# ✯ Interview Questions ✯

Blue Bird
PAGE NO:
DATE

## Assignment - 3

Q1. Explain the components of the JVM. JDK.

The Java Development kit (JDK) is a software development environment used for developing Java appl? and applets.

It includes the Java Runtime Environment (JRE), an interpreter/Loader (JAVA), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools needed in Java development.

1. Java Compiler (javac)

- Purpose: To convert Java source code into bytecode.

- Explanation: The Java compiler takes the .java files containing human-readable code and compiles them into .class files that contains bytecode. Bytecode is an intermediate representation that can be executed by the Java Virtual Machine (JVM).

2. Java Runtime Environment (JRE).

- Purpose: To provides the libraries, Java virtual Machine (JVM), and other components necessary to run application written in Java.

- Explanation: The JRE is a subset of JDK that allows for the execution of Java programs. It does not include tools for development such as Compiler. The key components of the JRE includes:
  - JVM
  - core libraries eg: (collections, I/O —, networking.)

3. Java Virtual Machine (JVM):
   An engine that provides a runtime to execute Java bytecode.

4. Java Standard Library: A collection of pre-written classes and interfaces that provide functionality such as data structures, networking, I/O, etc.

5. Java DebMgger (jdb): A tool to debug Java applications.

6. JavaDoc (javadoc): A documentation tool that generates API documentation from Java source code comments.

7. Java Archie Tool (jar): A tool for creating and managing JAR files, which

package multiple Java classes and resources into a single file.

Q.2 Differentiate between JDK, JVM, and JRE.

- JDK (Java Development kit): The full suite for Java development, including tools for compiling, running, and debugging Java programs. It contains both the JRE and development tools like the Java compiler.

- JVM (Java Virtual Machine): A part of the JRE, the JVM is the engine that runs Java bytecode on any platform. It translates bytecode into machine code specific to the hardware.

- JRE (Java Runtime Environment): A part of the JDK that includes the JVM and core libraries necessary to run Java applicⁿ. It does not include development tool like the compiler.

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

The JVM provides a platform-independent way of executing Java bytecode. It abstracts

away the hardware, allowing Java programs to run on any platform that has a compatible JVM,

How JVM executes Java code:

1. Class Loading: JVM loads class file from the disk into memory.
2. Byte code: JVM verifies the bytecode for security and correctness.
3. Execution: JVM interpretes the bytecode or compiles it into native machine code using the Just-In-time (JIT) compiler.
4. Memory Management: The JVM manages memory allocation and garbage collection for the program.

Q4. Explain the Memory Management system of JVM:

- JVM memory management system divides into several areas:

• Heap: The runtime area where objects are allocated. The heap is divided into the Young Generation (for new objects) & old Generation (for long-lived objects).

• Stack: Each thread in JVM has its own stack, which stores local variables, methods calls, and partial results.

- Method Area : stores class structures, method data, and constants.

- Program counter (PC) Register :-
  It holds the address of the current instruction being executed for each thread.

- Native Method stacks: Used for JAVA native (non-Java) method calls.

Garbage collection : The JVM automatically reclaims memory by identifying and disposing of objects that are no longer in use.

Q5. What are the JIT compiler and its role in the JVM? What is bytecode and why is it important for JAVA?

- JIT compiler : JIT is a part of JVM that improves performance of Java appl. by compiling bytecode into native machine code at runtime. This compiled code is cached for future execution, speeding up the program.

- Bytecode : Bytecode is the intermediate representation of Java code, generated by the Java compiler (Javac). It is a platform independent, allowing java programs to run on any

machine with a compatible JVM.

Q6. Describe the architecture of the JVM.

- The architecture consists of :-

- Class Loader Subsystem : Responding Responsible for loading, linking, and inticalzing classes.

- Runtime Data Areas : Includes heap, stack, method area, and PC register.

- Execution Engine : Contains the interpreter, JIT compiler, and garbage collector. It executes the bytecode.

- Native Method Interface : Allows Java code to interact with native appl^. written in languages like C or C++.

- Native Method Libraries : Contains native libraries required by the JVM.

Q7 How does Java achieve platform independence through the JVM?

- It achieves platform Independence through the use of bytecode. When a Java program is compiled, it is converted into bytecode,

Which is platform-independent. The JVM on each platform interprets this bytecode into the machine code specific to that platform, allowing the same Java program to run on different platforms without modification.

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

- The class loader is responsible for loading Java classes into memory when they are required by the appl^n. It also defines the scope of classes and ensures that classes are only loaded once.

Garbage Collection process: It is the process by which the JVM automatically frees memory by reclaiming occupied by object that are no longer in use. The JVM periodically checks for such objects and removes them, ensuring efficient memory management.

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

- The four access modifiers in Java are:

- Public: The member is accessible from any other class

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

- Package-private (Default) : The member is accessible only within its own package

- private : The member is accessible only within its own class.

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

-

- public : The public members accessible from anywhere.

- Protected : The protected members accessible within its own package and by subclasses.

- Package-private (Default): Default member accessible only within its own package.

Q.11. Can you override a method with a different access modifier in a subclass? for eg: can a protected method in a superclass be

overriden with a private method in a subclass?

-

No, we cannot override a method with a more restrictive access modifier in a subclass. for eg:
So, a protected method in a superclass cannot be overridden as private in a subclass. The overriding method must have the same or less restrictive access.

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

-

- Protected: Accessible within the same package and by subclasses in other packages.
- Package-private (Default): Accessible only within the same package, not by subclasses in other packages.

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

-

A top-level class cannot be declared as private in java. However, inner classes can be declared private. Private inner classes are only accessible within the enclosing class.

Q14. Can a top-level class in java be declared as protected or private? Why or why not?

No, a top-level class cannot be declared as protected or private. It can only be declared as public or with default access. This is because top-level classes need to be accessible to other classes, and restricting their access would make it impossible to instantiate them.

Q.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 a variable or method is declared as private, it cannot be accessed from another class, even if that class is in the same package. It is only accessible within the class where it is defined.

Q.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 members (variables, methods or

constructor) is accessible only to other classes within the same package. It is not accessible from classes in other packages, even if they are subclasses. This provides a level of encapsulation with the package, allowing package-level collaboration while preventing access from unrelated classes.