engine.
- In other coord you can say, JNI is responsible to provide the native information to JVM.