Assignment 3

Kunal

1) Explain the components of the JDK

=> The Java development kit (JDK) is a comprehensive package that provides everything needed to develop Java applications.

Java Compiler :- ~~Java~~ Converts Java source code into bytecode which the JVM can execute.

Java Runtime Environment (JRE) :- provides the libraries, Java virtual Machine (JVM)

Java Debugger (jbb): A tool for debugging Java programs.

Java Documentation (javadoc): A tool that generate API documentation from Java source code comments.

Java Archie (jar): A utility to bundle multiple files into a single JAR file for easier distribution.

Java Libraries (rt.jar) : A set of pere compiled class libraries that provide standard functionalities for Java programs.

2) Differentiation between JDK, JVM & JRE

=> JDK (Java Development kit)
A full development environment that includes the JRE tools for developing Java applications. (compiler, debugger etc) & other utilities.

JVM (Java Virtual Machine) :-
A virtual machine that executes Java bytecode is executed.

JRE (Java Runtime Environment) :-
A subset of the JDK that includes the JVM, core libraries & other components needed to run Java applications.

3) Role of the JVM in Java and How it Executes Java code.

=> The JVM plays a critical role in running Java applications. It abstracts the underlying hardware & operating system details providing a platform-independent environment

Java Code Executions

1. Compilation :- Java source code is compiled into bytecode by the javac compiler.

2. Class Loading - The JVM loads the bytecode using the ClassLoader.

3. Bytecode verification :- The bytecode is verified for security & correctness

4. Execution : The JVM interprets the bytecode or uses the Just in Time

4) Memory Management System of JVM.

=> The JVM manages memory in Foll. region :-

- Heap : stories object & their associated data. It is divided into.
  - young gen. :- where new objects are allocated & aged.
  - old gen :- where long-serving objects are stored.

- Stack :- Each threads in Java has its own stack, storing method calls & local variables.

- Method Area :- Stores class structures, method data, & constant pool.

- Program Counter :- keeps track of the JVM instruction currently being executed.

- Native Method stack :- Used for native method calls via JNI (Java interface native)

5) JIT Compiler, Bytecode & Their importance.

=> JIT compiler (Just in time) :-
A part of the JVM that optimize bytecode execution by compiling it into native machine code by runtime, improving performance.

ByTecode :- A platform independent, intermediate representation of Java code that is executed by the JVM. Bytecode is crucial because it enable Java's "write once, run anywhere" capability.

6) Architecture of JVM

=> The JVM arch. consist of
- Class Loder Subsystem :- Loads, links & initialize classes during runtime.
- Runtime Data Areas :- Includes the method area, heap, stack, program Counter and native method stacks.
- Execution Engine : Consist of
  Interperter : Executes bytecode instructions directly.

- JIT compiler :- Compiles bytecode to native for performance optimization.

- garbage Collector :- Automatically manages by reclaiming memory used by objects no longer in use.

Native Method Interface :- Allows Java to interact with native applications written in other languages like C or C++.

7) Java's platform Independence through the JVM.

=> Java achieves platform independence through the JVM by compiling Java source code into platform independent bytecode. The JVM on each platform interprets this bytecode and executes it on the specific hardware ensuring that the same java program can run any platform without modification.

8) Significance of the Class Ladder & Garbage Collection in Java.

=> Class Loader : Responsible for dynamically loading Java classes into the JVM at runtime. It seperates the loading of classes into three phases : loading, linking and initialization. The class Loader also helps with namespace management & ensure classes are not loaded more than once.

- Garbage Collection :-
Java uses garbage collection to automatically manage memory. The JVM identifies objects that are no longer referenced by any part of the program & reclaim their memory preventing memory leaks & improving efficiency.

9) Four Access Modifiers in Java.

=> Public :- The member or class is accessible from any other class.
Protected :- The member is accessible within its own package and by subclasses.
Default :- The member is accessible only within its own package. It is the default level if no modifiers is specified.

private :- The member is accessible only within the class it is declared in.

10) Differences between Public, Protected and default Access Modifiers.

=> . Public : Accessible from any other class regardless of the package.
. Protected :- Accessible within the same package and by subclasses, even if they are in different package.

- Default :- Accessible only within the same package, not visible to classes in other package.

11) Overiding a Method with different Access Modifier.

=> You cannot override a method in a subclass with a more restrictive access modifier. For example a protected method in a superclass cannot be overridden with a private method in a subclass. However, you can override it with a method having protected or public access.

12) Difference between Protected & Default (Package-private) Access.

=> Protected :- Allows access within the same package and also to subclasses, even if they are in different packages.

(Package-Private)

Default :- Restricts access to classes within the same package. Subclasses in ~~packages~~ diff. packages cannot access members with def.-lt access.

13) Making a class Private in Java

=> You cannot declare a top-level class as private. However you can declare inner classes (classes within a class) as private. A private inner class is accessible only within its outer class.

14) Declaring a Top-level class as Protected or private.

=> A top level class in Java connot be declared as protected or private. Top level classes can only be public or have default access. This is because protected and private access would restrict the visibility of the class too severly, conflicting with Java's package-based access control.

15) Accessing Private Variable or Methods form Another Class in the same package.

=> If a variable or method is declared as private, it cannot be accessed from another class even if that class is within the same package. Private members are only accessible within the class they are declared in.

16) Concept of "Package-Private" or "Default" access

=) "Package-private" or "default" access in Java means that the member is accessible only within its own package. If no access modifier is specified, the member is treated as having default access. This restricts visibility to other classes in the same package and prevents access from classes in other packages.