Q1. What is the difference between Compiler and Interpreter?

Answer:

Compiler

A compiler is a translator that produces an output of low-level language (like an assembly or machine language) by taking an input of high-level language. It is basically a computer program used to transform codes written in a programming language into machine code (human-readable code to a binary 0- and 1-bits language for a computer processor to understand). The computer then processes the machine code for performing the corresponding tasks.

- Compilers check all types of errors, limits, and ranges. Thus, it's more intelligent.
- The run time of its program is longer, and it occupies more memory.

Interpreter

It is a program that functions for the translation of a programming language into a comprehensible one. It is a computer program used for converting high-level program statements into machine codes. It includes pre-compiled code, source code, and scripts.

- An interpreter translates only one statement at a time of the program.
- They create an exe of the programming language before the program runs.

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

Answer:

JDK

JDK is an abbreviation for Java Development Kit. It is an environment of software development used for developing applets and Java applications. JDK has a physical existence, and it contains JRE + development tools. One can easily install more than one version of JDK on the same computer. The Java developers can make use of it on macOS, Windows, Linux, and Solaris. JDK assists them in coding and running the Java programs.

It is an implementation of any of the given Java Platforms that the Oracle Corporation released:

- Micro Edition
- Enterprise Edition
- Standard Edition

The JDK consists of a private JVM (Java Virtual Machine) along with a few other resources, java (a loader/interpreter), like javac (a compiler), Javadoc (a documentation generator), jar (an archiver), etc., for completing the process of Java application development.

JRE

JRE stands for Java Runtime Environment- also written as Java RTE. It is a set of software tools designed for running other software. It is an implementation of JVM, and JRE provides a runtime environment. In short, a user needs JRE to run any Java program. If not a programmer, the user doesn't need to install the JDK- JRE alone will help run the Java programs.

All the versions of JDK come bundled up with the JRE (Java Runtime Environment). This way, a user doesn't have to download and install JRE on their PC separately. The JRE also exists physically. It consists of a library set + a few more files that the JVM (Java Virtual Machine) deploys at the runtime.

JVM

JVM stands for Java Virtual Machine. It provides a runtime environment for driving Java applications or code. JVM is an abstract machine that converts the Java bytecode into a machine language. It is also capable of running the programs written by programmers in other languages (compiled to the Java bytecode). The JVM is also known as a virtual machine as it does not exist physically.

JVM is essentially a part of the JRE (Java Run Environment). You cannot separately download and install it. You first need to install the JRE to install the JVM. It is available for many software and hardware platforms. In various distinct programming languages, the compiler functions to produce machine code for specific systems. However, only the Java compiler produces code for a virtual machine- also known as JVM.

All three, JDK, JRE, and JVM, are dependent. It is because each Operating System's (OS) condition is different from one another. But Java is independent of the platform. The JVM has three notions: *implementation, instance,* and *specification*.

JVM primarily performs the following tasks:

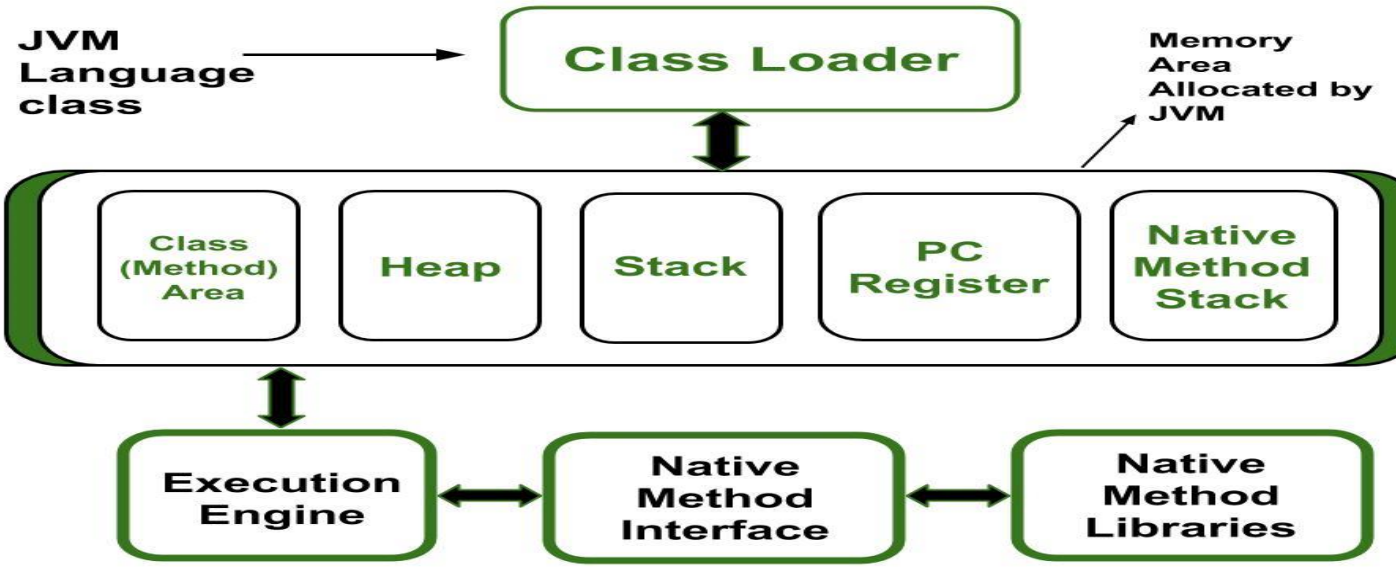
- Provides runtime environment.
- Verifies code.
- Loads code.
- Executes code.

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

Answer:

Types of Memory Areas Allocated By the JVM:

All these functions take different forms of memory structure. The **memory in the JVM is divided into 5 different parts**:



1. Class (Method) Area
2. Heap

3. Stack
4. Program Counter Register
5. Native Method Stack

Q4. What is JIT compiler?

Answer:

JIT in Java is an integral part of the **JVM**. It accelerates execution performance many times over the previous level. In other words, it is a long-running, computer-intensive program that provides the best performance environment. It optimizes the performance of the Java application at compile or run time.

The JIT compilation includes two approaches **AOT** (Ahead-of-Time compilation) and **interpretation** to translate code into machine code. AOT compiler compiles the code into a native machine language (the same as the normal compiler). It transforms the bytecode of a VM into the machine code. The following optimizations are done by the JIT compilers:

- Method In-lining
- Local Optimizations
- Control Flow Optimizations
- Constant Folding
- Dead Code Elimination
- Global Optimizations
- Heuristics for optimizing call sites.

Q5. What are the various access specifiers in Java?

Answer:

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is 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.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Q6. What is a compiler in Java?

Answer:

Java Compiler and Interpreter are the most fundamental tools in Java language that programmers use during programming.
A compiler in Java is a computer program that is used for compiling Java programs. It is platform-independent. It converts (translates) source code (.java file) into bytecode (.class file).

In other words, the compiler (javac.exe) generates bytecode during the compilation process.

A bytecode is a binary code that is understood and interpreted by Java Virtual Machine (JVM) on the underlying operating system. It is not similar to machine code.

It is unreadable by humans because it is composed of numbers that are the only language that computers understand.

Java compiler can be activated by using “Javac.exe” command from the command prompt.

Q7. Explain the types of variables in Java?

Answer:

There are three types of variables in Java:

- local variable
- instance variable
- static variable

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Q8. What are the Datatypes in Java?

Answer:

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Q9. What are the identifiers in java?

Answer:

Identifiers in Java are symbolic names used for identification. They can be a class name, variable name, method name, package name, constant name, and more. However, In Java, There are some reserved words that can not be used as an identifier.

Example:

```
public class MainClass {
public static void main(String[] args) {
int var1 = 99;
double var2 = 2.0;
System.out.println("Hello World!");
}
}
```

Example Java Code Snippet

Identifiers: Below is the list of identifiers that are present in the above sample code.

- MainClass (Class name)
- main (Method name)
- String (Predefined Class name)
- args (String variable name)
- var1 (integer variable name)
- var2 (double variable name)
- System(Predefined Class name)
- out(Variable name)
- println (Method name)

Q10.Explain the architecture of JVM

Answer:

JVM (Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that actually calls the **main** method present in a java code. JVM is a part of JRE (Java Runtime Environment).
Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM.

When we compile a *.java* file, *.class* files (contains bytecode) with the same class names present in *.java* file are generated by the Java compiler. This *.class* file goes into various steps when we run it. These steps together describe the whole JVM.

- **Classloader** − Loads the class file into the JVM.
- **Class Area** − Storage areas for a class elements structure like fields, method data, code of method etc.
- **Heap** − Runtime storage allocation for objects.

- **Stack** − Storage for local variables and partial results. A stack contains frames and allocates one for each thread. Once a thread gets completed, this frame also gets destroyed. It also plays roles in method invocation and returns.
- **PC Registers** − Program Counter Registers contains the address of an instruction that JVM is currently executing.
- **Execution Engine** − It has a virtual processor, interpreter to interpret bytecode instructions one by one and a JIT, just in time compiler.
- **Native method stack** − It contains all the native methods used by the application.