classmate
Date
Page

Assignment - 3

* Interview Questions :-

21) Explain the Components of JDK.

→ JDK is a software package used to develop Java applications.

• Components :-

1) Compiler :- Converts Java code into bytecode.
2) JRE :- Runs Java applications.
3) JVM :- Executes Java byte code to machine code
4) Standard library :- Pre-built classes for common tasks
5) Debugger :- Helps find and fix errors in Java programs
6) Documentation :- Provides details about classes methods and fields.

22) Differentiate JDK, JVM & JRE.

→ ① JDK :-

- A full software development kit for Java containing tools like compiler, debugger & other utilities
- It includes JRE and the JVM. So you can develop and run Java applications.

② JVM :-

- the engine that runs Java bytecode by converting it into machine code for specific system.
- It's Platform specific and is a part of both JDK & JRE.

③ JRE :
- Provides libraries and JVM necessary to run the Java programs / applications.
- It doesn't include development tools or compiler. It's used only to run java applications.

Q3. What is the role of JVM in Java? & How does the JVM execute Java code.

→

• Role of JVM :-
① Executes Java code by converting bytecode into machine code.
② Manages Memory with automatic garbage collection.
③ Enables platform independence so java programs can run on any OS.

• How JVM executes Java code :-
① Compiles Java sourcecode into bytecode.
② Loads bytecode into memory with class loader.
③ verifies bytecode for security & corrections.
④ Interprets or JIT Compiles bytecode into machine code.
⑤ Manages Memory through garbage collector.

Q4) Explain Memory Management in JVM

→ The JVM's Memory Management System is designed to efficiently handle the allocation and deallocation for Java applications. It consists several key areas.

① Heap Memory :- Stores objects and their corresponding data.

② Stack Memory :- Stores method calls, local variables and references to objects in heap.

③ Program Counter (PC) Register :- Holds the adress of next instruction to be executed in current thread.

④ Native method stack :- Supports the execution of native (non-Java) methods such as those written in C or C++.

⑤ Garbage Collectors :- automatically identifies and removes objects that are no longer in use to free up Memory.

Q5) What are Jit Compiler & it's role in JVM ? What is the bytecode and why it is important for Java?

→

- JIT (Just in time) compiler :- It is a component of JVM that enhances the performance of Java applications by converting into bytecode into machine code.

- Role :-
① Optimization :- optimizes the frequently executed codes and improves performance.
② Execution speed :- once compiled the native code can be executed directly by CPU, resulting in faster execution.

- Bytecode :-
- ~~An~~ The code generated by java compiler from Java source code which is stored in .class file, which are platform independent, Executed by JVM, which interprets or compiles it to Machine code for specific platform.

- Importance :-
① Platform Independent :- bytecode can run on any platform with JVM making it "write once, run anywhere".
② Security :- bytecode is verified by JVM before execution.
③ Efficiency :- allows effecient execution through JIT Compiler which can optimize code on runtime.

Q6) Describe the architecture of JVM
→ JVM architecture consists of :-

① Class loader subsystem :- Loads Java classes ~~from~~ into memory.

• Types :-

1.1) Bootstrap class loader :- loads standard classes in memory or JVM.

1.2) Extension class loader :- loads exe files from exe directory present in lib.

1.3) System class loader :- loades classes from HDD to JVM

② Memory areas :-

2.1 Method area :- Stores classes, methods,

2.2 Java stack :- Stores local variable, method calls.

2.3 Heap :- Stores objects & data related to it.

2.4 Program Counter registers :- tracks the address of next instruction to be executed

2.5 Native method stack :- supports Native method.

③ Execution engine :-

3.1 Interpreter :- Reads & executes bytecode instructions.

3.2 JIT compiler :- Compiles bytecode into machine code.

④ Native interface :- Allows Java code to interact with native applications & libraries.

⑤ Garbage Collector :- Automatically manages memory by removing unused or unnecessary objects.

Q7). How does Java achieve platform independence through JVM?

→. Java Compiler converts the Java source code into bytecode by compiling it. The bytecode is stored in a file with extension .class. JVM compiles or interprets the .class files to convert it into machine code which is specific to machine.

The bytecode is not platform specific. we can compile it on any system or OS with JVM which later compiles it to specific machine code.

Q8). What is significance of the class loader in Java? What is the process of garbage collection in Java.

→. • Significance of class loader.

Class loader loads the Java classed into JVM's memory at runtime as needed. It does not load all classes at once. It manages where classes come from and ensure the don't conflict.

• Garbage collection process.

1) Find unused objects.
2) Mark them
3) remove marked objects.

Q15) What are four access modifiers in Java, and how are the different from each other?

→ In Java there are four access modifiers that control the visibility and accessibility of classes, methods & variables.

1) Public :- accessable from any other class or package.
2) protected :- Access within its own package and by subclasses even if they are in different package.
3) Default :- Accessible only within it's own package.
4) private :- Accessible only within the class where it is declared.

Q16) what is difference between public, protected & default access modifier?

→

- Pubic :- It is accessible from any other class or package.
- Protected :- accessible in package as well as subclass in another package.
- Default :- accessible only within its own package.

Q11) Can you override a method with a different access modifier in a subclass? for ex. can a protected method in a Superclass be overriden with private method in Subclass? Explain.

→ No, we can't override a method in a subclass with a more restrictive access modifier.

Explaination :-

- If a ~~sub~~ method in Superclass is protected, it means it can be accessed by subclasses and other classes in the same package.
- If you try to override this protected method in a subclass with a private method, it would block access to the method from outside the subclass which isn't allowed.

Q12) ~~W~~hat is difference between protected and default access?

→ refer to Q10.

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

→

In Java it is not possible to make a class private. However we can make Inner class in nested classes, private.

- limitations :-
  a class can not be private. It can be public. but a class inside a class can be private. which is accessible in outer class only.

ex :-

```
public class Outer {
    private class Inner {

    }
}
```

Q14) Can a top-level class in Java be declared as Protected or Private?

→ No we cannot create top-level classes ~~are~~ private or Protected.
A top level class can only be pubic or default.
The Protected & private modifier is only for methods, fields inside a class or for inner class, not for top level class.

Since there's no use case for top-level class being private or Protected Java doesn't allow these modifiers for top-level classes.

Q15) What happens if you declare a variable variable or method as private in a class and try to access it from another class within same package

→ No we cant access a private members of a class from another class because those members are accessible in their class only.

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

→ In Java if you dont specify an access modifier for a class, method, or variable it automatically gets default access.

- visibility :- It is accessible only in the same package.

- Effect on visibility :-
if a class is default it is accessible only within package.
if a method or variable is default, it can be accessed by other classes in same package.