Assignment - 3

1. Explain the components of JDK

Ans The Java Development Kit is a software development environment that includes several components, including :

1) Java Runtime Environment - The JRE is a software platforms that runs Java source code. It acts as an intermediary between Java program & the OS, allowing Java Software to run on any operating system without modification.
2) Java Virtual Machine - The JVM is a software tool that creates a runtime environment for Java source code to run in. It's an interpreter that converts bytecode to machine-level anguages line by line.
3) Java Interpreter / loader - The interpreter / loader reads & executes Java programs.
4) Java Compiler (javac) - The compiler compiles programs into machine-readable bytecode.
5) Java Archive (jar) - The jar is similar to a zip file. (Compile code of Java API)
6) Java Debugger (jdb) - The jdb is a utility that allows you to debug Java programs in the command-line interface.
7) Documentation generator (Javadoc) - The Javadoc generateds documentation.
8) Other Utilities - javap, jstat, jstack, jmap, that These tools are used in monitoring & analyzing the performance of Java appln at runtime. The help in profiling & troubleshooting issues like memory leaks, CPU usage & thread dumps.

2 Difference between JDK, JVM & JRE

Feature	JVM	JRE	JDK
Defn	The engine that executes Java bytecode, providing a runtime environment for Java appln.	A package that includes the JVM & libraries (rt.jar) Java appln needed to run	A comprehensive toolkit for developing Java appln, including the JRE, compilers & tools.
Components	Bytecode interpreter, Just-In-Time (JIT) compiler, Garbage collector	JVM, core libraries & other components to support Java appln execution	JRE, javac compiler, tools for Java appln development (eg. debugger, javadoc)
Purpose	To provide a platform independent way of executing Java bytecode	To provide runtime environment necessary for executing Java applns.	To provide developers with the tools needed to develop, compile & debug Java applns.
Platform Dependency	Platform ~~Dependent~~ ~~i.e Diff~~ Independent	Platform Dependent	Platform Dependent
Implemen	JDK = JRE + Development tools.	JRE = JVM + Class libraries.	JVM = Provides a runtime environ

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

Ans Role of JVM in Java

- JVM is responsible for converting bytecode to machine specific code & is necessary in both JDK & JRE
- Memory Management : It handles memory allocation, garbage collection & cleanup.
- Platform Independence : By abstracting the underlying system, the JVM allows Java programs to be 'Write Once, Run Anywhere'.
- Security : It ensures safe execution of code through strict verification & sandboxing
- In addition, JVM can run programs that are written in other Prog. languages that have been converted to Java bytecode.

How the JVM Executes Java Code

1) Loading - The JVM loads compiled bytecode into memory.
2) Verification - It checks the bytecode for correctness & security.
3) Execution -
   - Interpretation - Bytecode is interpreted & executed line by line
   - JIT Compilation - Frequently executed bytecode is compiled to native machine code for ~~f~~master execution
4) Memory Management - The JVM automatically manages memory using garbage collection.

This process ensures java programs run efficiently & securely across different platforms.

4. Explain the memory management system of the JVM.

Ans The JVM manages memory through a struc-

tured system, divided into key areas:

1) Heap - The heap is the runtime data area from which memory for all class instances & arrays is allocated.

Components:

Young Generation - Holds newly created objects. Most objects are short-lived & collected by Minor GC.

Old Generation - Stores long-lived objects that survive multiple garbage collections.

2) Stack - Each thread has its own stack, storing local variables & method call information. Memory is managed ~~is~~ in a last-in, first-out manner.

3) Method Area (Metaspace) - Stores class-level data, such as method bytecode & class structure. Metaspace dynamically grows based on application needs.

4) Garbage Collection

Minor GC - Collects short-lived objects from the Young Generation.

Major GC - Collects long-lived objects from the Old Generation.

Full GC - Reclaims memory across the entire heap when necessary.

This system automates memory allocation & deallocation, optimizing performance & preventing memory leaks.

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

Ans JIT Compiler

The JIT compiler is a component of the JVM that improves the performance of Java applns by compiling bytecode into native machine code at runtime.

Role in the JVM:

- Performance Optimization - Instead of interpreting bytecode line by line, the JIT compiler translates it into machine code that can be directly executed by CPU. This reduces execution time & improves appln performance.
- Hotspot Optimization - The JIT compiler focuses on 'hot' code paths-sections of code that are frequently executed. It compiles these hot paths into native code, which results in faster execution.

Bytecode

Bytecode is an intermediate, platform-independent code generated by the java compiler (javac) from Java source code. It is stored in class files & executed by JVM

Characteristics: Platform-Independent, Compact & efficient.

Importance of Bytecode

- Platform Independence (Byte code allows Java to achieve it's Write once, Run anywhere promise. JVM can execute bytecode on any platform.
- Security
- Portability

6. Describe the architecture of the JVM.

Ans The architecture of the JVM is designed to provide a platform-independent execution environment of java programs. It consists of several key components that work together to load, verify, execute & manage java applications.

1) Class Loader Subsystem

Responsible for loading class files into the JVM. It reads the .class files from the disk, verifies them & prepares them for execution.

Components :

- Bootstrap Class Loader ⇒ Loads core Java classes from the JDK
- Extension Class Loader ⇒ Loads classes from the extension directories.
- Application Class Loader ⇒ Loads classes from the application classpath.

2) Runtime Data Areas

- Heap ⇒ The runtime area where all objects & class instances are stored. It is shared among all threads.
- Method Area (Metaspace) ⇒ Stores class-level data like method bytecodes, field data & static variables.
- Stack ⇒ Each thread has it's own stack, storing method call frames, local variables & partial results. It operates in a last-in, first-out manner.
- Program Counter (PC) Register ⇒ Each thread has a PC register that holds the address of the currently executing instruction.
- Native Method Stack ⇒ Manages native method calls (methods written in languages like c/c++)

3) Execution Engine

Executes the bytecode loaded into the JVM.

Components :

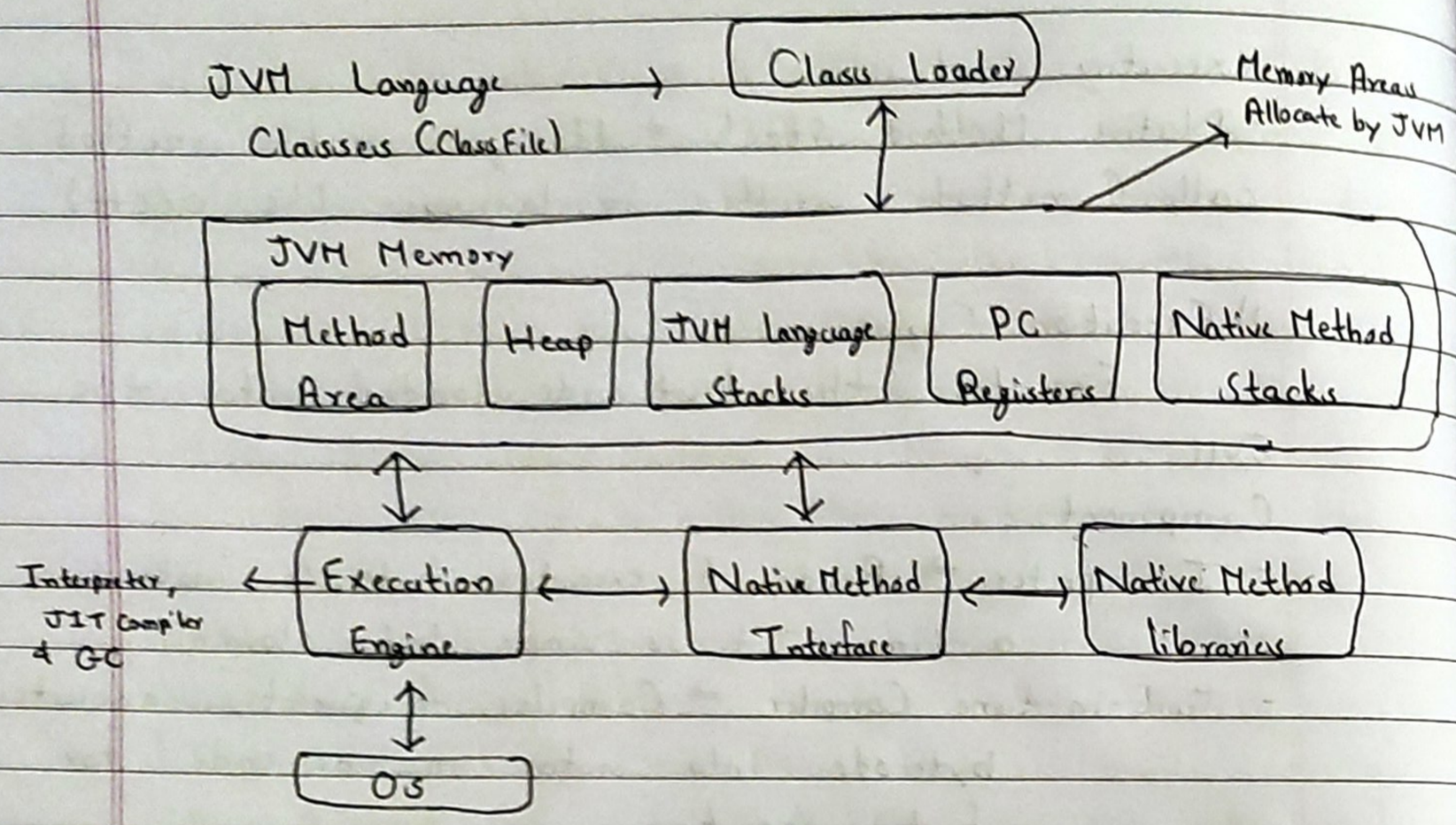
- Interpreter ⇒ Reads & executes bytecode instructions at a time. It is simple but slower.
- Just-in-time Compiler ⇒ Compiles frequently executed bytecode into native machine code for faster execution.
- Garbage Collector ⇒ Manages memory by automatically reclaiming memory used by objects that are no longer in use, preventing memory leaks.

4) Native Interface (JNI - Java Native Interface)

Allows the JVM to call & execute native code (written in lang like c/c++) from within java applns. This enables Java to interact with platform specific libraries.

5) Native Method Libraries

Libraries written in native languages that the JVM can load & use through the JNI. These are typically platform-specific & provide functionalities that are not natively available in Java.



Components of JVM Architecture

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

Ans Java achieves platform independence by using JVM as an intermediary between the Java program & the underlying operating system.

1) Compilation to Bytecode
- When you write & compile Java code, the Java Compiler (javac) converts the source code into bytecode, which is stored in class files.
- Bytecode is platform-independent, intermediate representation of code. It is not tied to any specific hardware or operating system.

2) JVM Execution on Any Platform
- The JVM is an abstract machine that exists for each platform (windows, macOS, Linux, etc) Each platform has its own specific JVM implementation.
- The bytecode generated from Java code is interpreted & executed by JVM. Since every platform has its own JVM, the same bytecode can run on any OS, provided a JVM is available for that system.

3) Write Once, Run Anywhere
- Because the bytecode is platform-independent & the JVM handles platform-specific details, Java programs can be written once & run on any device or OS that has a JVM.
- The JVM translates the bytecode into machine-specific instructions at runtime, making it possible for same bytecode to be executed on different platforms without modification.

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

Ans Significance of class loader

The class loader dynamically loads java classes into the JVM at runtime. It follows a delegation model, where it first asks its parent class loader to loader the class. The class loader ensures security by verifying bytecode & manages namespaces to prevent conflicts between classes with the same name.

Garbage Collection

Garbage Collection automatically frees memory by removing objects that are no longer referenced. It runs in the background, identifying unused objects & reclaiming their memory to prevent memory leaks & manage resources efficiently. The process involves making, sweeping & sometimes compacting the heap memory.

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

Ans 1) Default (Package Level Private)

When no access modifier is specified for a class, method or data member - It is said to be having default access modifier by default. The data members, classes or methods that are not declared using any access modifiers i.e. having default access modifiers are accessible only within same package.

2) Private

The private access modifier is specified using keyword private. The method or data members declared as private are accessible only within the class in which they are declared. Any other class of same package will not be able to access these members.

3) Protected

The protected access modifier is specified using the keyword protected. The method or data members ~~are~~ declared are accessible within the same package or ~~different~~ subclass in different packages.

4) Public

The public access modifier is specified using the keyword 'public. It has widest scope among all other access modifiers. Classes, methods or data members that are declared as public are accessible from everywhere in programs. There is no restriction on scope of public data.

10. What is DB public, protecteted, private & default access modifiers.

Ans Public - Access level of public modifie is everywhere. It can be accessed from within the class, outside the class, within the package & outside the package.

Protected - Access level of protected modifier is within the package & outside package ~~p~~ through the child class. If you do not make child class, it cannot be accessed from outside package.

Default - Access level of default modifier is only within package. It cannot be accessed from outside package. If you do not specify any access level, it will be default.

Private - Access level of private modifier is only within class. It cannot be accessed from outside the class.

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
default	Y	Y	N	N
private	Y	N	N	N

11. Can you override a method with a different ~~class~~ access modifier in a subclass? For eg. can a protected method in a superclass be overridden with a private method in a subclass? Explain.

Ans No, you cannot override a method with a more restrictive access modifiers in a subclass. In Java, when overriding a method, the access modifier in subclass must either be same or less restrictive than the method in superclass.

Eg.

If a superclass method is protected, it cannot be overridden with a private access modifier because private is more restrictive than protected. However, it can be overridden with a public method, which is less restrictive.

12. What is the difference between protected & default (package-private) access

Ans Protected

i) Visibility - Access within same package. Accessible in subclass (even if subclass is in different package)

2) The subclass can access the protected members of superclass, even if they are in a different package.

Default (Package-Private)

i) Visibility - Access only within the same package. Not accessible outside the package, even if a class is a subclass.

2) Even if a class is subclass, it cannot access default/package-private members from a superclass in another package.

13. Is it possible to make a class private in Java? If yes, where can it be done, what are limitation?

Ans In Java, is is not possible to make a top-level class private. However it's, possible to declare a nested (inner) class as private.

Where we can a class be made private

- Top level class ⇒ A top-level class (class declared ↓ within a package) cannot be declared private. It can only be public or default.
- Nested Class ⇒ An inner (nested) class declared within another class can be made private.

This means the nested class is only accessible from within outer class.

Limitations
- A private nested class is only accessible within the outer class that it is defined in. It cannot be instantiated or used directly by other classes, even if they are in same package.
- A top level class can never be private, so you cannot limit its access strictly to a single class

14. Can a top-lvel class in java be declared as protected or private? Why or why not?

Ans No, a top-level class in java be declared as protectated or private.
- Java is designed to so that top-level classes either have public access or default. Allowing a top-level class to be private & protected would defect it's purpose, as no other class would be able to access it.
- Public top-level classes are accessible from anywhere, even outside a package.
- Default top-level classes are accessible only within same package.
- Protected ~~top-level classes~~ allow access within same package & by subclasses, but top-level classes can't be subclass in the same way as inner class members. Hence protected has no meaning at top level.
- If a top-level class were private, it wouldn't be accessible by any other class, making it unstable outside of itself, which is illogical because class exist to be used by others.

Private & protected is meant for members (methods, fields & inner classes), not for top-level class.

15. What happens if you declare a variable or method as private in a class & try to access it from another class within the same package.

Ans If you declare a variable or method as private in class, it is only acccessible within that class. The private access modifier is most restrictive access level in Java & is used to encapsulate the details of a calss & prevent other class from accessing or modifying it's internal state.

You cannot access private variable or methods from another class, even if it is the same package. Private restricts visibility to the class itself, regardless of package boundaries.

16. Explain the concept of 'package-private' or 'default' access. How does it affect the visibility of class member?

PAGE No.
DATE / /

Ans - Package-private means that the class, method or field is accessible only within the same package.
- It restricts access to these members from classes that are outside the package.
- If you do not specify any access modifier, Java assumes package-private access.

Effect on class Members
1) Classes - A top-level class withe package-private access its accessible only within the same package. It cannot be accessed from classes in other packages.
2) Methods & Fields - Methods & fields of a class that are declared with package-private access can be accessed by other classes in the same package.
3) Access from Different Packages - Classes, methods or fields with package-private access are not accessible from classes in different packages.