

Assignment 3

Q1) Explain the components of JDK

- Ans Java Development Kit consists of following Key components:
- Java Compiler (javac): Compiles java source file into byte code.
 - Java Runtime Environment: Provides libraries and the JVM to run Java programs.
 - Java Virtual Machine: Executes Java bytecode providing platform independence.
 - Java Tools: Includes tools like javap, javadoc, jar.
 - Java Standard Library: A collection of packages like java.lang, java.util, java.io.

Q2) Difference between JDK, JVM, JRE

- Ans
- Java Development Kit (JDK): A full-featured software development kit for Java including JRE, compiler and tools required to develop Java applications.
 - Java Virtual Machine (JVM): A virtual machine that interprets and executes Java bytecode. It is platform-specific and is part of JRE.
 - Java Runtime Environment (JRE): Provides the runtime environment, including JVM and class libraries, for running Java applications.

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

Ans The JVM is responsible for interpreting Java bytecode and executing it on the

Underlying operating system, ensuring platform independence. The process of execution involves:

- 1) class loading: The class loader loads compiled class files into memory.
- 2) JVM: After class loader loads file JVM searches for main method and then starts executing the code inside main method.

Q4) Explain the memory management system of JVM?

Ans JVM memory is divided into several areas:

- Heap: Stores objects and class instances
- Stack: Stores method call frames, local variables and partial results.
- Method Area: Contains class-level data such as runtime constants and static variables.
- Program Counter Register: Points to current instruction in execution.
- Native Method Stack: Manager native code used by Java applications like C or C++ code.
- Garbage Collector: Memory management in JVM is handled primarily through Garbage collector which reclaims memory used by objects no longer referenced.

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

Ans The Just in time compiler converts bytecode into native machine code at runtime. Its role in JVM is it makes execution faster.

compared to interpretation because native code can run directly on hardware

Bytecode: Java source file is compiled into bytecode by Java compiler. Bytecode is important because it is platform-independent making Java "Write once, run anywhere"

Q6) Describe architecture of JVM

- Ans
- class loader: loads classes into memory
 - Memory Areas: Includes heap, stack, Method area, PC register, Native method stack
 - Execution Engine: Contains the interpreter and JIT compiler to execute bytecode
 - JNI: Allows integration with native code
 - Garbage collector: Manages automatic memory deallocation

Q7) How does Java achieve platform independence through JVM?

Ans Java achieves platform independence by compiling code into bytecode, which can be executed by the JVM on any platform. The JVM interprets or compiles the byte code into native machine code specific to host operating system, making the same Java program run on different platforms without modification.

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

Ans Class loader : Responsible for dynamically loading classes into JVM at runtime. It manages loading classes from different sources such as file systems, networks or JAR files.
Garbage collection : Automatically reclaims memory by deallocating objects that are no longer referenced.

Q9) What are four access modifiers in Java? and how do they differ from each other?

Ans Four access modifiers in Java :

- 1) Public : Accessible from anywhere
- 2) Private : Accessible only within same class
- 3) Protected : Accessible within same package and subclasses
- 4) Default : Accessible only within same package

Q10) What is difference between public, protected and default access modifiers?

Ans - Public : Accessible from any class in any package
- Protected : Accessible from classes in same package and subclasses even in different packages.
- Default : Accessible only to classes in same package

Q11) 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

Ans You cannot override a method with a more restrictive access modifier. A protected method in a superclass cannot be overridden as private in a subclass. However, you can make the overridden method more accessible. (Eg. from protected to public)

Q12) What is difference between protected and default access?

Ans

- Protected :- Accessible in the same package and by subclasses in other packages
- Default :- Only accessible within the same package and not in subclasses outside the package

Q13) Is it possible to make a class private in Java? If yes, where can it be done, and what are the limitations?

Ans Yes, it is possible to make class private. You can declare inner class as private within another class. However a top-level class cannot be private because it would be inaccessible to other classes, defeating its purpose.

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

Ans A top-level class cannot be declared as protected or private. It can only be declared as public or default access because the Java language specification does not allow restricting visibility of top-level classes to packages.

Q15) What happens if you declare a variable or method as private in a class and try to access it from another class within same package?

Ans If you declare a member variable or method as private in a class, it is inaccessible from other classes, even if they are in the same package. Trying to access it will result in a compilation error.

Q16) Explain the concept of "package-private" or "default" access. How does it affect the visibility of class members?

Ans Package-private or default access means that class members are only accessible to other classes within same package. It affects the visibility of class members as they are not visible or accessible from other packages.