Module 2 Assignment 3 Interview questions

1. Explain the components of the JDK

Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools needed in Java development.

2. Differentiate between JDK, JVM, and JRE.

- JDK is the development platform, while JRE is for execution.
- JVM is the foundation, or the heart of the Java programming language, and ensures the program's Java source code will be platform-agnostic.
- JVM is included in both JDK and JRE—Java programs won't run without it.

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

JVM is a virtual machine that enables the execution of Java bytecode. The JVM **acts as an interpreter between the Java programming language and the underlying hardware**

The JVM executes Java code by:

1. **Loading**: The class loader loads `.class` files (bytecode) into memory.
2. **Verifying**: It checks the bytecode for security and correctness.
3. **Executing**: It either interprets the bytecode directly or uses a Just-In-Time (JIT) compiler to convert it to native machine code for faster execution.
4. **Managing Memory**: It handles memory allocation and garbage collection automatically.

4. Explain the memory management system of the JVM.

JVM defines various run time data area which are used during execution of a program. Some of the areas are created by the JVM whereas some are created by the threads that are used in a program.

Heap :

- It is a shared runtime data area and stores the actual object in a memory. It is instantiated during the virtual machine startup.
- This memory is allocated for all class instances and array. Heap can be of fixed or dynamic size depending upon the system's configuration.
- There exists one and only one heap for a running JVM process.

Method Area:

- It is a logical part of the heap area and is created on virtual machine startup.
- This memory is allocated for class structures, method data and constructor field data, and also for interfaces or special method used in class. Heap can be of fixed or dynamic size depending upon the system's configuration.

- Can be of a fixed size or expanded as required by the computation. Needs not to be contiguous.

JVM Stacks:

- A stack is created at the same time when a thread is created and is used to store data and partial results which will be needed while returning value for method and performing dynamic linking.
- Stacks can either be of fixed or dynamic size. The size of a stack can be chosen independently when it is created.
- The memory for stack needs not to be contiguous.

Native method Stacks:

Also called as C stacks, native method stacks are not written in Java language. This memory is allocated for each thread when its created. And it can be of fixed or dynamic nature.

Program counter (PC) registers:

Each JVM thread which carries out the task of a specific method has a program counter register associated with it. The non native method has a PC which stores the address of the available JVM instruction whereas in a native method, the value of program counter is undefined. PC register is capable of storing the return address or a native pointer on some specific platform.

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

The JIT compiler in JVM **dynamically compiles sections of bytecode into native machine code at runtime**. This allows the JVM to strike a balance between the portability of bytecode and the efficiency of native code execution.

Java bytecode is **the instruction set of the Java virtual machine (JVM), the language to which Java source code is compiled**. Each instruction is represented by a single byte, hence the name bytecode.

6. Describe the architecture of the JVM.

The JVM architecture consists of several key components, including the class loader, runtime data areas (such as the heap and method area), execution engine, and native method interface (JNI). These components work together to execute Java bytecode and manage runtime operations

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

Java is platform-independent because it uses a "Write Once, Run Anywhere" approach. Java source code is compiled into bytecode, which is platform-neutral. This bytecode can be executed on any platform that has a Java Machine (JVM) compatible with that bytecode.

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

Class loaders are **responsible for loading Java classes dynamically to the JVM (Java Virtual Machine) during runtime**.

Garbage collection in Java is the process of automatically reclaiming memory by deleting objects that are no longer reachable by the application.

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

- **`public`** : Accessible from anywhere.
- **`protected`** : Accessible within the same package and in subclasses across packages.
- **`default`** (no modifier): Accessible only within the same package.
- **`private`** : Accessible only within the defining class.

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

- **`public`** : Accessible from anywhere.
- **`protected`** : Accessible within the same package and in subclasses across packages.
- **`default`** (no modifier): Accessible only within the same package.
- **`private`** : Accessible only within the defining class.

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

No, you cannot override a method with a more restrictive access modifier in a subclass. For example, a `protected` method in the superclass cannot be overridden as `private` in the subclass.

because it reduces visibility,Java ensures that the overridden method maintains or increases accessibility, but never reduces it. You can override a `protected` method with a `public` one, which makes it more accessible. This rule ensures that the subclass doesn't reduce the visibility of inherited methods, which could violate the principles of polymorphism.

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

Protected access allows a member to be accessible within the same package and by subclasses, even if they are in different packages.

Default (package-private) access allows a member to be accessible only within the same package, but not by subclasses in other packages.

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

No, you cannot make a top-level class private in Java. However, you can make an **inner class** (a class within another class) private.

Limitations:

- A private inner class is only accessible within the outer class that contains it.
- It cannot be accessed or instantiated from outside the outer class.

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

no, **`protected`** and **`private`** access modifiers are meant for controlling access to members (methods, fields) within a class or between classes within the same package or subclass. They do not apply to top-level classes.

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?

compilation error.

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

Package-Private (Default) Access:

- If you don't specify an access modifier (like `public` or `private`), the member (variable or method) or class has package-private access by default.
- **Visibility**:
  - **Within the Same Package**: Other classes in the same package can see and use it.
  - **Outside the Package**: Other classes in different packages cannot see or use it.