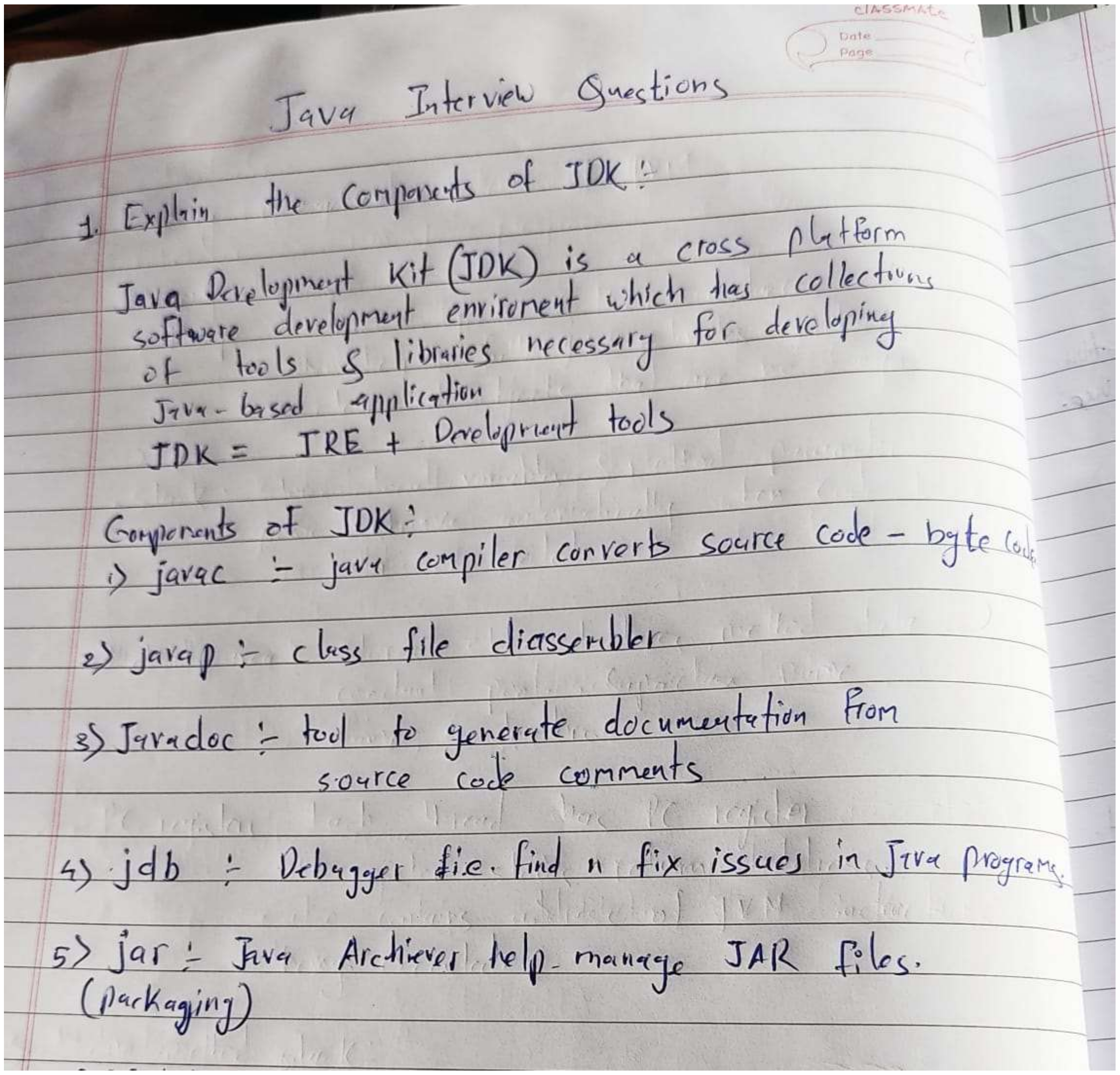
1) Explain the components of the JDK.



classmate
Date
Page

Java Interview Questions

1. Explain the Components of JDK :-

Java Development Kit (JDK) is a cross platform software development environment which has collections of tools & libraries necessary for developing Java-based application

JDK = JRE + Development tools

Components of JDK :-

1) javac :- java compiler converts source code - byte code

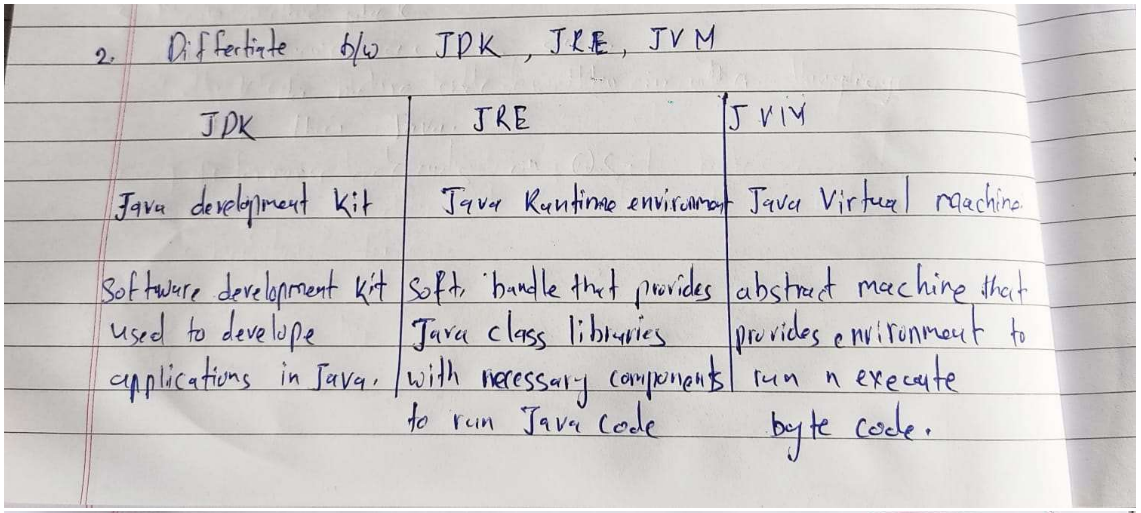
2) javap :- class file diassembler

3) Javadoc :- tool to generate documentation from source code comments

4) jdb :- Debugger i.e. find n fix issues in Java programs.

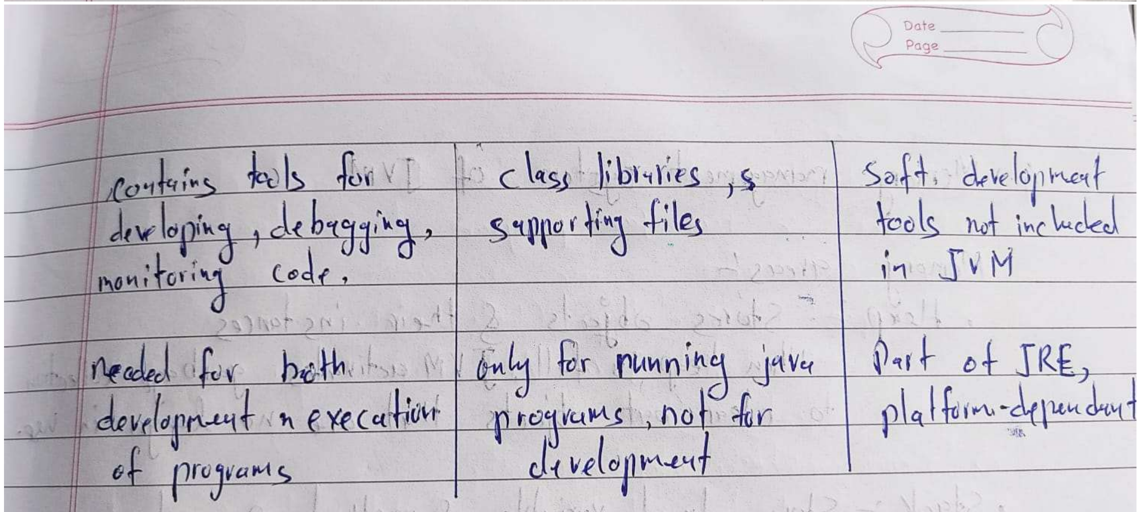
5) jar :- Java Archiever help manage JAR files. (packaging)

2.Differentiate between JDK, JVM, and JRE

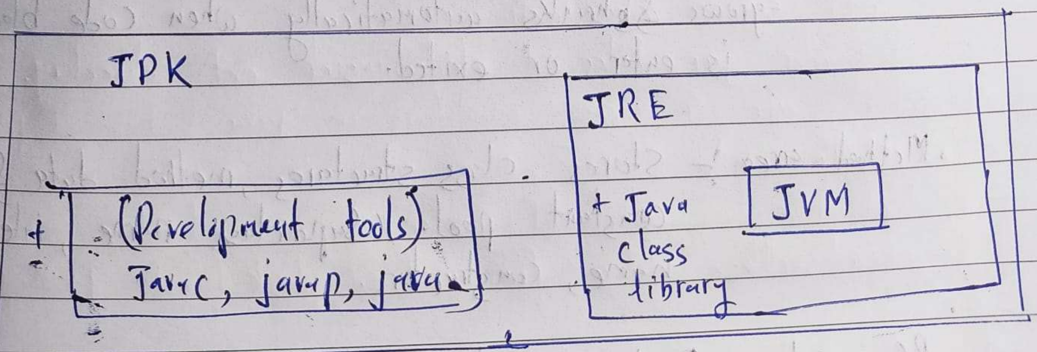


2. Differtiate b/w JDK, JRE, JVM

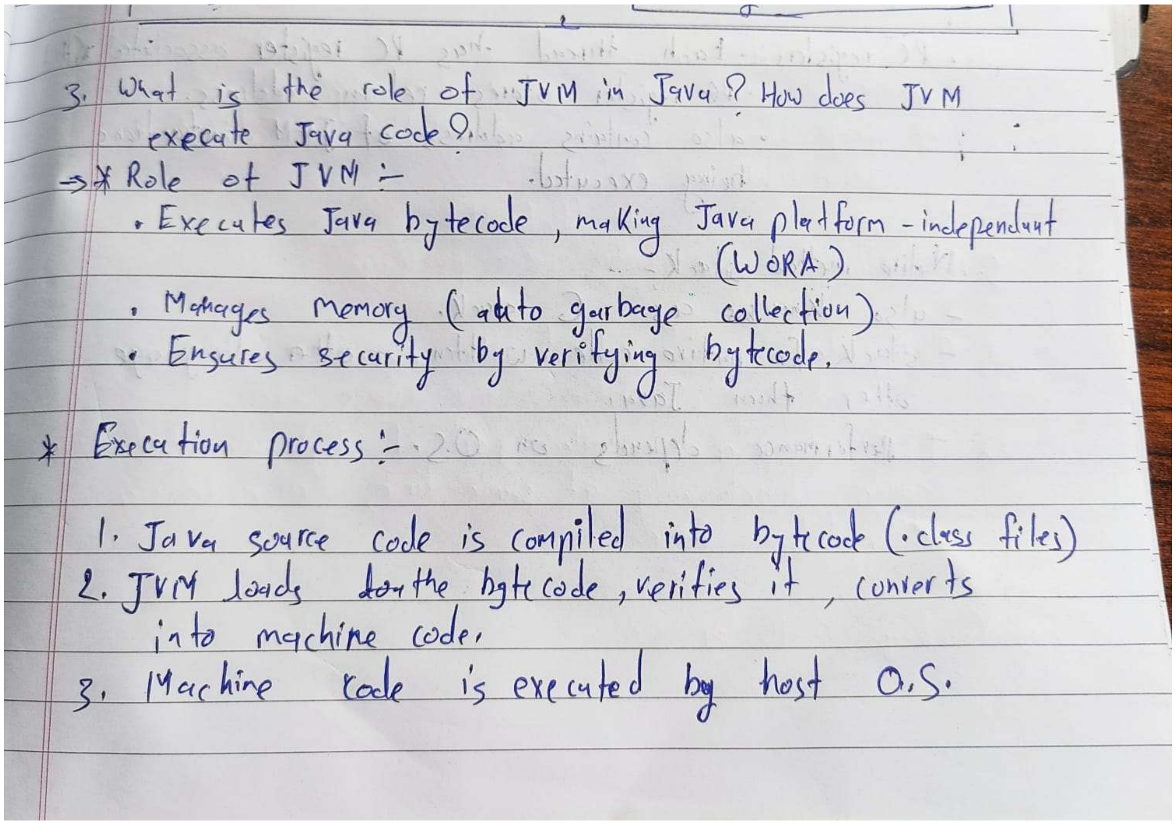
JDK	JRE	JVM
Java development kit	Java Runtime environment	Java Virtual machine
Software development kit used to develope applications in Java.	Soft. bundle that provides Java class libraries with necessary components to run Java code	abstract machine that provides environment to run n execute byte code.



contains tools for developing, debugging, monitoring code.	class libraries, & supporting files	Soft. development tools not included in JVM
needed for both development n execution of programs	only for running java programs, not for development	Part of JRE, platform-dependent



3.What is the role of the JVM in Java? & How does the JVM execute Java code?



3. What is the role of JVM in Java? How does JVM execute Java code?

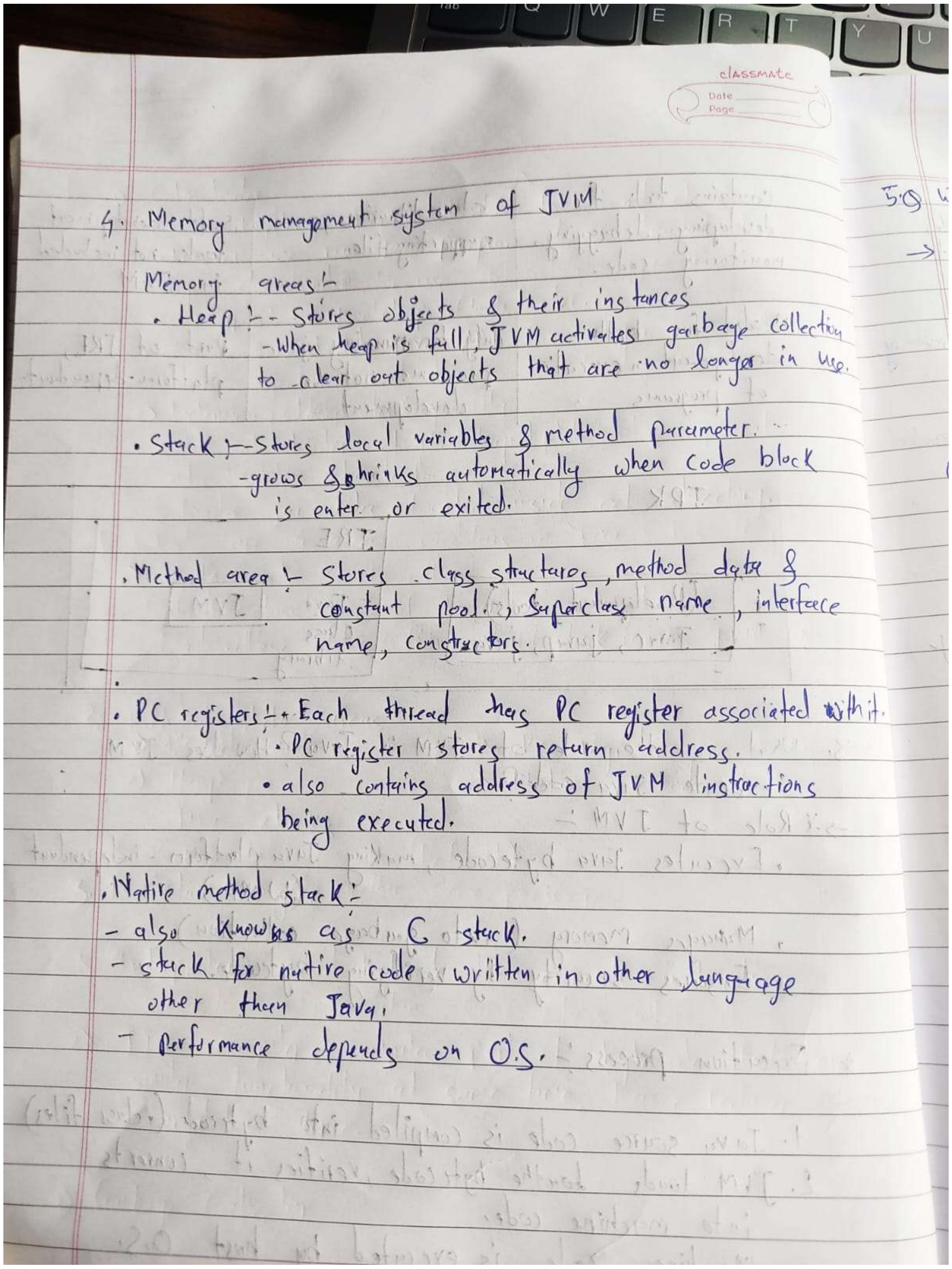
→ * Role of JVM :-

- Executes Java bytecode, making Java platform-independent (WORA)
- Manages memory (auto garbage collection)
- Ensures security by verifying bytecode.

* Execution process :-

1. Java source code is compiled into bytecode (.class files)
2. JVM loads ~~to~~ the bytecode, verifies it, converts into machine code.
3. Machine code is executed by host O.S.

4.Explain the memory management system of the JVM.

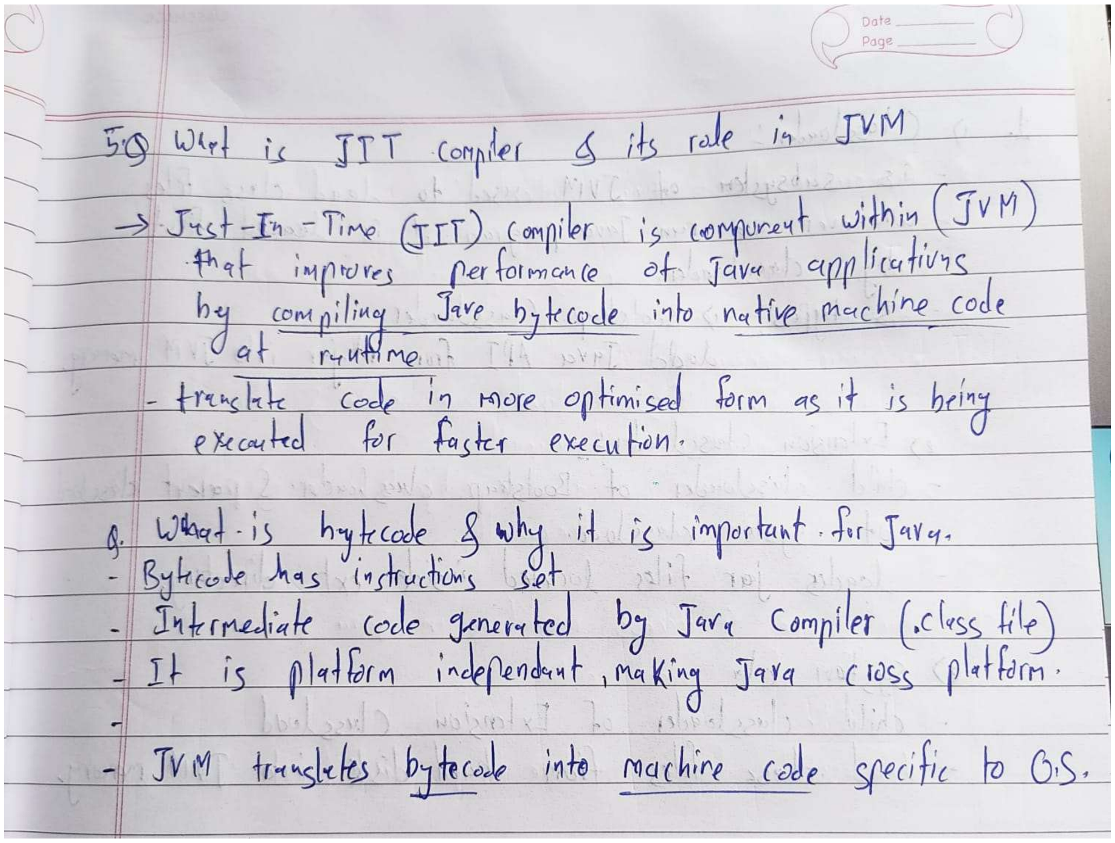


4. Memory management system of JVM

Memory areas :-

- Heap :- - Stores objects & their instances
  - When heap is full, JVM activates garbage collection to clear out objects that are no longer in use.
- Stack :- - Stores local variables & method parameter.
  - grows & shrinks automatically when code block is enter or exited.
- Method area :- Stores class structures, method data & constant pool, Superclass name, interface name, constructors.
- PC registers :- • Each thread has PC register associated with it.
  - PC register stores return address.
  - also contains address of JVM instructions being executed.
- Native method stack :-
  - also knows as C stack.
  - stack for native code written in other language other than Java.
  - performance depends on O.S.

5.What are the JIT compiler and its role in the JVM? What is the bytecode and why is it important for Java?

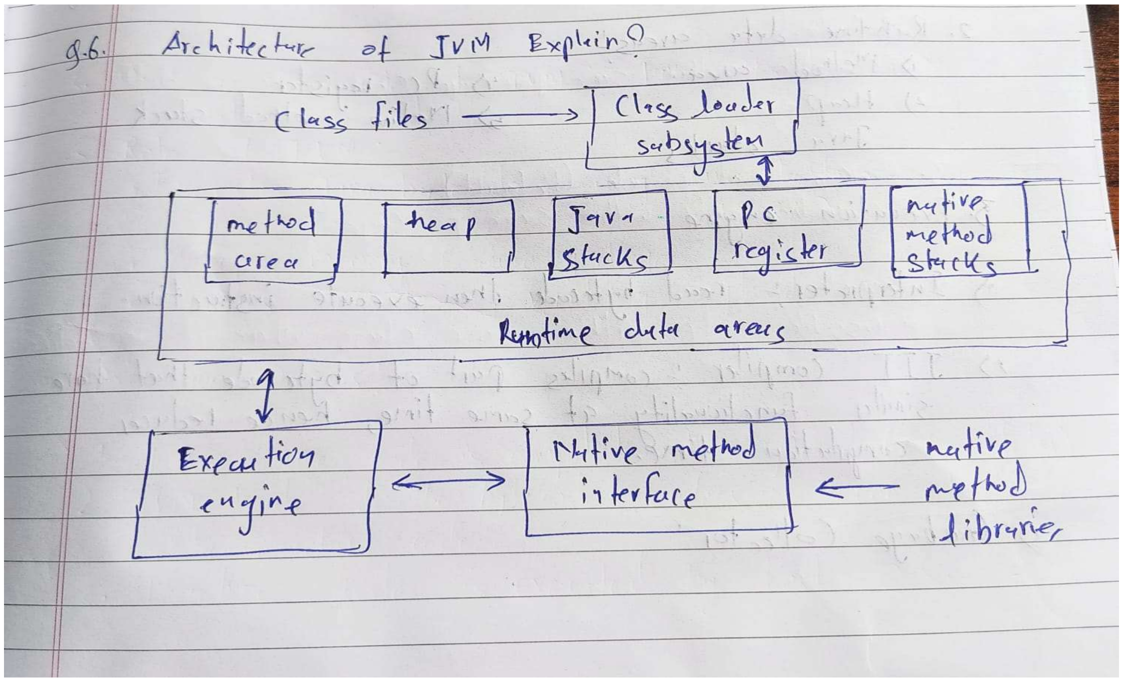

Date
Page

5Q What is JIT compiler & its role in JVM

→ Just-In-Time (JIT) compiler is component within (JVM) that improves performance of Java applications by compiling Java bytecode into native machine code at runtime.
- translate code in more optimised form as it is being executed for faster execution.

Q. What is bytecode & why it is important for Java.
- Bytecode has instructions set
- Intermediate code generated by Java Compiler (.class file)
- It is platform independant, making Java cross platform.
-
- JVM translates bytecode into machine code specific to O.S.

6. Describe the architecture of the JVM


Q.6. Architecture of JVM Explain?

1. 1) Classloader :-
- is subsystem of JVM used to load class files
- whenever we run Java program it is loaded first by classloader
- 3 types :- 1) Bootstrap Classloader,
lodd Java API from rt.jar into JVM memory.

2) Extension classloader :-
- child classloader of Bootstrap classloader & parent classload of system classload.
- loades jar files located inside ext directory.

3) System cla
- child classloader of Extension Classload
- load .class files from hard disk into JVM memory.

2. Runtime data areas :-
1) Method area
2) Heap
Java stacks
4) PC register
5) Native method stack

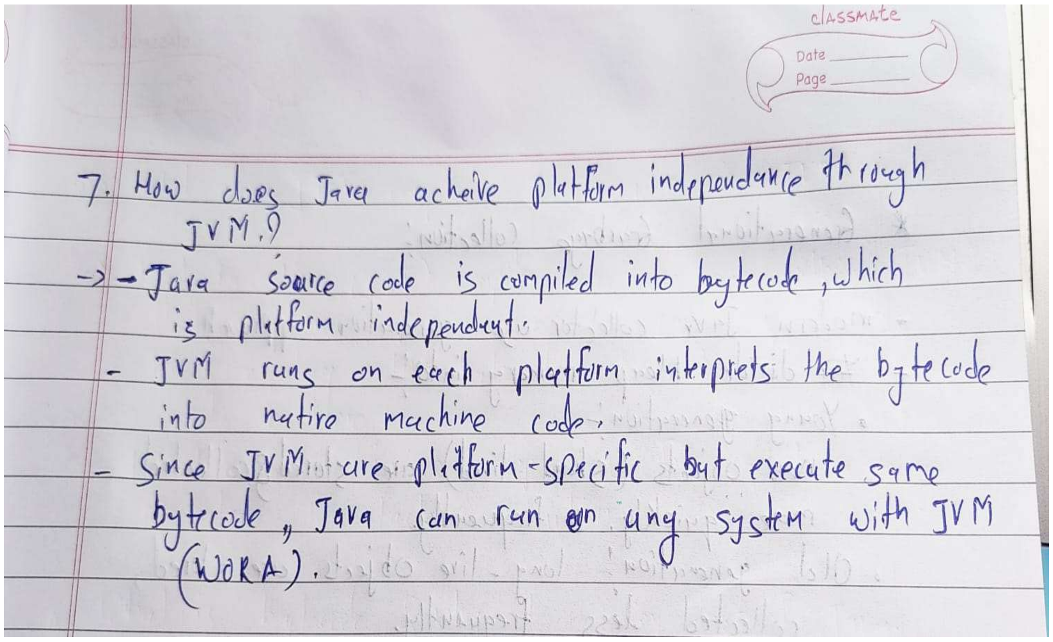
3. Execution engine

1) Interpreter :- read bytecode then execute instruction.

2) JIT compiler :- compiles part of bytecode that have similar functionality at same time, hence reduces compilation time.

3) Garbage Collector

7.How does Java achieve platform independence through the JVM?

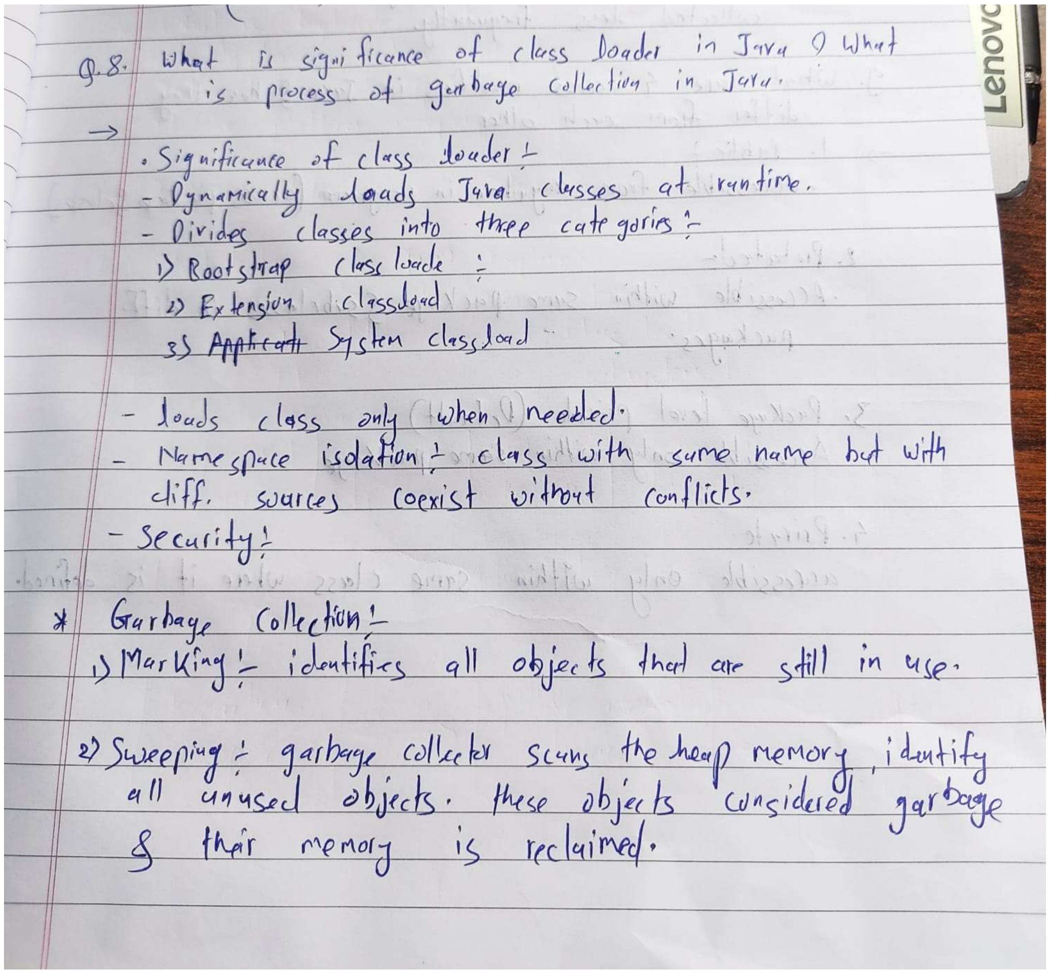


classmate
Date
Page

7. How does Java acheive platform independance through JVM.?

-> - Java source code is compiled into bytecode, which is platform independent.
- JVM runs on each platform interprets the bytecode into native machine code.
- Since JVM are platform-specific but execute same bytecode, Java can run on any system with JVM (WORA).

8.What is the significance of the class loader in Java? What is the process of garbage collection in Java.?



Q.8. What is significance of class Loader in Java ? What is process of garbage collection in Java.

→

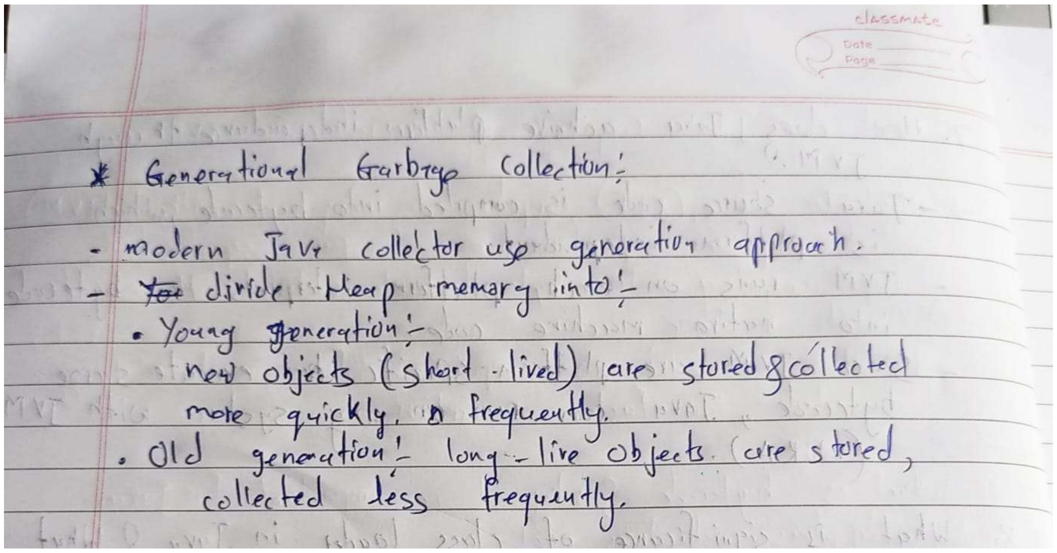
• Significance of class loader :-

- Dynamically loads Java classes at runtime.
- Divides classes into three categories :-
  1) Rootstrap class loade :
  2) Extension classload
  3) ~~Applicat~~ System classload
- loads class only when needed.
- Namespace isolation :- class with same name but with diff. sources coexist without conflicts.
- Security :

* Garbage Collection :-

1) Marking :- identifies all objects that are still in use.

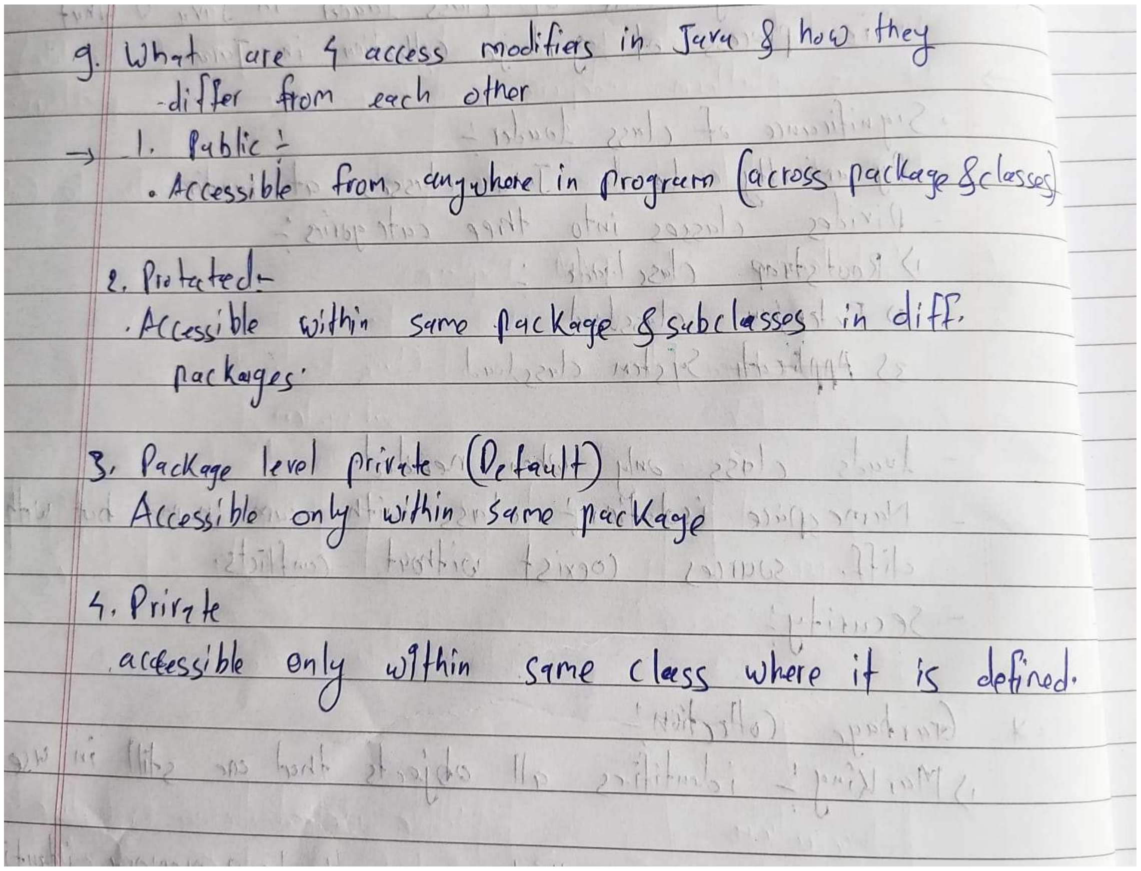
2) Sweeping :- garbage collecter scans the heap memory, identify all unused objects. these objects considered garbage & their memory is reclaimed.



* Generational Garbage collection :

- modern Java collector use generation approach.
- ~~To~~ divide Heap memory into :-
  • Young ~~g~~ generation :-
    new objects (short lived) are stored & collected more quickly, & frequently.
  • Old generation :- long - live objects are stored, collected less frequently.

9)What are the four access modifiers in Java, and how do they differ from each other?



9. What are 4 access modifiers in Java & how they differ from each other

→ 1. Public :-
- Accessible from anywhere in program (across package & classes)

2. Protected:-
- Accessible within same package & subclasses in diff. packages.

3. Package level private (Default)
- Accessible only within same package

4. Private
- accessible only within same class where it is defined.