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

**Ans.** Compiler 🡪 It is a software which takes source code(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 source code(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).

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

**Ans.** JDK stands for Java Development kit, it provides libraries and the required files to run our java programs => JDK :: JRE + JVM

JRE :: Java Run Time Environment, 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 Multi Threading.

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

**Ans.** Heap: The heap is the main memory area used for dynamic memory allocation.

Method Area: The method area (also known as the non-heap memory or the permanent generation in older JVM versions) is used to store class structures, method information, and constant pool data.

Program Counter (PC) Registers: Each JVM thread has its own program counter, which keeps track of the current executed byte code instruction.

**Q4. What is JIT compiler?**

**Ans.** JIT stands for "Just-In-Time." A JIT compiler is a type of compiler that dynamically compiles and executes code at runtime, as opposed to ahead-of-time (AOT) compilation where code is compiled before it is executed. JIT compilation combines elements of both interpretation and compilation.

When a program is executed by a JIT compiler, it first starts by interpreting the code, which means executing the program line by line. However, as the program continues to run, the JIT compiler analyses the code and identifies hotspots, which are frequently executed portions of the code that could benefit from optimization.

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

**Ans.** In Java, access specifiers are keywords used to define the accessibility or visibility of classes, methods, variables, and constructors within a program. Java provides four access specifiers: i) public ii) protected iii) default iv) private

**Q6. What is a compiler in Java?**

**Ans.** It takes the Java source code as input and produces bytecode, which is a low-level representation of the program that can be executed by the Java Virtual Machine (JVM).

The Java compiler performs several tasks during the compilation process:

1. Syntax Checking: The compiler checks the syntax of the Java source code for errors and ensures it follows the rules of the Java programming language. It verifies that the code is written according to the specified grammar and structure.
2. Semantic Analysis: The compiler performs semantic analysis to ensure that the code follows the rules of Java semantics. It checks for things like variable declarations, type compatibility, and correctness of expressions.
3. Intermediate Code Generation: The compiler translates the Java source code into an intermediate representation known as bytecode. Bytecode is a platform-independent format that can be executed on any system with a compatible JVM.
4. Optimization: The compiler may apply various optimization techniques to improve the efficiency and performance of the generated bytecode. These optimizations can include constant folding, dead code elimination, and code rearrangement to make the program run faster or use fewer resources.
5. Error Reporting: If any errors or warnings are encountered during the compilation process, the compiler generates diagnostic messages indicating the nature and location of the issues. These messages help developers identify and fix the problems in their code.

Once the compilation process is completed, the resulting bytecode can be executed by the JVM. The JVM translates the bytecode into machine-specific instructions that can be understood and executed by the underlying hardware.

It's important to note that Java uses a two-step compilation process: first, the source code is compiled into bytecode by the Java compiler, and then the bytecode is interpreted or just-in-time (JIT) compiled by the JVM at runtime. This allows Java programs to be portable and run on any system with a compatible JVM, regardless of the underlying hardware or operating system.

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

**Ans.** In Java, variables are named memory locations used to store data during program execution. Java supports several types of variables based on their characteristics and usage. Here are the main types of variables in Java:

1. Local Variables: Local variables are declared within a method, constructor, or block and have limited scope. They are used to store temporary data and are only accessible within the block in which they are declared. Local variables must be explicitly initialized before they are used.
2. Instance Variables: Instance variables, also known as member variables or fields, are declared within a class but outside of any method, constructor, or block. They hold data specific to each instance (object) of the class. Instance variables are initialized with default values if not explicitly assigned, and their values persist as long as the object exists.
3. Class Variables: Class variables, also called static variables, are declared with the static keyword within a class but outside of any method, constructor, or block. They are associated with the class itself rather than with any particular instance of the class. Class variables are shared across all instances of the class and are accessible through the class name. They are initialized with default values if not explicitly assigned.
4. Parameters: Parameters are variables declared in a method or constructor's signature and are used to pass values into the method or constructor. They act as placeholders for the actual values that will be provided when the method or constructor is called. Parameter variables have a local scope within the method or constructor.

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

**Ans.** In Java, data types define the kinds of values that variables can hold. Java provides several built-in data types that can be categorized into two main categories: primitive types and reference types.

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

**Ans.** identifier is a name in java program. identifier can be class name, method name, variable name, label name.

**Q10. Explain the architecture of JVM**

**Ans.** The Java Virtual Machine (JVM) responsible for executing Java bytecode, which is a compiled form of Java source code. The JVM provides an abstraction layer between the Java programs and the underlying hardware and operating system.

The architecture of the JVM can be divided into several key components:

1. Class Loader: The Class Loader is responsible for loading Java class files into the JVM. It takes the compiled bytecode and transforms it into a binary format that can be executed by the JVM. The Class Loader also performs tasks like verifying the bytecode for security and resolving dependencies between classes.
2. Runtime Data Areas: The JVM maintains various runtime data areas during the execution of a Java program. These data areas include:
   * Method Area: It stores the class-level data, including the bytecode of methods, field and method references, and static variables.
   * Heap: The heap is a runtime data area used for dynamic memory allocation. It stores objects and arrays created during the execution of the program.
   * Java Stack: Each thread in the JVM has its own Java Stack, which is used for method invocations and local variable storage. It keeps track of method calls, parameters, and local variables.
   * PC Registers: PC (Program Counter) Registers store the address of the currently executing instruction.
   * Native Method Stack: It is used for native method invocations and is separate from the Java Stack.
3. Execution Engine: The Execution Engine is responsible for executing the compiled bytecode. It reads the bytecode instructions and performs the necessary actions. There are different types of execution engines, including:
   * Interpreter: The interpreter reads the bytecode line by line and executes it directly. It is simple but slower compared to other execution engines.
   * Just-In-Time (JIT) Compiler: The JIT compiler dynamically compiles frequently executed bytecode into machine code for improved performance. It analyses the bytecode and identifies hotspots to optimize the execution.
   * Ahead-of-Time (AOT) Compiler: In some JVM implementations, an AOT compiler can be used to compile the entire bytecode into machine code before execution. This eliminates the need for runtime bytecode interpretation or JIT compilation.
4. Native Method Interface (JNI): The JNI allows Java programs to interact with code written in other programming languages such as C or C++. It provides a way to call native methods and access native libraries from within Java code.
5. Java Native Library: The Java Native Library consists of native libraries and files required by the JVM. It includes the standard Java class libraries, as well as platform-specific libraries necessary for interacting with the underlying operating system.

The JVM architecture provides a portable and platform-independent environment for running Java programs. It shields the Java applications from the underlying hardware and operating system, allowing them to run consistently across different platforms as long as a compatible JVM is available.