Q1) Explain the components of the JDK.

Ans:

JDK: It stands for java development kit and the JDK is a software development environment which is used to develop java application and applets.

here are some java platforms which is released by oracle corporation:

--Java SE

--Java EE

--Java ME

--Java FX

the jdk contains a private jvm and few other resources such as interpreter, compiler an archiever.

COMPONENTS OF JDK

_____

1. appletviewer: this tool is used to run and debug java applets without a web browser.

2. apt: it is an annotation processing tool

3. java: The loader for Java applications. This tool is an interpreter and can interpret the class files generated by the javac compiler. Now a single launcher is used for both development and deployment. The old deployment launcher, jre, no longer comes with Sun JDK, and instead it has been replaced by this new java loader.

4. javac: It specifies the Java compiler, which converts source code into Java bytecode.

5. javap: the class file disassembler

Q. 2) Differentiate between JDK, JVM, and JRE.

Ans:

	JDK	JRE	JVM
Full Form	Java Development Kit	Java Runtime Environment	Java Virtual Machine
Definition	It's a software development kit used to develop applications in Java.	It's a software bundle that provides Java class libraries with the necessary components to run Java code.	It's an abstract machine that provides and environment to run and execute Java byte code.
What it Contains	It contains tools for developing, debugging, and monitoring java code.	It contains class libraries and other supporting files that the JVM requires to execute the program.	Software development tools are not included in the JVM.
What it Enables	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 the JDK that creates the JVM.	It is the Java platform component that executes source code.
Architecture	Superset	Subset of JDK	Subset of JRE

Q3. What is the role of the JVM in Java? & How does the JVM execute Java code?

Ans: JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent)

It is:

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

The JVM performs following operation:

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

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

Q4. Explain the memory management system of the JVM.

Q6. Describe the architecture of the JVM.

Ans.

JVM stands for Java Virtual Machine. It is a crucial component of the Java platform and serves as an execution environment for Java bytecode. The JVM is responsible for interpreting or compiling Java bytecode into machine code that can be executed by the underlying operating system and hardware. It provides platform independence by allowing Java programs to run on any system that has a compatible JVM implementation. The JVM also includes various runtime services, such as memory management, garbage collection, and exception handling, to ensure efficient and secure execution of Java programs.

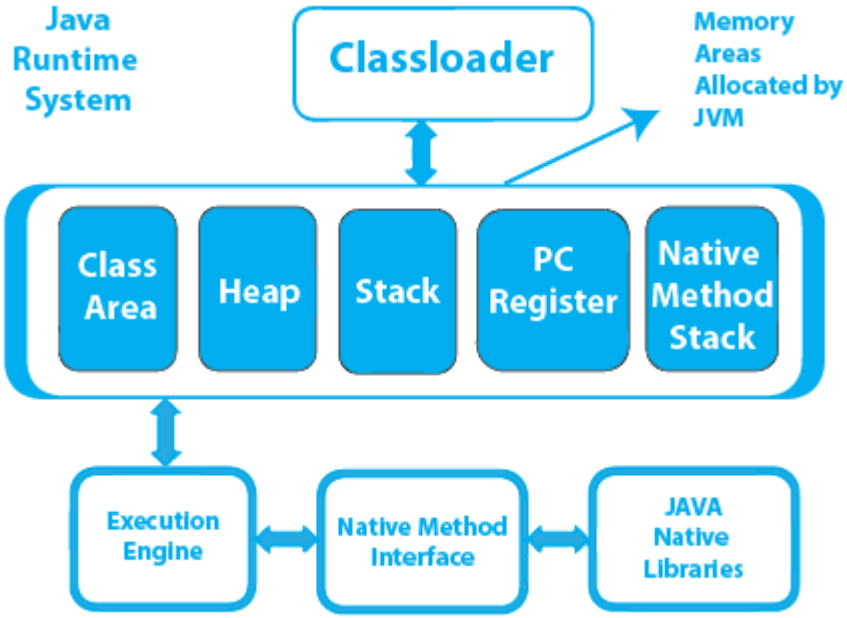
What does JVM do?

The JVM (Java Virtual Machine) performs several important tasks, including:

- **Execution of Java Bytecode:** The JVM interprets or compiles Java bytecode into machine code that can be executed by the underlying operating system and hardware.
- **Memory Management:** The JVM manages memory allocation and deallocation for Java objects. It automatically handles memory allocation and deallocation through techniques like garbage collection, which frees up memory occupied by objects that are no longer needed.
- **Garbage Collection:** The JVM automatically identifies and frees up memory occupied by objects that are no longer referenced or in use. This helps prevent memory leaks and ensures efficient memory utilization.
- **Just-In-Time (JIT) Compilation:** The JVM employs a Just-In-Time compiler to dynamically analyze and optimize sections of Java bytecode that are frequently executed. JIT compilation can significantly improve the performance of Java applications by translating bytecode into machine code at runtime.
- **Exception Handling:** The JVM provides robust exception-handling mechanisms to catch and handle runtime errors, ensuring that Java programs can gracefully handle unexpected situations.
- **Security:** The JVM enforces various security measures, such as bytecode verification, to ensure that Java programs cannot access unauthorized resources or perform malicious activities.
- **Class Loading and Dynamic Linking:** The JVM dynamically loads and links Java classes as they are referenced, allowing for on-demand class loading and the use of dynamic libraries.

What is JVM Architecture?

The JVM (Java Virtual Machine) architecture consists of several components that work together to execute Java programs. Here is a high-level overview of the JVM architecture:



- **Class Loader:** The Class Loader component is responsible for loading Java class files into the JVM at runtime. It performs tasks such as locating and reading class files, verifying their bytecode, and defining the classes within the JVM.
- **Runtime Data Area:** The Runtime Data Area is the memory area where the JVM manages data during program execution. It consists of several components:
- **Method Area:** The Method Area stores class-level data, including the bytecode of methods, constant pool, static variables, and method metadata.
- **Heap:** The Heap is the runtime data area where objects are allocated. It is divided into two parts: Young Generation and Old Generation. The Young Generation is further divided into Eden Space, Survivor Space, and other survivor spaces, while the Old Generation holds long-lived objects.
- **Java Stack:** Each thread in the JVM has a Java Stack that stores method-specific data, including local variables, method arguments, and method invocation records. It also manages method calls and returns.
- **Native Method Stack:** The Native Method Stack holds native method-specific data, similar to the Java Stack. It is used for executing native (non-Java) methods.
- **Program Counter:** The Program Counter (PC) keeps track of the currently executing bytecode instruction.
- **Execution Engine:** The Execution Engine executes Java bytecode. It can employ different techniques for bytecode execution, such as interpretation, Just-In-Time (JIT) compilation, or a combination of both. The execution engine interacts with the Runtime Data Area and coordinates the execution of Java programs.

- **Native Method Interface (JNI):** The JNI allows Java programs to interact with native code written in languages like C or C++. It provides a mechanism for Java code to call native methods and access native libraries.
- **JVM Languages:** The JVM architecture supports languages other than Java through additional compilers and runtime support. Examples include Kotlin, Scala, and Groovy, which can all be compiled into bytecode and executed on the JVM.

Q7. How does Java achieve platform independence through the JVM?

Ans:

Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is intended to let application developers write once and run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without recompilation. Java was first released in 1995 and is widely used for developing applications for desktop, web, and mobile devices. Java is known for its simplicity, robustness, and security features, making it a popular choice for enterprise-level applications.

One of the most prominent features of Java is platform independent. Platform-independent means the Java code that has been compiled generates compiled code or the byte code, and this byte code can run on all of the operating systems provided they have JVM installed in it. Basically, a program is a set of instructions that have been written in a human-readable language. Note that programs written by humans are not understood by machines. Here compilers come into the picture. A compiler compiles the program or source code into a format that can be easily understood by machines.

Therefore, one can say that a compiler is a translator that does the translation of code from human-readable format to executable code. The executable code may be run directly by the machine or maybe an intermediate representation that is run on the virtual machine. In Java, that intermediate representation is known as the Java Byte Code.

Step By Step Code Execution in Java

- The developer or programmer writes the code in a human-readable format. Java compiler compiles it into byte code. Note that Byte code is not the native code for the machines. (Unlike C++/C compiler)
- Byte code is not the machine executable code and hence, requires an interpreter to interpret it. The interpreter is the Java Virtual Machine (JVM) that executes the Byte code.
- Eventually, the program executes and shows the output.

In C or C++ compiler compiles the code to generate the .exe file that has been executed by the machine on which the compiler is installed. However, in the case of Java, instead of .exe file, a .class file or Byte code is generated.

Java is Platform Independent but not JVM

It is important to note that JVM is platform dependent. So, for the Windows operating system, we have the JVM specific to Windows. For the Macintosh operating system, we have the Macintosh-specific operating system, and the same concept is applied to the Linux operating system too. The same is evident when we go to the websites to download the JDK (JVM is part of JDK). A list of operating system-specific JDKs is shown, and the user has to decide which one is suitable for accomplishing the task.