1. Explain the components of the JDK

Ans → Components of JDK:

(i) javac - The java compiler, which translates java source code into bytecode.

(ii) java - Executes bytecode on the JVM

(iii) javap - used to examine the bytecode instructions of compiled java classes.

(iv) jar - used to package java classes & related resources into a single file for distribution.

(v) javadoc - generates HTML documentation from java source code.

(vi) jdb - allows developers to debug java appl's.

2. Differentiate b/w JDK, JVM & JRE.

Ans → (i) JDK: • Includes tools for developing java appl's.
• It contains the compiler, runtime environment, and other development tools.

(ii) JVM: • Its an abstract computing machine that provides the runtime environment in which java bytecode can be executed. It is responsible for converting into machine code and managing memory.

(iii) JRE: • Its a part of JDK & provides the runtime environment for java appl's.
• It includes the JVM, class libraries & other runtime components necessary to run java applications.

3. what is the role of the JVM in Java? & How does the JVM execute Java code?

Ans → (i) The Java is responsible for executing Java bytecode, making Java platform independent.

(ii) It loads bytecode, verifies its integrity, and then executes it line by line using Just-in-time (JIT) compiler.

(iii) The JVM manages memory allocation & garbage collection, ensuring efficient memory usage.

(iv) It provides a secure execution environment by enforcing Java's security restrictions.

4. Explain the memory management system of the JVM.

Ans → (i) The JVM divides memory into various regions, including the heap, method area, stack & native method stacks.

(ii) The heap is used for dynamic memory allocations where objects are stored. It is managed by the garbage collector.

(iii) The method area ~~stored~~ stores class structures, method information, and constant pool data.

(iv) The method area stores class structures, method information, and constant pool data.

(v) The stack is used for storing invocation and local variables. Each thread has own stack.

(vi) Native method stacks are used for native method or methods written in other languages

4. ~~JIT Compiler & Bytecode Gan~~

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

Ans → (i) JIT (just-in-time) compiler:

It is a component of the JVM that compiles bytecode into machine code at runtime for improved performance.

(ii) Bytecode:

It is the intermediate as it allows Java programs to be executed on any devices with a JVM installed, regardless of the underlying hardware and operating system.

6. Describe the architecture of the JVM.

Ans → (i) The JVM architecture consists of various components, including class loader, runtime data areas (heap, method areas, stack), execution engine, and native method interface (JNI).

(ii) The class loader load class files into memory dynamically.

(iii) Runtime data areas manage memory allocation and storage for JVM execution.

(iv) The execution engine executes bytecode instructions, including interpretation & JIT compilation.

(v) The native method interface allows Java code to call native code written in other languages.

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

Ans ⇒ (i) Java achieves platform independence by compiling source code into bytecode, which is executed by the JVM.

(ii) The JVM provides an abstraction layer between Java bytecode and the underlying hardware and operating system, ensuring that Java programs can run on any platform with a compatible JVM installed.

~~8. Significance~~

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

Ans → Class loader:

(i) It is responsible for loading classes into the JVM dynamically.

(ii) It helps in achieving Java's dynamic extensibility and code reusability.

Garbage collection:

(i) It is the process of reclaiming memory occupied by objects that are no longer in use.

(ii) Java's garbage collector automatically manages memory allocation and deallocation, preventing memory leaks and improving memory efficiency.