Sneha Motghare
085_ JUH21.

Page No.:
Date :

JAVA ASSIGNMENT.

1) Explain the Components of the JDK.

Ans- 1. Java Compiler (javac)
The Java compiler is a key component of JDK that transforms Java source code (.java files) into bytecode (.class files). The generated bytecode can be executed on any problem with JVM installed, ensuring the "write once, run anywhere" philosophy of Java

2. Java Virtual Machine (JVM): The Java Virtual machine is the runtime engine that executes Java bytecode. It provides an abstraction layer between the java application & the underlying system.

3. Java Runtime Environment (JRE)
The Java Runtime Environment (JRE) is a subset of JDK that includes the JVM and essential class libraries

4. Java API Libraries :- Java API libraries

provide a vast collection of pre-built classes and methods that simplify common programming task.

5. Java Debugger :- The Java Debugger (jdb) is a powerful tool for debugging Java applications. It allows developer to set breakpoints, inspect variables, and step through the code to identify and fix issues during development.

6. Java Documentation Generator (Javadoc) :- Java Documentation (Javadoc) automatically generates documentation, making it easier for developers to understand & use the classes & methods provided by the application's codebase

7. Additional Utilities :- Apart from the major components mentioned above, JDK also includes various utilities that facilitate develop tasks.

Page No.:
Date:

) Differentiate between JDK, JVM & JRE.

JDK	JRE	JVM
Stands for Java Development kit	Stands for java runtime environment	Stands for Java Virtual machine.
It primarily insists in executing codes & functions in development	major responsibility for creating an environment for the execution of code.	Specifies all the implementations. It is responsible for providing all of the implementations to the JRE.
The JDK is platform dependent. it means that for every diffent platform, you require a different JDK	Platform dependent. for every different platform, you require a different JRE.	Platform independent. won't require a different JVM for every diffent platform
JDK = Development tools + JRE (Java Runtime environment)	JRE = Libraries for running applications + JVM.	Only the runtime environment help in execution

Page No.:
Date:

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

Ans 1. Loads the Java code :- The JVM loads the java bytecode into memory.

2. Verifies the code :- The JVM verifies the bytecode to make sure that it is valid & does not contain any security vulnerabilities.

3. Execute the code :- The JVM interprets the bytecode & execute the instructions

4. Provides runtime environment :- Provides a runtime environment for the java program, which includes things like memory management, garbage collection & security features.

4) Explain the memory management system of the JVM.

Ans The memory management system of the JVM is crucial for managing memory resources efficiently & ensuring the proper execution of java applications.

a. Heap Memory :- The JVM's heap memory is where objects are allocated. It's the runtime data area from which memory for all class instances & arrays are allocated.
The heap is divided into two areas:
1) Young generation :- Newly created objects are initially allocated in young generation. It's further divides into Eden Spaces & two Survivor Spaces.
2) Old Generation :- This area holds objects that have survived multiple gabage collection cycles in the young generation. Objects in the old generations are typically longer-lived.

b. Stack Memory :- Each thread in Java application has its own stack memory, which stores local variables, method invocations, and partial results.

c. Method Area :- The area stores metadata, static variable & constant pool.

d. Garbage Collection :- The JVM periodically garbage collection to reclaim memory occupied by objects that are no longer referenced or needed.

e. Tuning & Optimization :- JVM provides various command-line options and settings for tuning and optimising the memory management system according to the application's requirements.

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

Ans→ The JIT Compiler is a key ~~important~~ component of the JVM responsible for optimizing the execution of java bytecode at runtime.

Role of JIT Compiler :-

- When a java program is compiled it translated into platform-independent bytecode, which is executed by the JVM.
- The bytecode is platform independent, meaning it can run on any system with a compatible JVM.
- Executing bytecode directly can be less efficient than executing native machine code.

- It takes the bytecode and translate it into native machine code specific to the underlying hardware & operating system.

Bytecode :

- Bytecode is the intermediate representation of java source code after it compiled by the java compiler.
- It's a set of understandable by the JVM.
- Bytecode is portable & platform - independent, allowing java programs to run on any system with a JVM installed without requiring recompilation.
- Importance of Bytecode is :-

- Portability :- Bytecode allows java applications to run on any platform with a JVM, eliminating the need of platform-specific binaries.

- Security :- Bytecode execution within the JVM provide a secure environment by preventing direct access to system resources

- Performance :- While bytecode execution may not be as efficient as native code, the JIT compiler optimizes it to improve performance.

- Flexibility :- Bytecode can be dynamically loaded and executed, enabling features such as dynamic class loading & code generation.

6) Describe the architecture of the JVM ?

Ans - The architecture of JVM is designed to provide a runtime environment for executing java bytecode. It consists of several components that work together to interpret & execute Java programs efficiently.

1) Class loader Subsystem :-

- The class loader Subsystem is responsible for loading classes into the JVM at runtime.

- It loads classes from various sources such as the local file system, network or other custom sources.

- The class loader Subsystem maintains a class hierarchy & ensures that classes are loaded only when needed.

2. Runtime data areas :

- The JVM's runtime data areas include memory areas used during program execution to store data & support dynamic operations.

Key runtime data areas include :-

a) Method Area :- Stores class metadata, static variables & constant pool.

b) Heap :- Memory are used for storing objects created during runtime. its divided into young generation, old generation & permanent generation.

c) Stack :- Each thread in a java application has its own stack memory, used for method invocations, local variables, & partial result.

3. Execution Engine :- The execution Engine is responsible for executing java bytecode.

It consists of :

1. Interpreter :- interprets bytecode instructions & executes them sequentially. This allows for platform-independent execution but may be slower compared to native code execution.
2. JIT Compiler : Compiles frequently executed bytecode sequences into native machine code at runtime for performance optimisation

Page No.:
Date:

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

Ans → Java achieves platform independance through JVM and the use of bytecode.

1. Bytecode :- The java compiler translates the source code into platform-independent bytecode. Bytecode is a set of instructions designed to be executed by the JVM.

2. JVM interpretation :- The JVM interprets bytecode instructions at runtime. Instead of executing the bytecode directly on the underlying hardware, the JVM interprets the bytecode instructions & execute them accordingly.

3. Standard Libraries :- Java provides a comprehensive set of standard libraries that abstract away system-specific functionalities.

4. Class loading Mechanism :- Java's class loading mechanism allows classes to be loaded dynamically at runtime. This flexibility enables java applications to load classes from various sources such as the local file system, network or custom resources.

Page No.:
Date :

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

Ans- The significance of the class loader is :-

1) Dynamic Loading :- Java's class loader dynamically loads classes into the JVM at runtime when they are needed.

2) Classpath Management :- Class loaders are responsible for managing the classpath, which is a list of directories & JAR files where the JVM searches for classes.

3) Security :- Class loaders play a role in enforcing security policies by controlling access to classes and resources.

4) Customization & Extensibility :- Java's class loading mechanism is highly customizable and extensible. Developers can create custom class loaders to implement specific loading behaviour, such as loading classes from non-standard locations or applying custom transformation or validation logic during class loading.

Page No.:
Date:

- Garbage Collection :- Its the process of reclaiming memory occupied by objects that are no longer referenced or needed by the application.