05 September 2024 12:23

1) Explain the components of the JDK.

JRE (Java Runtime Environment): This is the environment needed to run Java programs. It includes the JVM, core libraries, and other files that support Java execution.
Compiler (javac): It converts Java source code (.java files) into bytecode (.class files) which can be run by the JVM.
Debugger (jdb): Helps in debugging Java programs to fix errors.
JavaDoc: A tool to generate API documentation in HTML format from Java source code.
Java tools: Additional utilities like JAR (Java Archive), javap (Java class disassembler), etc.

2) Differentiate between JDK, JVM, and JRE.

JDK (Java Development Kit): A package for developers that includes the JRE, a compiler, and other tools to develop Java applications.
JVM (Java Virtual Machine): The runtime engine that runs Java bytecode (compiled Java programs). It provides platform independence by interpreting the bytecode.
JRE (Java Runtime Environment): Provides the libraries and JVM required to run Java programs. It doesn't include development tools like the compiler.

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

The JVM is responsible for executing Java bytecode. It reads the compiled bytecode and translates it into machine-specific instructions.
When you run a Java program, the JVM:

1. Loads the .class file (bytecode).
2. Verifies the bytecode.
3. Executes the bytecode, translating it into machine code specific to your operating system.

4) Explain the memory management system of the JVM.

Heap: Where objects are stored. It is managed by the garbage collector.
Stack: Stores local variables and method calls.
Method Area: Stores class-level data like field, method data, and the bytecode of methods.
PC Register: Keeps track of which instruction is being executed.
Native Method Stack: Used for native methods (methods written in languages

like C/C++).

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

JIT (Just-In-Time) Compiler: A part of the JVM that improves performance by compiling bytecode into native machine code at runtime, so it doesn't need to interpret the bytecode every time.
Bytecode: This is an intermediate code generated by the Java compiler from source code. It is important because it allows Java to be platform-independent; the bytecode can run on any system with a JVM.

6) Describe the architecture of the JVM.

Class loader: Loads class files.
Memory Areas (Heap, Stack, Method Area): Manages memory allocation.
Execution Engine: Executes the bytecode.

- **Interpreter:** Interprets bytecode one instruction at a time.
- **JIT Compiler:** Optimizes bytecode into machine code for faster execution.

Garbage Collector: Frees memory by removing objects that are no longer in use.

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

Java achieves platform independence through its **bytecode** and the JVM. The Java compiler converts source code into platform-independent bytecode. Any system with a JVM can execute the bytecode, making Java programs runnable on different operating systems without modification.

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

Class Loader: It is part of the JVM that loads classes into memory when required. It handles dynamic loading, linking, and initialization of classes.
Garbage Collection: This is the process of automatically freeing memory by removing objects that are no longer referenced or needed by the program.

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

Public: Accessible from anywhere.
Private: Accessible only within the same class.
Protected: Accessible within the same package and subclasses (even in different packages).
Default (Package-private): Accessible only within the same package, but not outside the package.

10) what is the difference between public, protected, and default access

modifiers?

Public: The member can be accessed from any other class or package.
Protected: The member can be accessed within the same package and by subclasses outside the package.
Default: The member is accessible only within the same package (not outside).

11) 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.

We cannot override a method with a more restrictive access modifier in a subclass. For example, you cannot override a protected method in the superclass and make it private in the subclass. You can, however, make the access level more permissive (e.g., changing protected to public).

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

Protected: Allows access within the same package and to subclasses in other packages.
Default (Package-private): Only accessible within the same package, not from outside the package or subclasses in other packages.

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

You cannot make a top-level class private in Java. However, **inner classes** (nested within another class) can be private, but they will only be accessible within the enclosing class.

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

Top-level classes cannot be **protected** or **private**. They can only be public or have default (package-private) access. This is because a top-level class needs to be accessible from outside the class file.

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. Private access restricts access only to the class in which it is defined.

16) Explain the concept of "package-private" or "default" access. How does it affect the visibility of class members?

When a class member has default (package-private) access, it means that it can

only be accessed by other classes in the same package. It is not accessible from classes outside the package, even if they are subclasses.