PAGE No.
DATE / /

Assignment 3

1] Explain the components of the JDK.

Ans) The Java Development kit (JDK) is a complete software development environment used for developing Java applications. It consists of the following key components.

1] Java Compiler (javac):
Converts java source code (.java files) into bytecode (.class files) that can be executed by the JVM.

2] Java Runtime Environment (JRE):
Provides the libraries, Java Virtual Machine (JVM), and other components to run Java applications. The JRE is part of the JDK.

3] Java Virtual Machine (JVM):
The JVM executes Java bytecode on any platform. It is the engine that runs Java applications, providing platform independence

4] Java Standard Library:-
A large collection of pre-built classes and methods that developers can use to perform various tasks such as data manipulation, networking, and file handling.

5] Java Debugger:-
A tool that helps developers debug Java applications by providing functionalities like setting breakpoints stepping through code, and inspecting variables.

6] Java Doc (javadoc):
A tool that generates HTML documentation from Java source code comments.

PAGE No.
DATE / /

7] Java Archiver (jar):
A tool to package multiple files into a single archive file (JAR). JAR files are used to distrilade Java applications and libraries

2] Differentiate between JDK, JVM, and JRE
Ans 1] JDK (Java Development kit):
A complete environment for Java development, including the tools needed to compile, debug, and run Java applications. used by developers to write and compile Java Code.
2] JVM (Java Virtual machine):
A virtual machine that runs Java bytecode. It enables Java's platform independence by allowing the same bytecode to be executed on any platform with a compatible JVM.
It executes the bytecode generated by the Java Compiler.
3] JRE (Java runtime environment):
JRE is used to run Java applications, required by end-users to execute Java programs.

Q3] What is the Role of the JVM in Java? and How Does the JVM Execute Java Code?
Ans] Role of JVM in Java:
It's providing a Platform independent execution environment for Java applications.
JVM Executes Java Code by Loading it is used to ClassLoader subsystem to load .class files into memory.

2] Linking :- It verifies and links the loaded class files, ensuring they adhere to the Java language specifications.
3] Initialization : The JVM initializes the static variables and blocks of the class.
4] Execution:-
→ Interpreter:- Initially, the JVM interprets or run the bytecode, executing it line by line.
→ JIT Compilation:- Frequently executed code is compiled into native machine code by the Just-In-Time (JIT) compiler, improving performance.
5] Memory management : The JVM Garbage Collector automatically manages memory, freeing up space used by objects no longer in use.

Q 4] Explain the memory management system of the JVM
Ans] The JVM uses a sophisticated memory management system to handle the allocation and deallocation of memory for Java applications. Key components of this system include :-
1] Heap :- The Heap is where all Java objects are stored. It is divided into : Young Generation and old Generation.
2] Java stack : Each thread has its own stack, which stores method calls (frames). Each frame contains local variables, the operand stack, and method return values.
3] PC Register:- stores the address of the next instruction to be executed.
4] Method Area:- stores class structures and methods.

PAGE No.
DATE / /

Q5] Native method stack:
- used for native methods written in languages like C/C++. It manages the execution of native code.

Q5] What are the JIT compiler and its Role in the JVM? what is the Bytecode and why is it important for Java.

Ans] 1] JIT compiler:-
The Just in time compiler is part of the JVM's execution engine. It improves performance by compiling bytecode into native machine code at runtime, allowing the JVM to execute the compiled code directly on the hardware.

2] Role in JVM:-
The JIT compiler reduces the overhead of interpreting frequently executed bytecode by converting it to native code.

3] Bytecode:
Bytecode is the intermediate representation of Java code. It is a set of instruction that the JVM executes. Bytecode is plateform oriented and is stored in .class file generated by Java compiler.

4] Bytecode enables Java's platform independence. Since the JVM can execute bytecode on any platform.

PAGE No.
DATE / /

6] Describe the Architecture of the JVM.

Ans] 1] Class Loader subsystem:-
Responsible for loading, linking, and initializing classes and interfaces.

2] Runtime Data Areas:-

a] Method Area: stores per-class structures such as the runtime constant pool, field and method data.

b] Heap: stores all objects and arrays. Divided into Young Generation and old Generation.

c] Java stack: stores frames, which hold local variables operand stacks, and partial results for method invocations.

d] PC Register: stores the address of the current instruction for each thread.

e] Native method stack: Manages the execution of native methods.

3] Execution Engine:-

1] Interpreter: Reads and executes bytecode line by line.

2] JIT compiler: Compiles frequently executed bytecode into native machine code for faster execution.

3] Garbage collector:- Manages memory by reclaiming space used by objects no longer in use.

4] Native method Interface: can be run in another language

5] Java Native Libraries:

tenouk -

PAGE No.
DATE / /

7] Java achieve platform independence through the JVM

Ans] Java achieves platform independence by compiling source code into bytecode, which is executed by the JVM. The JVM translates bytecode into machine-specific instructions, allowing the same bytecode to run on any platform with a JVM.

8] Significance of class loader in Java? what is the process of garbage collection in Java?

Ans] The class loader loads Java class into memory at runtime. This allows dynamic loading of classes,

Garbage collection: Automatically frees memory by removing objects that are no longer in use thus preventing memory leaks.

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

Ans] Access modifiers

Public : Accessible from anywhere.

Protected : Accessible within the same package and subclasses.

Default : Accessible within the same package

Private : Accessible only within the same class

PAGE No.
DATE / /

10] What is the difference between public, protected and default access modifiers?

Ans] 'Public': Accessible everywhere.
Protected: Accessible in the same package and through inheritance.
Default: Accessible only within same package.

11] 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] No, 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 as private in a subclass. This is because method overriding requires that the overridden method must be accessible wherever the ~~sp~~ superclass method is accessible. So the access level of an overridden method must be the same or more permissive. However, you can override a protected method with a public one since it is more permissive.

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

Ans] protected: Accessible in the same package and subclasses (even in other packages.)
Default: Accessible only within the same package.

Scanned with OKEN Scanner

PAGE No.
DATE / /

13] Private Class in Java :- A top-level class cannot be private, but nested classes within a class can be private.

14] Top-Level Class Access Modifiers : A top-level class cannot be protected or private because classes must be accessible at least to their package.

16] Accessing Private Variables/methods :-
Ans] You cannot access private members from another class in the same package. Private members are strictly for internal class use.

15] Package-Private Access : If no access modifier is specified, the member is accessible only within the same package. This ensures controlled visibility for class within the same package.