# **Assignment 1**

Q1. What is the difference between Compiler and Interpreter?

## Compiler

The Compiler is a translator which takes input i.e., High-Level Language, and produces an output of low-level language i.e., machine or assembly language. The work of a Compiler is to transform the codes written in the programming language into machine code (format of 0s and 1s) so that computers can understand.

* A compiler is more intelligent than an assembler, it checks all kinds of limits, ranges, errors, etc.
* But its program run time is more and occupies a larger part of memory. It has a slow speed because a compiler goes through the entire program and then translates the entire program into machine codes.

## Interpreter

An Interpreter is a program that translates a programming language into a comprehensible language. The interpreter converts high-level language to an intermediate language. It contains pre-compiled code, source code, etc.

* It translates only one statement of the program at a time.
* Interpreters more often than not are smaller than compilers.

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

|  |  |  |
| --- | --- | --- |
| **JDK** | **JRE** | **JVM** |
| The full form of JDK is Java Development Kit. | The full form of JRE is Java Runtime Environment. | The full form of JVM is Java Virtual Machine. |
| JDK is a software development kit to develop applications in Java. | It is a software bundle which provides Java class libraries with necessary components to run Java code. | JVM executes Java byte code and provides an environment for executing it. |
| JDK is platform dependent. | JRE is also platform dependent. | JVM is highly platform dependent. |
| It contains tools for developing, debugging, and monitoring java code. | It contains class libraries and other supporting files that JVM requires to execute the program. | Software development tools are not included in JVM. |
| It is the superset of JRE | It is the subset of JDK. | JVM is a subset of JRE. |
| The JDK enables developers to create Java programs that can be executed and run by the JRE and JVM. | The JRE is the part of Java that creates the JVM. | It is the Java platform component that executes source code. |
|  |  |  |

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

### 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

#### 1. Class (Method) Area

The class method area is the memory block that stores the class code, variable code(static variable, runtime constant), method code, and the constructor of a Java program. (Here method means the function which is written inside the class). It stores class-level data of every class such as the runtime constant pool, field and method data, the code for methods.

#### 2. Heap

The Heap area is the memory block where objects are created, or objects are stored. Heap memory allocates memory for class interfaces and arrays (an array is an object). It is used to allocate memory to objects at run time

#### 3. Stack

Each thread has a private JVM stack, created at the same time as the thread. It is used to store data and partial results which will be needed while returning value for method and performing dynamic linking.

Java Stack stores frames and a new frame is created each time at every invocation of the method. A frame is destroyed when its method invocation completes

#### 4. Program Counter Register:

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

#### 5. Native method Stacks:

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

Q4. What is JIT compiler?

The Just-In-Time (JIT) compiler is a component of the runtime environment that improves the performance of Java™ applications by compiling bytecodes to native machine code at run time.

Java programs consists of classes, which contain platform-neutral bytecodes that can be interpreted by a JVM on many different computer architectures. At run time, the JVM loads the class files, determines the semantics of each individual bytecode, and performs the appropriate computation. The additional processor and memory usage during interpretation means that a Java application performs more slowly than a native application. The JIT compiler helps improve the performance of Java programs by compiling bytecodes into native machine code at run time.

The JIT compiler is enabled by default. When a method has been compiled, the JVM calls the compiled code of that method directly instead of interpreting it. Theoretically, if compilation did not require processor time and memory usage, compiling every method could allow the speed of the Java program to approach that of a native application.

Q5. What are the various access specifiers in Java?

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

**Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

**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.

**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.

**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?

## Compiler

The Compiler is a translator which takes input i.e., High-Level Language, and produces an output of low-level language i.e., machine or assembly language. The work of a Compiler is to transform the codes written in the programming language into machine code (format of 0s and 1s) so that computers can understand.

* A compiler is more intelligent than an assembler, it checks all kinds of limits, ranges, errors, etc.
* But its program run time is more and occupies a larger part of memory. It has a slow speed because a compiler goes through the entire program and then translates the entire program into machine codes.

Q7. Explain the types of variables in Java?

There are three types of variables in [Java](https://www.javatpoint.com/java-tutorial)

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](https://www.javatpoint.com/static-keyword-in-java).

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.

### Example

**public** **class** A

{

**static** **int** m=100;//static variable

**void** method()

{

**int** n=90;//local variable

}

**public** **static** **void** main(String args[])

{

**int** data=50;//instance variable

}

}

Q8. What are the Datatypes in Java?

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

**Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.

**Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Default Value** | **Default size** |
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

Q9. What are the identifiers in java?

In programming languages, identifiers are used for identification purposes. In Java, an identifier can be a class name, method name, variable name, or label.

For example:   
 public class Test  
{  
 public static void main (String [] args)  
 {  
 int a = 20;  
 }  
}

In the above java code, we have 5 identifiers namely :

**Test:** class name.

**main:** method name.

**String:** predefined class name.

**args**: variable name.

**a:** variable name.

Q10. Explain the architecture of JVM?

## JVM Architecture

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.

### 1) Classloader

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

**Bootstrap ClassLoader**: This is the first classloader which is the super class of Extension classloader. It loads the *rt.jar* file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.

**Extension ClassLoader**: This is the child classloader of Bootstrap and parent classloader of System classloader. It loads the jar files located inside *$JAVA\_HOME/jre/lib/ext* directory.

**System/Application ClassLoader**: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

### 2) Class (Method) Area

Class (Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

### 3) Heap

It is the runtime data area in which objects are allocated.

### 4) Stack

Java Stack stores frames. It holds local variables and partial results and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation is complete.

### 5) Program Counter Register

PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

### 6) Native Method Stack

It contains all the native methods used in the application.

### 7) Execution Engine

It contains:

**A virtual processor**

**Interpreter:** Read bytecode stream then execute the instructions.

**Just-In-Time (JIT) compiler:** It is used to improve performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

### 8) Java Native Interface

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.