- Code samples (example projects that demonstrates how to implement certain features using the SDK)

Assignment : 3

1. Explain the components of the JDK

→ • Java compiler (javac) : It transforms the java source code into bytecode. The generated bytecode can be executed on any platform with a JVM installed, ensuring the "write once, run anywhere" philosophy of java.

• JVM : It is a runtime engine that executes java bytecode. It provides an abstraction layer betn the java appln and the underlying OS. The JVM enables java programs to run independently of the hlw & OS, enhancing portability & security

• JRE : It is a subset of JDK that includes the JVM & class libraries. It is required to run java appln on end-user systems without the need for dev. tools.

• Java API libraries : It provides a vast collection of pre-built classes & methods that simplify common programming tasks.

• Java Debugger : It is a powerful tool for debugging java applications. It allows developers to set breakpoints, fix issues during development

• Java Documentation Generator : It automatically

generates documentation from the source code. It helps in creating comprehensive API documentation making it easier for developers to understand

- Additional utilities: Apart from the major components, JDK also includes various utilities that facilitate development tasks.

2. Differentiate bet" JDK, JVM, JRE

→

JDK	JRE	JVM
• It is a SDK used to develop Java applications	• It is a s/w package that provides JVM, class libraries & other components to run appli" of java	• It is an abstract machine that provides an envirn for the execution of java Bytecode
• It contains tools for developing monitoring & debugging java codes	• It contains class lib & othe supp- orting files req. by JVM for executing java prg.	• It does not include any s/w dev. tools
• It is a plat- form dependent	• It is also plat form dependent	• It is platform independent
• JDK = JRE + development tools	• JRE = JVM + class libraries	• JVM = Runtime environment

3. What is the role of JVM & how does the JVM executes java code?

→ JVM loads and verifies and runs Java bytecode It is known as the interpreter or the core of the Java prog. lang.

execution :

- Loading : The JVM loads the class file containing the java bytecode into memory. The process is managed by the class loader.
- Bytecode verification : It ensures that the bytecode does not perform illegal operation or violate access rights.
- JIT : While executing the bytecode, the JVM may use JIT compilation to convert the bytecode into native machine code
- Garbage collection : During execution, the JVM monitors memory usage and automatically performs GC to freeup memory by removing obj that are no longer in use.

4. Explain memory mgt system of JVM

→ JVM memory structure

- Heap : The Heap is the area of memory where all objects and their instance variable are stored.
- Stack : The purpose of stack is where method invocation, local variables, and ref. variables are stored. Each thread has its own stack
- Method area : It stores class structures such as runtime constant pool, field &

method data, and the code for methods. Static variables are also stored here.

- Program Counter: Small memory area which holds the addr. of the current instruction that the JVM is executing for each thread
- Native method stack: This stack is used for executing native methods via JNI
- Garbage collector: JVM includes a garbage collector that automatically manages memory by reclaiming memory from obj. that are no longer in use
- Tuning and monitoring: Java provides several tools like jvisualvm, jconsole, and profilers to monitor memory usage & garabage collection

5 What is the JIT compiler & its role in JVM? What is bytecode & why it is imp?

→ Just-in-time compiler is a component of JVM that enhances application performance by converting java bytecode into native machine code at runtime. Instead of interpreting bytecode line-by-line the JIT compiler identifies frequently executed code paths and compiles them into machine code, allowing the CPU to execute them directly

Bytecode is an intermediate, platform independent code that is generated when java source file is compiled by java compiler. The bytecode is stored in .class files.

Importance of Bytecode: It provides - Platform independence, security, optimization, portability.

6 Describe the architecture of JVM.

→ • classloader : It is a subsystem which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. It contains
- Bootstrap classloader : It loads the rt.jar files into JVM memory
- extension classloader : Loads classes from the ext directory, which are extensions to the core java libraries
- Application classloader : loads classes from the application classpath, including user-defined classes

• class (Method) Area : It stores per class structure such as the runtime constant pool, field and method data, the code for methods.
- Heap : It is the runtime data area in which objects are allocated.
- Stack : It holds local variables & partial results and plays a part in method invocation & return.
- PC : PC stores the address of JVM instruction currently being executed
• execution Engine : It contains
- A virtual processor
- Interpreter
- Garbage collector
- JIT

• Java Native Interface : It is a framework which provides interface to comm" with another written in another lang.

Date
Page

8. How does Java achieve platform independence ?

→ Java achieve platform independence through JVM by compiling Java source code into an intermediate form know as bytecode. This byte code is platform-independent and can be executed on any device or OS that has a compatible JVM. Each platform has its own JVM implementation, which knows how to translate the platform-agostic bytecode into native instructions required by system.

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

→ There are four types of access modifiers in Java

- Private : The access level of a private modifier is only with the class. It cannot be accessed from outside the class.
- Default (Package private) : It cannot be accessed from outside the package. It is accessible only within the package
- Protected : It can be accessible within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package
- Public : It can be accessible everywhere within the class, outside the class, within the package and outside the package.

10 Difference between public, protected, default access modifiers

→

Public	Protected	Default
• This is applicable for both top level classes and interfaces	• This modifier is not applicable for both top-level classes & interfaces	• This modifier is applicable for both top level & interface
• Public members can be accessed from the child class of the same package	• Protected members can be accessed anywhere from the same package & only by child classes outside the package	• Package members can be accessed from the child class of the same package
• It is the most accessible modifier among all modifiers	• It is more accessible than the package and private modifier but less accessible than public modifier	• Package modifier is more restricted than the public & protected modifier but less restricted than the private modifier

11. Can you override a method with a diff access modifier in a subclass? ex.?

→ In java you cannot override a method with a different access specifier in a subclass. Access modifiers define the visibility

and accessibility of methods and members. When overriding a method, the overriding method in the subclass must have the same or more permissive access level as the method in superclass - this means that if a method in the superclass is protected, it can only be overridden by methods that are protected or public in the subclass

can a protected method in a superclass be overridden with a private method in a subclass "No"

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

→ In Java, you cannot make a top-level class private. The access modifiers for top-level classes are public or package-private. However, you can make a nested class.

limitations:

- Top-level classes: Top-level classes can only be declared as public or package-private. A public top-level class must have the same name as the filename and is accessible from any other class in any package.
- Private Nested classes: You can declare a nested class as private. This means the nested class is only accessible within its enclosing class.

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 as protected or private because these access modifiers are not applicable to top level classes. The protected modifier is intended for members within a class or nested classes to control access with the same package and sub-classes but it does not make sense for top level classes which do not have a parent-child relationship in the same way. Similarly, the private modifier restricts access to the declaring class itself, which is irrelevant for top-level classes as they are not nested within another class. Consequently, top-level classes can only be public or package-private, aligning with Java's access control model for class visibility & encapsulation

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?

→ If you declare a variable or method as private in a class, it can only be accessed within that class itself, even if other classes are in the same package. The private access modifier enforces strict encapsulation, preventing any external class, regardless of its package from accessing or modifying private members directly.

This means that other classes in the same package cannot interact with private variable or methods, ensuring that the internal details of the class remain hidden and controlled solely by the class that declares them.