Rajshree
PAGE NO:
DATE:

Interview Question.

ASSIGNMENTS - 3

Q1. Explain the Components of the JDK.

Ans :- 1) Java Compiler (Javac): Converts Java source code into bytecode.
2) Java Runtime Environment (JRE): Provides the libraries, Java Virtual Machine (JVM) and other components to run application written in Java.
3) Java debugger: Helps in debugging java application.
4) Java Doc (javadoc): A tool for generating API documentation in HTML format from Java source code.
5) Java Disassembler (javap): Disassembles class files into readable bytecode.
6) Java Archiver (jar): Packages Java classes and metadata into a single archive file.

Q2 Differentiate b/w JDK, JVM and JRE.

Ans :- JDK (Java Development Kit): A full software development kit that includes the JRE an interpreter/loader (Java),

a Compiler (javac), an archiver (jar), a document ation generator (javadoc), and other tools needed for java development.

JVM (Java virtual Machine) :- This is an abstract machine that enables a computer to run java programs. It is responsible for converting bytecode into machine-specific code, making java platform-independent.

JRE (Java Runtime Environment) :- This is a software package that provides java class libraries, JVM, and other components required to run java applications. Unlike the JDK, it does not include development tools like compilers and debugger.

Q3 What is the role of the JVM in java? How does the JVM execute java code?

Ans → Role of JVM :-

The JVM plays a crucial role in executing java applications. It loads the bytecode, verifies it for security, interprets it, and translates it into machine code for execution. The JVM also manages system resources and memory through its garbage collection mechanism.

Rajshree
PAGE NO:
DATE:

⇒ Execution of Java Code :- The JVM executes Java Code in several stages:

1. Loading → the JVM loads the .class file containing the bytecode.

2. Verification → the bytecode is verified to ensure it adheres to Java's security rules.

3. Execution → the bytecode is either interpreted line-by-line or compiled into native machine code using the JIT Compiler.

4. Runtime → The JVM manages the execution of the program, including memory management, exception handling and system calls.

Q4 Explain the memory management system of the JVM.

Ans :- The JVM memory management system is divided into several region

1. Heap :- This is where objects are allocated the heap is further divided into generations :-
   → Young generation → Old generation

2. Stack :- Each thread has its own stack, storing local variables, method calls, and return values.

3. Method Area :- This area stores class structure such as runtime constant pool, field and method data and code for methods.

4. PC Register :- Holds the address of the currently executing instruction.

5. Native Method Stack :- Used for native method calls.

Q5 What are the JIT Compiler and its role in the JVM? What is bytecode and why is it important for Java?

Ans → JIT Compiler :- The just-In-time (JIT) Compiler is a part of the JVM that compiles bytecode into native machine code at runtime. This optimizes the execution of the code, making it faster.

Bytecode :- Bytecode is an intermediate code generated by the Java compiler

that is platform - independent. It can be executed on any platform with a compatible JVM, enabling Java's "write once; run anywhere" capability.

Q6 Describe the architecture of the JVM.

Ans:- JVM architecture consists of several comp.

1. Classloader :- loads class files into memory.
2. Memory Area :- Includes the heap, Stack, method area, and native method stack
3. Execution Engine :- Executes the loaded bytecod It consists of the interpreter, JIT Compile and garbage Collector
4. Native Method Interface (JNI) :- Allows the JVM to interact with native application written in languages like C/C++
5. Native Method libraries :- libraries that provide native functionalities required by the JVM.

Rajshree
PAGE NO:
DATE:

Q7 How does Java achieve platform independence through the JVM?

Ans :- Java achieves platform independence through the use of bytecode and the JVM. Java source code is compiled into bytecode, which is platform-independ. This bytecode is then executed by the JVM, which is platform-specific but interprets the bytecode in a consistent manner across different platform. As a result Java programs can run on any devices with a compatible JVM, regardless of the underlying hardware or operating system.

Q8: What is the significance of the class loader in Java? What is the process of garbage collection in Java?

Ans :- Class Loader :- The Class loader is responsible for dynamically loading Java classes into the JVM during runtime. It ensures that classes are loaded only once and handles the delegation model, where classes are loaded by parent loaders before being passed to

Child loaders.

Garbage Collection :- Garbage Collection in Java is the process of automatically freeing memory by reclaiming objects that are no longer in use. The JVM performs this process to ensure efficient memory management and prevent memory leaks. Java uses various garbage collection algorithms like Serial, Parallel, CMS and G1.

Q9 What are the four access modifiers in Java and how do they differ from each other?

Ans :- 1. Public :- The member is accessible from any other class.

2. Protected :- The member is accessible within its own package and by subclasses.

3. default (Package-private) :- The member is accessible only within its own package.

4. Private :- The member is accessible only within the class it is declared in.

Q10 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.

Ans :- No, you cannot override a method with a more restrictive access modifier in a Subclass. For instance, if a method is protected in a Superclass, it cannot be overridden with a private modifier in a Subclass because it would violate the liskov Substitution principle, which states that a Subclass should be Substitutable for its Superclass.

Q11 What is the difference between protected and default (package-private) access?

Ans :- Protected :- The member is accessible within the same package and by Subclasses in different packages.

default (Package-private) :- The member is accessible only within its own package and not by Subclasses outside the package.

Q12 It is possible to make a class private in Java? If yes, where can it be done, and what are the limitation?

Ans:- A class can be made private only if it is a nested class within another class. A top-level class cannot be declared as private. The limitation is that the private class is only accessible within the enclosing class.

Q13 Can a top-level class in Java be declared as protected or private? why or why not?

Ans:- No, a top-level class in Java cannot be declared as protected or private. Top-level classes can only have public or default access. This restriction exists because protected and private access control is meant to restrict access to members within a class hierarchy or within a package, and applying them to a top-level class would contradict the intended visibility scope.

Q15 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?.

Ans :- If a variable or method is declared as private in a class, it cannot be accessed from another class, even if the other class is in the same package. Private members are only accessible within the class they are declared in.

Q16 Explain the Concept of "package-private" or "default" access? How does it affect the visibility of class members?

Ans :- Package-private (default) access means that class members are accessible only with the same package. This access level does not use any keyword; if no access modifier is specified, the member is package-private by default. This affects visibility by restricting access to the class members to other classes within the same package, while preventing access from classes outside the package.