Assignment 3

M T W T F S S
Page No.:
Date:
YOUVA

1) Explain the components of the JDK.

⇒ The Java Development Kit (JDK) includes the following components :-

1. Java compiler (javac) :- Translates Java code into bytecode.
2. Java Runtime Environment (JRE) :- Provides the libraries, Java virtual Machine (JVM) and other components to run Java application.
3. Java Virtual Machine (JVM) :- Executes the compiled Java bytecode
4. Java Standard Library :- A collection of pre-written classes and functions that provide essential utilities.
5. Development tools :- Includes tools like 'javadoc' for documentation, 'jdb' for debugging and 'jar' for packaging Java applications into JAR files.

2) Differentiate between JDK, JVM and JRE.

⇒ Java development Kit :-

i) used for developing java applications.
ii) Java compiler (javac), JRE, development tools like javadoc, jar, jdb. are the components of JDK.
iii) Developers who need to write, compile and debug Java programs.

Java Virtual machine (JVM):-

i) Executes Java bytecode on any device or OS.

ii) Components of JVM :- Bytecode interpreter, garbage collector (part of JRE).

iii) End-users indirectly use the JVM when running Java application.

Java Runtime Environment (JRE):-

i) Provides the environment to run Java applications.

ii) Components of JRE :- JVM, standard libraries (excludes development tools).

iii) End-users who wants to run, not develop, Java application.

3) What is the role of the JVM in Java? and How does the JVM execute Java code?

⇒ Role of the JVM in Java:-

The Java Virtual Machine (JVM) executes Java bytecode, providing platform independence by abstracting the underlying operating system. It manages memory, performs garbage collection and ensures security, enabling Java's "write once, run anywhere" capability.

How the JVM Executes Java Code:-

The JVM loads & verifies bytecode, then executes it using an interpreter or Just-In-Time (JIT) compiler, which converts frequently used bytecode into native machine code for faster

performance. It also handles memory management and provides runtime services like threading & exception handling to ensures smooth and efficient execution of Java applications.

4) Explain the memory management system of the JVM.

⇒ The JVM's memory management system includes:-
1. Heap memory - stores Java objects; divided into Young & Old Generations for efficient garbage collection.

The memory management system in JVM is known as the Runtime Data Area.
It is divided into 5 parts : 1. Method Area.
2. Heap section.
3. Java stack.
4. PC Register.
5. Native Method stack.

↳ All the class-level variables (static field) and methods will get space in Method Area.
↳ All the instances of the class created using new keyword will get space in Heap Memory Area.
↳ All the method-local variables including references will be stored in Java Stacks Memory Area. Also, every time a method gets called, a stack frame (just like func. Activation record in C) will be made for it where its local variables (including parameters) will get space.

↳ PC registers are used to store the current executing instruction and once it gets executed, the next instruction will be stored here.

↳ If the code is implemented in c/c++ (native with respect to Java), then Native Method Stacks will be used.

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

⇒ The JIT compiler in the JVM improves Java application performance by converting frequently executed bytecode into machine code at runtime allowing faster execution. It works alongside the JVM's interpreter, compiling hot spots of code & optimizing them for efficiency.

Bytecode is platform-independent intermediate code generated by the Java compiler. It is executed by the JVM, allowing Java to maintain its "write once, run anywhere" promise. Bytecode is crucial because it enables platform independence, enhances security through JVM checks & allows runtime optimizations by the JIT compiler for better performance on different platforms.

6) Describe the architecture of the JVM

⇒ The architecture of the Java Virtual Machine (JVM) consists of several key components that work together to execute Java programs:

1. Class Loader :-
Responsible for loading Java classes (bytecode) into the JVM. It loads classes as needed and handles tasks like loading, linking & initializing classes dynamically during runtime.

2. Runtime Data Area :-
is divided into Method Area, Java Stacks, Heap, PC Register & Native Method Stacks.

3. Execution Engine :-
Responsible for executing bytecode. It includes:
- Interpreter :- Reads & executes bytecode instructions line by line.
- Just-In-time (JIT) Compiler :- compiles frequently executed bytecode into native machine code for better performance.
- Garbage Collector :- Manages automatic memory management by reclaiming memory used by objects that are no longer in use.

4. Native Method Interface (JNI) :-
Allows the JVM to interface with native code written in languages like C/C++.

5. Native Labraries :-

External libraries that the JVM uses to interacts with the underlying operating system and hardware.

The JVM architecture ensures platform independence, memory management & runtime efficiency for Java programs.

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

⇒ The compiled code (.class file) on machine can be executed on any other machine/platform as long as they have the JRE i.e. Java library and JVM installed. So, a .class file is platform independent & can be executed on any platform.
Note :- JVM itself is platform dependent as there are other versions of it based on the OS.

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

⇒ Significance of the class loader in Java :-
The class loader is a crucial part of the Java Virtual Machine (JVM) responsible for dynamically loading Java classes at runtime. Its significance includes :-

M T W T F S S

Page No.: YOUVA

Date:

1. Dynamic class Loading :-
Classes are loaded only when required, saving memory & improving efficiency.

2. Namespace Management :-
It ensures that each class is loaded within its own namespace, avoiding conflicts.

3. Security :-
The class loader ensures that classes are loaded from trusted sources and can enforce access restrictions.

4. Extensibility :-
Developers can define custom class loaders for special class-loading mechanism (e.g. loading classes from non-standard sources like databases or network locations).

Process of Garbage Collection in Java

Garbage Collection (GC) in Java is an automatic process that manages memory by reclaiming memory used by objects no longer referenced by the program. The JVM's garbage collector identifies these unused objects & frees up memory, ensuring efficient memory use without manual intervention.

Key steps in garbage collection:-

1. Marking : The GC identifies which objects are still reachable (in use) & marks them.
2. Sweeping : It identifies unreferenced objects that are no longer needed.
3. Compacting :- After reclaiming memory from unreferenced objects, the remaining objects are compacted to reduce fragmentation & make memory allocation more efficient.

Java's garbage collection process runs in the background, improving memory management without the need for explicit memory deallocation by the developer.

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

Modifier	Class	Package	Subclass	World
Public	Yes	Yes	Yes	Yes
Protected	No	Yes	Yes	No
Default	No	Yes	No	No
Private	No	No	No	No

↳ The 4 access modifiers in java are private, default (package level private), protected & public.

- The member (fields or methods) declared as private cannot be accessed outside of their own class. Usually, the fields should be declared as private.
- All the members are by default package level private i.e. there are accessible anywhere inside the same package.
- Protected members are same as default i.e. accessible everywhere within their own package except they are also accessible inside the sub class outside the package.
- Public members are available everywhere. The methods should be usually declared as public

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

⇒ Default (package level private) members are accessible anywhere inside the same package.
Protected members are same as default i.e accessible everywhere within their own package except they are also accessible inside the sub class outside the package.
Public members are available everywhere. The methods should be usually declared as public.

11) Can you override a method with a different access modifier in a subclass? For example, can a protected method in a superclass be override with a private method in a subclass? Explain.

⇒ The overriding method in the subclass should have the same access modifier or of a wider type. i.e protected method in super class can be overridden by protected or public method in sub class. protected method in super class cannot be overridden with a private method in the subclass.

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

⇒ Default (package level private) members are accessible anywhere inside the same package.
Protected members are same as default. i.e accessible everywhere within their own package except they are also accessible inside the sub class outside the package.

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

⇒ In Java, you cannot make a top-level class private. Top-level classes can only have 'public' or package-private (default) access. A 'public' class is accessible from any package, while a package-private class is accessible only within its own package.
Inner classes (nested classes) can use 'private', 'protected', 'public' or package-private access. A 'private' inner class is accessible only within its enclosing class, allowing for encapsulated implementation details. This distinction helps manage visibility & encapsulation, but top-level class visibility is restricted to 'public' or package-private only.

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

⇒ No, a top-level class in Java cannot be declared protected or private. Top-level classes can only be public or package-private (default). This is because Java requires top-level classes to be accessible at the package or global level for proper compilation & runtime management. protected & private access modifiers are meant for controlling access to class members (fields, methods) rather than the classes themselves. These restrictions ensures that top-level classes are either accessible from other packages (if public) or restricted to the same package (if package-private), enabling effective class organization and usage.

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

⟹ You cannot access a private field or method from outside the class it is defined in as they are visible only inside their own class.

16) Explain the concept of "package-private" or "default" access. How does it affect the visibility of class members?

⟹ Default or package level private means that the member can only be accessed within the same package. They can be accessed from any other file as long as all the files are present inside the same package.