Page No
Date

Assignment 2 012_Amol_Sonawane_KH

Q1) Explain the Components of SDK.

→ The Java Development Kit is set of tools used for developing Java applications.

⇒ It includes following Components:

1) JRE (Java Runtime Environment): The JRE is a collection of libraries and Components that provide the runtime environment for Java applications to execute. JRE also contains core libraries, such as Java API.

2) Java Compiler (Javac): This is the tool used to compile Java Source code (.java files) into bytecode (.class files), which can be executed by the JVM. The compiler ensures that the code adheres to the Syntax and semantics of the Java language.

3) Java virtual Machine (JVM): The JVM is the runtime engine responsible for executing Java bytecode. It translates bytecode instructions into machine code that can be understood by the underlying hardware.

4) Java Development tools: JDK includes various development tools to aid programmers in creating, debugging and optimizing Java code/application.
   i) Java Debugger (JDB)
   ii) Java Archive (JAR)
   iii) JavaDOC
   iv) JavaFx Compiler and Runtime.

5) API Documentation: The JDK includes comprehensive documentation for the Java API, which consists of thousands of classes and interfaces organized into packages.

Q2) Differentiate between JDK, JVM & JRE.

Ans: 1) JDK (Java Development Kit)

→ The JDK is a software development kit used by developers to create Java applications.

→ It includes tools as the Java compiler (javac), Java Archive (JAR) utility, and debugging tools.

→ The JDK also contains the Java Runtime Environment (JRE), allowing developers to compile and run Java programs.

→ In summary, the JDK is a complete package that includes everything needed for Java development.

2) JRE (Java Runtime Environment)

→ The JRE is a runtime environment that is necessary to run Java applications.

→ It includes the Java Virtual Machine (JVM) and core libraries required for executing Java bytecode.

→ While the JRE does not include development tools like compilers or debuggers, it provides the essential runtime components for running Java programs.

→ We only need JRE on a machine to run Java Application

3) JVM (Java Virtual Machine)

→ The JVM is a virtual machine that provides the runtime environment for executing Java bytecode.

→ It interprets Java bytecode and translates it into machine code that can be executed by the underlying hardware.

→ The JVM also manages memory allocation, garbage collection and exception handling during program execution.

* Both JDK and JRE include a JVM, but the JDK also includes development tools, while the JRE only includes the runtime environment.

Q3) What is the role of the JVM in Java? & How does the JVM execute Java code?

Ans The Java Virtual Machine plays a crucial role in the Java ecosystem. Its primary functions include executing Java bytecode, managing memory allocation, handling garbage collection and providing runtime environment support for Java applications.

→ Java code is compiled by Java compiler (Javac)
→ into bytecode which is stored in .class files.
→ Bytecode is platform independent.
→ Verification is done by JVM before execution. It checks for things like proper use of access modifiers, type safety and adherence to java language specifications.
→ Execution: Once the bytecode is loaded & verified JVM employs an interpreter to interpret

bytecode and execute the instructions.

→ Optimization: During execution, the JVM optimizes the code based on runtime profiling information. It identifies hotspots and applies various optimization to improve performance.

→ Garbage Collection: The JVM manages memory allocation and deallocation through automatic garbage collection.

→ Exception Handling: The JVM handles exceptions that occur during the execution of Java code. It provides mechanisms for catching and handling exceptions, allowing programs to gracefully recover from errors.

→ Termination: Once the Java program completes execution or encounters an unrecoverable error, the JVM terminates, releasing an allocated resources.

Q4 Explain Memory Management in JVM.

Ans. The memory management system of the JVM is responsible for allocating memory to Java objects and reclaiming memory that is no longer in use. It consists of:

1) Heap Memory: Heap memory is used to store states of ~~variables~~ or field variables.
2) Heap Memory does not have its own name so a reference is created, which is stored in stack.
3) Methods are stored & are allocated space in Method ~~Space~~ Area.
4) Garbage Collection: Process of reclaiming memory occupied by unused objects. ~~It includes~~ memory ~~Update~~.

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

Ans: Just-In-Time (JIT) Compiler is a crucial component of the Java Virtual Machine (JVM). Its primary role is to improve the performance of Java applications by dynamically translating Java bytecode into native machine code that can be executed directly by the host CPU.

→ When a Java program is executed, the JVM initially interprets the bytecode instructions. However, interpreting bytecode can be slow, especially for frequently executed code paths.

→ To overcome this performance bottleneck, the JVM employs the JIT Compiler, which identifies "hot spots" in the code - sections of bytecode that are executed frequently.

→ The JIT Compiler then translates these hot spots into native machine code, which is optimized for the specific hardware architecture of the host system.

→ This native code can execute much faster than interpreted bytecode, resulting in significant performance improvements for the Java application.

2) Role of JVM

→ The JIT Compiler plays a crucial role in balancing between the portability of Java bytecode and the performance of native machine code execution.

→ By dynamically compiling bytecode into native code at runtime, the JVM can achieve high performance while retaining platform independence

→ The JIT Compiler adapts to the runtime behaviour of the application, optimizing frequently executed code paths and reducing the overhead of interpretation.

→ It also allows Java applications to take advantage of the latest hardware features and optimizations without requiring modifications to the source code.

* Bytecode

→ Bytecode is the intermediate representation of Java source code. When you compile a Java program, the Java compiler converts the source code into bytecode.

→ Bytecode is platform-independent, meaning it can be executed on any system that has a compatible JVM installed, regardless of the underlying hardware and operating system.

→ Java bytecode is designed to be compact and efficient, allowing for faster transmission over networks and quicker loading time.

→ It serves as a "universal language" for Java applications, enabling developers to write code once and run it anywhere without the need for recompilation.

→ Bytecode is also crucial for security as it allows the JVM to enforce various runtime checks and access controls to prevent malicious behaviour.

Q6) Describe the architecture of JVM?

Ans6) The architecture of the Java Virtual Machine consists of several key components:

1) Class loader subsystem: Responsible for loading classes into memory.
   - It consists of:
     i) Bootstrap class loader
     ii) Extension class loader
     iii) Application class loader.

2) Runtime Data Areas: Memory areas used during the execution of a Java application.
   i) Method Area: Stores class metadata, static fields and constant pool.
   ii) Heap: Runtime data area where objects are allocated.
   iii) Java stack: Stores ~~method~~ reference of instance.
   iv) PC Register: Keeps track of the program counter for each thread.
   v) Native Method Stack: Stores native method invocations for each thread.

3) Execution Engine: Responsible for executing Java bytecode. It consists of
   → i) Interpreter
   → ii) Just-In-Time (JIT) compilers

4) Garbage Collector:
5) Native Method Interface
6) Execution Monitoring and profiling.
7) Security Manager:

Page No.
Date

Q7) How does Java achieve platform independence through JVM?

Ans Java is platform independent because it is compiled to a bytecode that can be run on any device that has a Java Virtual Machine (JVM). This means that you can write a Java program on one platform (such as windows) and then run it on a different platform (such as macOS or Linux) without making any changes to the code. The JVM acts as an interpreter for the Java bytecode, translating it into instructions that the host machine can understand and execute. This means that the same Java program can run on any device that has a JVM, making it a truly write once, run anywhere language.

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

- Class Loader

Ans8) Class loader dynamically loads classes into memory during runtime. Manages classpath, creating separate namespaces for classes, and enforcing security. Enables customization and extensibility through custom class loader implementations.

- Garbage Collection Process:
- Objects are allocated memory on the heap. Automatic memory management tracks reference, unreachable objects become

eligible for garbage collection. Garbage collection triggered by heap usage or JVM's discretion. Mark-and-Sweep algorithm identifies and reclaims memory occupied by unreachable objects. Some collectors perform to reduce fragmentation. Generational collection divides heap into different generations for optimized collection. Finalization, if defined, allows objects to perform cleanup before being reclaimed.