💡 **Q1.What is the difference between Compiler and Interpreter**

Compiler:
It is a software which takes sourcecode(HLL) as the input and generates MLL code as the output
To convert the HLL code to MLL code compiler will scan the HLL code only once.

Interpreter:
It is a software which takes sourcecode(HLL) as the input and generates MLL code as the output
To convert the HLL code to MLL code interpreter will scan the HLL code multiple times(depends on the instructions).

Performance measurement of Compiler vs Interpreter:
→ Compiler will speed up the process ,where as interpreter will slow down the process.
→ Compiler in one Scan will identify all the problems in the code(if found),where as interpreter will do scanning line by line so
It takes more time for identifying the problem.

💡 **Q2.What is the difference between JDK, JRE, and JVM?**

if we run java program we need to set up the environment in our machines/laptops/system.
-> To set up the environment we need to install JDK software to our machines.
-> JDK stands for Java Development kit, it provides libraries and the requried files to run our java programs.
-> JDK = JRE + JVM
-> JRE : (Java RunTimeEnvironment), It provides suitable environment to run our java program.
=> JVM : (Java Virtual Machine), It is responsible to run our java programs on the basis of MultiThreading.

💡 **Q3.How many types of memory areas are allocated by JVM?**

Two kinds of memory allocated by JVM. The JVM divides its memory into two main categories: heap memory and non-heap memory.

The Java Virtual Machine (JVM) allocates 5 different types of memory areas:

1. Class(Method) Area
2. Heap
3. Stack
4. Program Counter Register
5. Native Method Stack

- **Method Area** (also known as the Class Area) stores the class code, variables, and methods of all the classes in the Java program. This area is shared by all the threads in the JVM.

- **Heap** stores all the objects created by the Java program. This area is not shared by the threads, and each thread has its own private heap.
- **Stack** stores the local variables and the method call stack for each thread. This area is also not shared by the threads.
- **Program Counter Register** stores the address of the next instruction to be executed by the thread.
- **Native Method Stack** stores the information related to native methods that are called from Java code.

💡 ****Q4.What is JIT compiler?****

JIT starts for *Just-In-Time compiler (JIT),* It is one of the integral parts of the Java Runtime Environment. It is mainly responsible for performance optimization of Java-based applications at run time or execution time. In general, the main motto of the compiler is increasing the performance of an application for the end user and the application developer.

→ javaprogram would be first compiled(javac filename.java)
→ If the compilation is succesfull, it would generate .class file
→ These .class files will be used by jvm during the execution

→ .class file generated will have instructions in bytecode(neither HLL nor MLL).
→ bytecodes will be taken by JVM and it will be loaded inside JRE, then the execution begins.

Java programs → java compilation + Execution
(javac) (java-> JIT)
.

💡 **Q5.What are the various access specifiers in Java?**

Types Of Access Modifiers In Java

Java provides four types of access specifiers that we can use with classes and other entities.

These are:

1) Default: Whenever a specific access level is not specified, then it is assumed to be 'default'. The scope of the default level is within the package.

2) Public: This is the most common access level and whenever the public access specifier is used with an entity, that particular entity is accessible throughout from within or outside the class, within or outside the package, etc.

3) Protected: The protected access level has a scope that is within the package. A protected entity is also accessible outside the package through inherited class or child class.

4) Private: When an entity is private, then this entity cannot be accessed outside the class. A private entity can only be accessible from within the class.

💡 **Q6.What is a compiler in Java?**

This will compile the Java source code file **Test.java** and create a bytecode file called **Test.class.** The bytecode file can then be executed by the JVM.

The Java compiler performs a number of tasks when it compiles a Java program. These tasks include:

- Syntax checking: The compiler checks the Java source code for syntax errors.
- Type checking: The compiler checks the Java source code for type errors.
- Code generation: The compiler generates bytecode from the Java source code.
- Optimization: The compiler can optimize the bytecode to improve performance.

The Java compiler is a powerful tool that can help you to write efficient and portable Java programs.

Some of the benefits of using a Java compiler:

- Portability: Java code compiled by the Java compiler can be executed on any platform that has a JVM.
- Performance: The Java compiler can optimize the bytecode to improve performance.
- Security: The Java compiler can help to prevent security vulnerabilities in Java code.

💡 ****Q7.Explain the types of variables in Java?****

There are three types of variables in Java:

- **Local variables** are declared inside a method or a block. They are only accessible within the method or block in which they are declared.
- **Instance variables** are declared outside of any method or block, but within a class. They are accessible to all the methods and constructors of the class.
- **Static variables** are declared with the static keyword. They are also declared outside of any method or block, but within a class. Static variables are shared by all the objects of the class.

1) Local variable: Within the method or block in which it is declared.

```
int myVariable;         //Local variable Declaration
```

2) Instance variable : Within the class in which it is declared, and accessible to all the methods and constructors of the class.

```
int myVariable;         //Instance variable    Declaration
```

3) Static variable: Within the class in which it is declared, and shared by all the objects of the class.

```
static int myVariable;          //Static variable Declaration
```

💡 ****Q8.What are the Datatypes in Java?****

Data types in Java are used to define the type of data that a variable can store. There are two types of data types in Java: primitive data types and non-primitive data types.

Primitive data types are the most basic data types in Java. They are used to store simple values, such as integers, floating-point numbers, characters, and boolean values. There are eight primitive data types in Java:

- boolean - Stores a Boolean value, either true or false.
- byte - Stores a signed byte value, from -128 to 127.
- short - Stores a signed short value, from -32768 to 32767.
- int - Stores a signed integer value, from -2147483648 to 2147483647.
- long - Stores a signed long value, from -9223372036854775808 to 9223372036854775807.
- float - Stores a single-precision floating-point value.
- double - Stores a double-precision floating-point value.
- char - Stores a character value.

Non-primitive data types are objects that are used to store more complex data. Some examples of non-primitive data types in Java include:

- String - Stores a sequence of characters.
- Arrays - Stores a collection of elements of the same type.
- Classes - Defines a blueprint for creating objects.

💡 ****Q9.What are the identifiers in java?****

An identifier is a name that is used to identify a variable, method, class, or other entity. Identifiers must follow a specific set of rules, and they can be used to make your code more readable and understandable.

Rules for identifiers in Java:

- Identifiers must start with a letter or the underscore character (_).
- Identifiers can contain letters, digits, and the underscore character (_).
- Identifiers cannot contain spaces or special characters other than the underscore character (_).
- Identifiers are case-sensitive, so myVariable is different from myvariable.

Here are some examples of valid identifiers in Java:

- myVariable
- _myVariable
- my_variable
- myVariable123

Here are some examples of invalid identifiers in Java:

- 123myVariable
- my variable
- my-variable

- myVariable@123

Identifiers are used to identify different entities in your code, so it is important to choose them carefully. You should use names that are descriptive and easy to remember. You should also avoid using names that are already used for reserved words in Java.

💡 ****Q10.Explain the architecture of JVM****

The Java Virtual Machine (JVM) is a software program that executes Java bytecode. It is responsible for loading, verifying, and executing Java bytecode. The JVM is also responsible for managing the memory used by Java applications.

The JVM architecture is divided into three main components:

- **Class loader:** The class loader is responsible for loading Java classes into the JVM. It also verifies that the classes are valid and that they can be executed by the JVM.
- **Runtime data areas:** The runtime data areas are where the JVM stores the data that is used by Java applications. These data areas include the heap, the stack, the method area, and the native method area.
- **Execution engine:** The execution engine is responsible for executing Java bytecode. It does this by translating the bytecode into machine code that can be executed by the underlying operating system.