Name : Tejwal R. Dudhe

Date :
Page :

# Assignment

Interview Q&A.

1) Explain the component of JDK ?

→ The JDK (Java Development kit) is a complete Software development environment used for developing Java application or applet.

# JDK Typically includes :-

1. Java Development Tools
   - Javac, Java, javap, jstack, jdb etc.
2. Java API documentation.
3. Java API ~~Lib~~ Libraries
   - rt.jar ( byte code).
4. Execution Environment
   - Java virtual machine (JVM)
5. Code Samples
   - Src.zip ( source code )

2) Differentiate between JDK, JVM, and JRE.

→ JDK ( Java Development kit )

- It is a complete development environment used to build Java Application.
- It includes JRE, a compiler (javac), debugger and other tools necessary for development.

JVM ( Java virtual Machine)

- It is a virtual machine that runs bytecode and allows program to be platform independent.

- The JVM manages tasks like memory management and garbage collection, ensuring Java application run smoothly.

JRE (Java Runtime Environment)
- It is an environment to run Java Application.
- JRE includes the JVM and standard Java libraries but does not contain development tool like compiler.

In Summary.
- JDK = JRE + development Tools.
- JRE = JVM + Libraries needed to run Java code
- JVM = The engine that run Java bytecode and convert machine native code

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

→ Java virtual machine (JVM) is key component of Java platform that enables "write once, run anywhere" philosophy. Its main main role is to execute Java bytecode and convert into machine native code which makes Java application platform-independent.
- JVM manage memory allocation and garbage collection and prevents memory leaks.
- JVM is also perform optimization as it includes JIT (Just In Time) compilation, which improves runtime compilation.

Date :
Page :

# The process of how the JVM executes Java code involves following steps :

1. Compilation :
   The Java compiler (javac) compile the source code (.java files) into platform independent (.class file).

2. Class Loading :
   The JVMs classloader subsystem loads the bytecode into memory as needed. The classloader is responsible for finding, loading and linking the class files during program execution.

3. Bytecode verification :
   The JVMs Bytecode verifier checks the Bytecode to ensure it is valid and does not violate Java's Security Restrictions.

4. Execution :
   The JVM interprets the bytecode using either an Interpreter or a Just-In-Time (JIT) compiler.
   - Interpreter : Execute the bytecode line by line translating it into machine specific instruction.
   - JIT compiler : optimizes performance by compiling frequently executed bytecode into native machine code which directly executed by processor.

5. Memory Management and Garbage Collection :
The JVM manages memory using a Heap (for objects) and Stack (for method call). The Garbage collector automatically reclaims memory occupied by objects that are no longer in use.

6. Execution Engine :
The execution engines in JVM converts bytecode into native machine code and execute it.

4) Explain memory Management System of JVM?

→ The JVM's memory management System is responsible for efficiently allocating, managing and reclaiming memory for Java application. It is critical for performance and stability of Java program.

JVM memory Areas :

1. Heap memory :
- The heap is where all java object are stored during the program execution
- It is divided into two main parts.
  Young Generation & Old Generation.
- Young generation contains newly created objects
- Old generation contain long lived objects.

2. Stack Memory :
- The stack memory is used for storing method calls, local variables and references to objects in heap.

3. Metaspace
- Metaspace stores class metadata, method data and constants like string literals.

4. Program Counter Register :
- Each thread has its own PC register, which keep track of the address of next instruction to be executed.

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

→ The 'Just in Time Compiler' is critical part in JVM that improves the performance in Java application by converting bytecode into native machine code at runtime.

1. Performance Optimization :
The JIT compiler optimizes the execution of Java program by converting it frequently executed portion of bytecode into native machine code. This reduces overhead of interpreting the bytecode repeatedly resulting in fast execution.

2. Dynamic Compilation.
— Instead of compiling entire program at once, the JIT compiler compiles parts of bytecode that are frequently executed. This minimizing startup delays.

3. In line Expansion and Loop Unrolling.
It applies techniques like in lining method calls and loop unrolling, which reduces method invocation overhead and improves performance in repetive loops.

## Bytecode : It is an intermediate, platform-independent code generated by Java compiler (javac) when source code is compiled. Bytecode is stored in (.class) file and is executed by JVM.

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

→ Java achieves platform independence through the JVM by compiling source code into bytecode, which is platform - neutral. This bytecode can run on any machine that have a JVM regardless of underlying operating system or hardware. The JVM interprets and execute bytecode, allowing Java application to run consistenly across different platforms.

Date :
Page :

8) What is Significance of classLoader in Java ?

→ The classLoader in Java responsible for dynamically loading classes into JVM at runtime. It finds and loads required .class files and ensure that they are available for execution. It also handles linking, which includes verification, preparation and resolution of classes. This dynamic loading supports Java flexibility and modularity.

9) Process of Garbage Collection in Java ?

→ Garbage Collection in Java is automic process of reclaiming memory by identifying and removing objects that are no longer referenced by the program. The JVM uses algorithm like mark-and-sweep and Generational GC to freeup the memory in the heap.

10) What are four access modifiers in java and how they differs from each others ?

→ The four access modifiers in Java are :

1. public : Accesible from anywhere in program across all packages.
2. private : Accesible only within same class, not visible to other classes.
3. protected : Accesible within same package and by subclasses, even in different packages

Date :
Page :

4. Default : Accesible only within the same package. also known as package private access.

11) Can you override a method with a different access modifiers in Subclass ?
For ex : Can a protected method in a Superclass can be overriden with private method in Subclass?

→ No, you cannot override in a Subclass with more restrictive access modifies.
for ex : If a method in Superclass is protected. you cannot override it in a Subclass with private access modifies, as it would restrict access to the method further.

13) It is possible to make a class private in Java ? If yes, where can it be done and what are limitations

→ No, it is not possible to make top level class private in Java. However the inner classes (nested classes) can be declared private.
— The private inner class cannot be accessed from outside the outer class. This limits its visibility strictly to be enclosing class, making it useful for encapsulation helps functionality that should not be exposed outside the outer class.

14) Can a Top level class in Java be declared as protected or private? Why or why not?

→ No, top level class in java cannot be declared as protected or private, Access modifiers like Protected and private are meant to control member level access within the class or package. Top level classes can only be declared as public or with default access ensuring they are visible within the package or to world respectively.

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

→ If you declare variable or method as private in a class, it cannot be accessed from another class within the same package. Private member are only accesible within the class where they are declared.

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 the classes member are visible only within the same package. It restricts the access from outside the package but allows unrestricted access by other classes within the same package.