Page No
Date

Assignment-3

Q1. Explain the Components of JDK?

Ans. The JDK is a comprehensive software development kit provided by Oracle Corporation for developing Java applications.

Components :-

i) Java Compiler (javac) :-

Converts .java files into bytecode (.class files) The bytecode is platform-independent and can be executed on any device with JVM.

ii) Java Virtual Machine (JVM) :-

JVM is crucial as it interprets and runs the Compiled bytecode.

iii) Java Runtime Environment (JRE) :-

Provides the necessary libraries and Components to Run Java applications.

iv) Java Standard library :-

A set of pre-written classes and APIs that provide standard functionality for Java programs. like networking, file I/O, UI.

v) Java Debugger (jdb):

A command line tool for debugging Java prog.

vi) JavaDoc (javadoc):

A documentation generator that creates HTML documentation from Java source code comments.

vii) Java Archive (jar):

A tool for packaging Java classes & associated metadata into a single archive file (.jar).

Q2. Differentiate b/w JDK, JVM & JRE?

Ans. JDK :- (Java Development Kit)

The JDK is a software development environment which is used to develop Java Applications & applets. It physically exists. It contains JRE + development tools.

JDK contains a private JVM & a few other resources such as interpreter/loader, a compiler, an archiver, a documentation generator.

JRE :- (Java Runtime Environment)

It is used to provide the Runtime environment.

It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

JVM :- (Java Virtual Machine)

It is an abstract Machine. It is called a Virtual Machine because it doesn't physically exists. It provides a RTE in which Java bytecode can be executed.

There are three norms of the JVM :-

- specification
- Implementation
- Instance.

Q3. What is the role of the JVM in Java? And how does the JVM execute Java code?

Ans Role of JVM :-

1. Platform Independence → The JVM provides the ability to run Java applications on any device or operating system. That has a compatible Jvm implementation. This is possible because Java Code is compiled into bytecode, which the JVM interprets or compiles into native machine code.

2. Byte Code Execution → The JVM interprets the Compiled Java bytecode and Converts it into machine code.

3. Memory management → includes garbage collection automatically, which reclaims memory occupied by objects that are no longer in use.

4. Security → This includes verifying bytecode for safety and preventing unauthorized access to system resources.

How the JVM Executes Java Code :

1. Compilation to Bytecode
2. Loading Bytecode.
3. Bytecode verification
4. Execution
5. Execution of Native Code
6. Garbage Collection.

Q4. Explain the memory management system of the JVM.

Ans. In Java, memory management is the process of allocation & de-allocation of objects, called Memory Management. Java does memory management automatically by using a garbage collector.

Java Memory Management divides into two major parts →

- Jvm Memory structure
- Working of the Garbage Collector.

Q5. What are the JIT Compilers and its role in the Jvm? What is the bytecode and why is it important for Java?

Ans The JUst-In-Time Compiler is an integral part of the Jvm that optimizes Java Bytecode execution. Instead of interpreting bytecode line-by-line, the JIT Compiler translates bytecode into native machine code at runtime. By optimizing frequently executed code paths and reducing the overhead of interpretation the JIT compiler enhances the efficiency of Java applications.

Bytecode is an platform-independent representation of Java code produced by the Java Compiler. It is crucial because it allows Java prog. to be portable across different OS & hardware platforms.
When Java code is compiled into bytecode, it can be executed on any device with a compatible JVM.

Q6. Describe the architecture of the JVM?

Ans 1. Class Loader Subsystem:
- Function: Loads Java classes into the JVM
- Components:-
  - Bootstrap Class loader → Loads core Java classes from the JDK.

- Platform Class Loader → Loads standard library classes
- Application Class loader → loads classes from the application's classpath.

2. Runtime Data Areas:
   - Heap: stores objects and arrays; managed by GC
   - Method Area: Contains class-level data, such as class definitions and static variables.
   - Stack: Each thread has its own stack for method calls and local variables.
   - Program Counter Register: - keep track of the current instruction being executed.
   - Native Method Stack: - Handles non-java method calls.

3. Execution Engine:
   - Interpreter: Executes bytecode instructions directly, suitable for initial runs.
   - JIT Compiler: Converts bytecode into native mc.
   - Garbage Collector: - reclaims memory by removing objects that are no longer in use.

Q1. How does Java achieve platform independence through the JVM?

Ans. Java achieves platform independence by compiling Java source code into platform-neutral bytecode using the Java compiler. This bytecode is then executed by the JVM, which is available for different platforms. The JVM interprets or compiles the bytecode into native machine code specific to the host

Page No.
Date

OS and hardware.

Q8. What is the significance of the class loader in Java? What is the process of Garbage Collection in Java?

Ans. The class loader is responsible for dynamically loading Java classes into the JVM during runtime. It ensures that classes are loaded in a systematic order and provides a way to locate and load classes from various sources. (e.g. files, network).

Garbage Collector process → automatically reclaims memory used by objects that are no longer reachable from the prog.

1. Marking
2. Normal Deletion
3. Deletion with Compaction.

Q9. What are the four access modifiers in Java, and how they are differ from each other?

Ans. 1. Public : Accessible from any other class.

2. protected : Accessible within the same package and by subclass

3. default : Accessible only within the same package, no explicit modifier is needed.

4. private : Accessible only within the same class.

Q10. What are the diff b/w public, protected & default access modifiers?

Ans. public :- The member is accessible from any class, regardless of the package.

protected :- The member is accessible within the same package and by subclass, even if they are in different packages.

default :- The member is accessible only within the same package. No modifier is used to indicate default access.

Q11. Can you override a method with a different access modifiers 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 with a more restrictive access modifier. If a method is protected in a superclass, it can be overridden in a subclass with protected or public access, but not with private access. Overriding with a more restrictive access modifier would result in a compilation error.

Q12. What is the diff. b/w protected & default access?

Ans. protected :- Accessible with the same package & by the subclass

default :- Accessible only within the same package. Subclasses in other packages can't access these members

Page No.
Date

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

Ans No, you cannot declare a top-level class as private in Java. The only allowed access modifiers for top-level classes are public and default (package-private). A class can be private only if it is an inner class (nested class) within another class, which restricts its visibility to the enclosing class only.

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

Ans No, a top-level class cannot be declared as protected or private. The only valid access modifiers for top-level classes are public & default. This restriction exists because top-level classes need to be accessible to other classes in the package or to be publicly available if declared as public.

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

Ans. If a variable or method is declared as private, it is accessible only within the same class. Even if another class is in the same package, it cannot access private members of another class.

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

Ans The Package-private or default access is the level of access that is granted when no access modifier is specified. Members with package-private access are visible to all classes within the same package but are not accessible from classes outside the package. This access level is useful for encapsulating implementation details that are shared within a package but should not be exposed outside it.