CDAC Mumbai PG-DAC AUGUST 24
Assignment No- 3

Note: Write down this Interview questions & answers in your notebook .take a screenshorts ,make word file & upload on Github.

1) Explain the components of the JDK.

The **JDK (Java Development Kit)** is a software development environment used to develop Java applications. The key components include:

- **Java Compiler (javac):** Converts Java source code (.java files) into bytecode (.class files).
- **Java Runtime Environment (JRE):** Includes the JVM and libraries necessary to run Java programs.
- **Java Virtual Machine (JVM):** Executes Java bytecode and provides platform independence.
- **Java Debugger (jdb):** Used for debugging Java programs.
- **Java Archive Tool (jar):** Packages Java files into a .jar archive.
- **JavaDoc:** A tool to generate documentation from comments in the source code.

2) Differentiate between JDK, JVM, and JRE

- **JDK (Java Development Kit):** A full suite for developing, compiling, and running Java programs. It includes the JRE, a compiler, and other development tools.
- **JVM (Java Virtual Machine):** A virtual machine that runs Java bytecode. It is part of the JRE and provides platform independence by converting bytecode into machine code specific to the platform.
- **JRE (Java Runtime Environment):** Provides the environment needed to run Java applications. It includes the JVM and core libraries but does not include development tools like the JDK.

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

The **JVM** plays a critical role in executing Java code:

- **Loading:** The class loader loads Java bytecode (.class files).
- **Verification:** The bytecode is verified for security and correctness.
- **Execution:** The JVM executes the bytecode, either by interpreting it or by using the **JIT (Just-In-Time) compiler** to translate it into machine code for faster execution.

4) Explain the memory management system of the JVM.

The JVM memory is divided into several areas:

- **Heap:** Stores objects and instance variables.
- **Stack:** Holds local variables and partial results. Each thread has its own stack.
- **Method Area (or Permanent Generation in older versions):** Stores class structures and method code.
- **PC Registers:** Stores the address of the current instruction being executed.

- **Native Method Stack:** Contains the native methods used in the application.

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

The **JIT (Just-In-Time) compiler** is part of the JVM. It compiles bytecode into machine code at runtime for optimized performance, reducing interpretation overhead.

- **Bytecode:** It is the intermediate, platform-independent code generated by the Java compiler. It's important because the JVM can execute this code on any platform, achieving platform independence.

6) Describe the architecture of the JVM.

The JVM is composed of:

- **Class Loader Subsystem:** Loads class files into memory.
- **Runtime Data Areas:** Includes the heap, method area, stack, and more.
- **Execution Engine:** Interprets or compiles bytecode to machine code using the JIT compiler.
- **Garbage Collector:** Manages memory by freeing up objects that are no longer in use.

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

Java is platform-independent because the **Java compiler** converts source code into bytecode, which is platform-agnostic. This bytecode is executed by the JVM, which is platform-specific. The same bytecode can run on any platform that has a compatible JVM.

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

- **Class Loader:** Responsible for loading Java classes at runtime. It follows three principles:
  - **Delegation:** Requests are passed to the parent class loader.
  - **Visibility:** A child class loader can access classes loaded by the parent.
  - **Unloading:** Once classes are loaded, they can be unloaded by the garbage collector when not in use.
  - **Garbage Collection:** The JVM automatically frees memory by destroying objects that are no longer referenced, preventing memory leaks.

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

- **Public:** Accessible from any other class.
- **Protected:** Accessible within the same package or subclasses.
- **Default (Package-private):** Accessible only within the same package.
- **Private:** Accessible only within the class it is declared.

10) What is the difference between public, protected, and default access modifiers?

- **Public:** Available to all classes.
- **Protected:** Available to classes within the same package and subclasses.
- **Default (Package-private):** Available only to classes in the same package but not

subclasses.

11) Can you override a method with a different access modifier in a subclass? For example, can a protected method in a superclass be overridden with a private method in a subclass? Explain.

No, you cannot override a method with a **more restrictive** access modifier. A **protected** method in a superclass cannot be overridden by a **private** method in a subclass. However, it can be overridden by a method with **public** or **protected** access.

12) What is the difference between protected and default (package-private) access?

- **Protected:** Members are visible to classes within the same package and to subclasses, even if they are in different packages.
- **Default (Package-private):** Members are only visible within the same package but not to subclasses in different packages.

13) Is it possible to make a class private in Java? If yes, where can it be done, and what are the limitations?
No, top-level classes cannot be private. However, **inner classes** (classes within other classes) can be declared private. This limits their visibility only to the outer class.

14) Can a top-level class in Java be declared as protected or private? Why or why not?
No, a top-level class in Java cannot be **protected** or **private**. It can only have **public** or **default** (package-private) access, as top-level classes must be accessible from the package level.

15) What happens if you declare a variable or method as private in a class and try to access it from another class within the same package?
If you declare a variable or method as **private**, it is not accessible from any other class, even if the class is in the same package. It can only be accessed within the same class.

16) Explain the concept of "package-private" or "default" access. How does it affect the visibility of class members?
Package-private or **default** access means that a class, variable, or method is only accessible by other classes within the same package. It is the most restrictive access modifier that allows access across multiple classes. This is useful for grouping related classes and ensuring encapsulation within a package.