4/1/24

classmate
Date
Page

Assignment No3

1) Explain the components of JDK.

Ans- JDK is comprehensive tool which has key components

1) Java compiler - converts java source code into bytecode which can be executed by the Java Virtual Machine (JVM)

2) JVM - executes java bytecode, providing runtime environment for Java application.

3) JRE (Java Runtime Environment) - Includes JVM & standard libraries necessary to run Java applications. It's an essential runtime portion of JDK

4) Java API libraries - A set of pre-written classes & interfaces that provide essential functionality such as data structures, networking & file I/O.

5) Java Debugger - A tool for java programs to debug, allowing developers to inspect & control the execution of their code.

6) Java documentation generator - Automatically generates documentation from Java source code used.

7) Archive (jar) - packages related class file into a single JAR which can be easily distributed & used.

8) Java Disassembler (javap) - disassembles class files, providing a human-readable version of bytecode.

9) Java web start - launches Java applications directly from the web.

2) Differentiate bet^n JDK, JVM & JRE.

Ans- JDK -

purpose - provides tools for developing Java applications

Components - javac, Java Debugger (jdb), java documentation generator (javadoc)

Usage - Used by developers to write, compile & debug Java programs.

2) Java Virtual Machine (JVM)
Purpose - Executes java bytecode, providing runtime environment
Components - Includes java interpreter & JIT compiler.
Usage - Runs java applications by converting bytecode into machine code specific to OS.

3) Java Runtime Environment (JRE)
purpose - provides libraries & JVM required to run java app.
Components - Includes JVM & core libraries but lacks dev. tools like the compiler & debuggers.
Usage - Used by end users to run java applications.

3) What is the role of the JVM in Java & How does JVM execute java code?
Ans - i) Platform Independence - JVM allows java programs to be written once & run anywhere (WORA). This means java code runs on any device OS of that has a compatible JVM.

2) Bytecode Execution - JVM executes java bytecode which is intermediate representation of java code compiled from source code. Bytecode is platform independent

3) Memory management - JVM handles memory allocation and garbage collection, ensuring efficient use of memory & preventing memory leaks.

4) Security - JVM provides a secure execution environment by verifying bytecode before execution, preventing malicious code from causing harm.

5) performance optimization
JVM includes JIT compiler that optimizes bytecode into native m/c code at runtime, improving performance.

JVM execution -

1) class loading - the classloader subsystem loads the .class files into memory. It reads bytecode & stores it in the method area.

2) Bytecode verification -
Bytecode verifier checks the loaded bytecode for validity and security. It ensures that the bytecode adheres to JVM's safety rules & doesn't perform illegal operations.

3) Execution - Execution engine is responsible for executing the bytecode. It can interpret the bytecode or JIT compiler to convert it into native m/c code for faster execution.

4) Runtime Data Areas -
JVM manages several memory areas during execution.
1) Method Area - stores class structures, including metadata method data & constants.
2) Heap - Allocates memory for objects & class instances.
3) Stack - Holds local variables & partial results
4) PC registers - keep track of the address of the currently executing instruction.
5) Native method stack - manages native method calls.

4) Explain memory management system of the JVM

Ans- JVM memory structure

1. Heap memory - young generation - this is where all new objects are created.

Eden space - Most new objects are allocated here.

Survivor spaces - two spaces where objects are that are survive garbage collection in the Eden space are moved.

2) Permanent Generation -

stores metadata required by the JVM, such as class definition and method information.

3) Stack memory

Each thread has its own stack, which stores local variables, method call information & references to objects in the heap.

Garbage collection

Minor GC - occurs in young generation. It is quick & frequent, collecting short-lived objects

Major GC - occurs in old generation. It is less frequent but takes longer, collecting long-lived objects.

stop the world events - during garbage collection, all application threads are paused which can affect performance.

5) JIT compiler and its Role in the JVM

Ans- JIT is crucial component of JVM & here it how works -

Compilation at Runtime - the JIT compiler converts Java bytecode into native machine code at runtime. this means that instead of interpreting bytecode line by line the JVM can execute native machine code directly which is much faster.

optimization - the JIT compiler performs various optimizations such as inlining methods and eliminating redundant code to improve execution speed.

Adaptive compilation - the JIT compiler monitors the execution of the application & compiles frequently executed methods, optimizing them for better performance.

6) Describe the architecture of JVM?

Ans- i) class loader - this part of JVM is like librarian. It finds & loads the Java class files into memory so they can be used by your program.

2) Runtime Data Areas - think of this as memory management system of the JVM. It includes
method area - stores information about the classes, like methods & variables.
Heap - this is where all the objects are created & stored.
Stack - Each thread has its own stack, which keeps track of method calls & local variables.
PC registers - these are like bookmarks for each thread, keeping track of which instruction is being executed.
Native method stack - Manages calls to native methods.

3) Execution engine.
1) Interpreter - Reads & executes the bytecode instruction one by one.
2) JIT compiler - converts bytecode into native m/c code for faster execution.
3) Garbage collector - Automatically clears up memory by removing objects that are no longer needed.

4) JNI - (Java native interface)
this is like a translator that allows Java to interact with code written in other languages like C or C++.

5) Native method libraries.
these are the actual libraries of native code that the JVM uses to execute native methods.

*) How does the Java achieve platform independence

Ans- It is achieved by JVM & following steps.

1) Compilation to Bytecode
when you write Java code, it gets compiled into an intermediate form called as Bytecode. It is set of instructions that is platform independent, meaning it can run on any system

2) JVM Interpretation
- Each OS has its own version of the JVM
- JVM reads the bytecode & translates it into m/c specific instructions that the host system can execute.

3) execution
- JVM executes translated instructions on the host m/c.
- this process ensures that the same Java program can run on a diff. platforms w/o modification.

5) what is the significance of class loader in Java?
what is the process of garbage collection in Java?

Ans - the class loader in Java is a crucial component of the Java Runtime Environment. It dynamically loads Java classes into the Java virtual machine (JVM) during runtime.

1) Dynamic loading - classes are loaded into memory only when they are needed, which helps in reducing the startup time of application.

2) Delegation model - class loaders follow a delegation hierarchy. When a class is requested, the class loader delegates the request to its parent class loader before attempting to load the class itself.

3) Visibility - classes loaded by a parent class loader are visible to child class loaders, but not vice versa. This ensures encapsulation & prevents conflicts bet^n classes loaded by different class loaders.

4) Uniqueness - A class is loaded only by a class loader, ensuring that the same class is not loaded multiple times.

Types of class loaders

1) Bootstrap class loader - loads core classes from the Java Runtime Environment (JRE)

2) Extension class loader - loads classes from the extensions directory

3) System/Application class loader - loads classes from the classpath defined for the application.

* Garbage collection in Java

It is done in 3 steps

1) Marking - the garbage collector identifies which objects are still in use & which are not. This is done by traversing object references starting from the root objects.

2) sweeping - the garbage collector removes objects that are no longer reachable from the root objects, freeing up memory.

3) compacting - optionally, garbage collectors can compact the memory by moving the remaining objects together to reduce fragmentation.

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

Ans- i) public - Most accessible, can be accessed from any other class.
2) protected - Accessible within the same package & by subclasses.
3) default - Accessible only within the same package
4) private - least accessible only within the same class.

10) What is the difference between public, protected & default access modifiers?

Ans- public - most accessible, can be accessed from any other class
protected - Accessible within the same package & subclasses.
default - accessible within same package.

11) Can you override a method with a different access modifier in a subclass? For ex. can a protected method in a subclass be overridden with a private method in a subclass?

Ans- In Java, when overriding a method in a subclass you can't use a more restrictive access modifier than the one used in the superclass.

A protected method in a superclass can't be overridden with a private method in a subclass.

- You can override a method with a less restrictive access modifier.

12) What is the diff. b/t protected and default access?

Ans- protected -
1) Accessible within the same package
2) Accessible by subclasses even if they are in different packages.
Usage - often used for methods & variables that should be accessible to subclasses but not to the entire world.

default (package-private) Access
1) Accessible only within the same package
2) Not accessible from subclasses in different packages.

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

Ans- yes but only when it is done with nested (inner) classes. It can declare as a nested class. This means the nested class is only accessible within the outer class.
Limitations - you can't declare a top-level class as a private if you try to do so, you will get compile-time error because a private top-level class would be completely inaccessible to any other class, making it useless.

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

Ans- No, a top-level class in Java can't be declared as protected or private. It can only have public or default.
Reasons are
1) private - If a top-level class were private. It would be completely inaccessible to any other class, making it unusable. The purpose of a private class is to restrict access but for a top-level class, this only means no other class could ever use it.

2) protected - the protected access modifier is designed to allow access within the same package & by subclasses. However, for a top class this doesn't make sense because it would imply that the class should be accessible to subclasses in other packages.

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

Ans → If you declare a variable or method as a private in a class, it means that variable or method is only accessible within the same class. It can't be accessed directly from any other class, even if those classes are in the same package.

key points -

- private access modifier - the private keyword restricts the visibility of the variable or method to the class in which it is declared.
- Encapsulation - this is a core principle of object-oriented programming ensuring that the internal state of an object is hidden from the outside world & can only be accessed thru public methods.

16) Explain the concept of 'package-private' or 'default' access.

Ans - 'package-private' is the access level assigned to class members when no explicit access modifier is specified. visibility - within the same package. members with package private access are accessible to all other classes outside the package - these members are not accessible to any class outside the package.