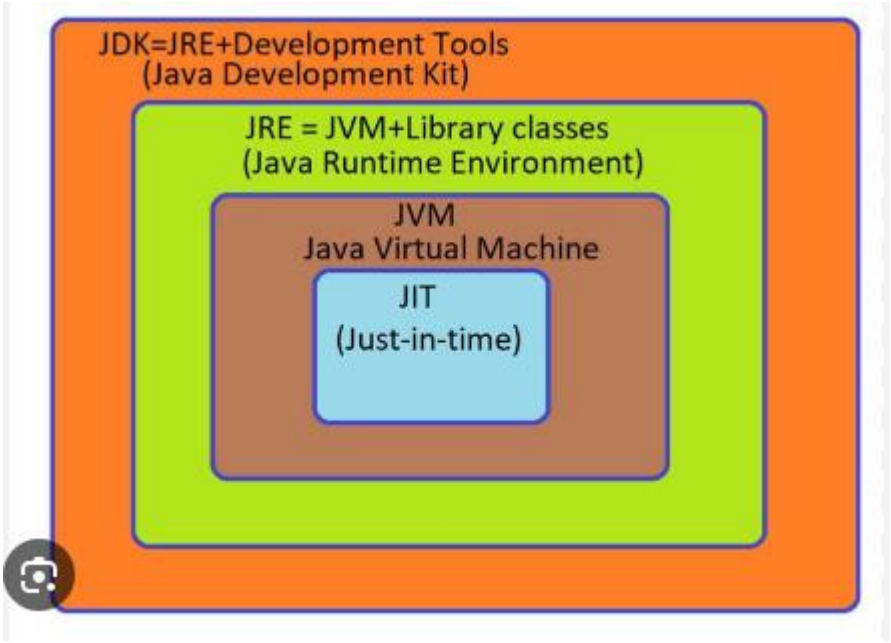
Theoretical Assignment

Q1) Explain the components of the JDK.

Ans: The Java Development Kit (JDK) is a software development environment used to develop Java applications and applets. It contains JRE and several development tools, an interpreter/ loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) accompanied with another tool.



Q2) Differentiate between JDK, JVM, and JRE.

Difference Between JDK, JRE, and JVM

Parameter	JDK	JRE	JVM
Full-Form	The JDK is an abbreviation for Java Development Kit.	The JRE is an abbreviation for Java Runtime Environment.	The JVM is an abbreviation for Java Virtual Machine.
Definition	The JDK (Java Development Kit) is a software development kit that develops applications in Java. Along with JRE, the JDK also consists of various development tools (Java Debugger, JavaDoc, compilers, etc.)	The Java Runtime Environment (JRE) is an implementation of JVM. It is a type of software package that provides class libraries of Java, JVM, and various other components for running the applications written in Java programming.	The Java Virtual Machine (JVM) is a platform-independent abstract machine that has three notions in the form of specifications. This document describes the requirement of JVM implementation.

Functionality	The JDK primarily assists in executing codes. It primarily functions in development.	JRE has a major responsibility for creating an environment for the execution of code.	JVM specifies all of the implementations. It is responsible for providing all of these implementations to the JRE.
Platform Dependency	The JDK is platform-dependent. It means that for every different platform, you require a different JDK.	JRE, just like JDK, is also platform-dependent. It means that for every different platform, you require a different JRE.	The JVM is platform-independent. It means that you won't require a different JVM for every different platform.
Tools	Since JDK is primarily responsible for the development, it consists of various tools for debugging, monitoring, and developing java applications.	JRE, on the other hand, does not consist of any tool- like a debugger, compiler, etc. It rather contains various supporting files for JVM, and the class libraries that help JVM in running the program.	JVM does not consist of any tools for software development.
Implementation	**JDK** = Development Tools + JRE (Java Runtime Environment)	**JRE** = Libraries for running the application + JVM (Java Virtual Machine)	**JVM** = Only the runtime environment that helps in executing the Java bytecode.
Why Use It?	Why use JDK? Some crucial reasons to use JDK are: • It consists of various tools required for writing Java programs. • JDK also contains JRE for executing Java programs. • It includes an Appletviewer, Java application launcher, compiler, etc. • The compiler helps in converting the code written in Java into bytecodes. • The Java application launcher helps in opening a JRE. It then loads all of the necessary details and then executes all of its main methods.	Why use JRE? Some crucial reasons to use JRE are: • If a user wants to run the Java applets, then they must install JRE on their system. • The JRE consists of class libraries along with JVM and its supporting files. It has no other tools like a compiler or a debugger for Java development. • JRE uses crucial package classes like util, math, awt, lang, and various runtime libraries.	Why use JVM? Some crucial reasons to use JVM are: • It provides its users with a platform-independent way for executing the Java source code. • JVM consists of various tools, libraries, and multiple frameworks. • The JVM also comes with a Just-in-Time (JIT) compiler for converting the Java source code into a low-level machine language. Thus, it ultimately runs faster than any regular application. • Once you run the Java program, you can run JVM on any given platform to save your time.
Features	Features of JDK	Features of JRE	Features of JVM Here are a few crucial

	• Here are a few crucial features of JDK: • It has all the features that JRE does. • JDK enables a user to handle multiple extensions in only one catch block. • It basically provides an environment for developing and executing the Java source code. • It has various development tools like the debugger, compiler, etc. • One can use the Diamond operator to specify a generic interface in place of writing the exact one. • Any user can easily install JDK on Unix, Mac, and Windows OS (Operating Systems).	• Here are a few crucial features of JRE: • It is a set of tools that actually helps the JVM to run. • The JRE also consists of deployment technology. It includes Java Plug-in and Java Web Start as well. • A developer can easily run a source code in JRE. But it does not allow them to write and compile the concerned Java program. • JRE also contains various integration libraries like the JDBC (Java Database Connectivity), JNDI (Java Naming and Directory Interface), RMI (Remote Method Invocation), and many more. • It consists of the JVM and virtual machine client for Java HotSpot.	features of JVM: • The JVM enables a user to run applications on their device or in a cloud environment. • It helps in converting the bytecode into machine-specific code. • JVM also provides some basic Java functions, such as garbage collection, security, memory management, and many more. • It uses a library along with the files given by JRE (Java Runtime Environment) for running the program. • Both JRE and JDK contain JVM. • It is easily customizable. For instance, a user can feasibly allocate a maximum and minimum memory to it.

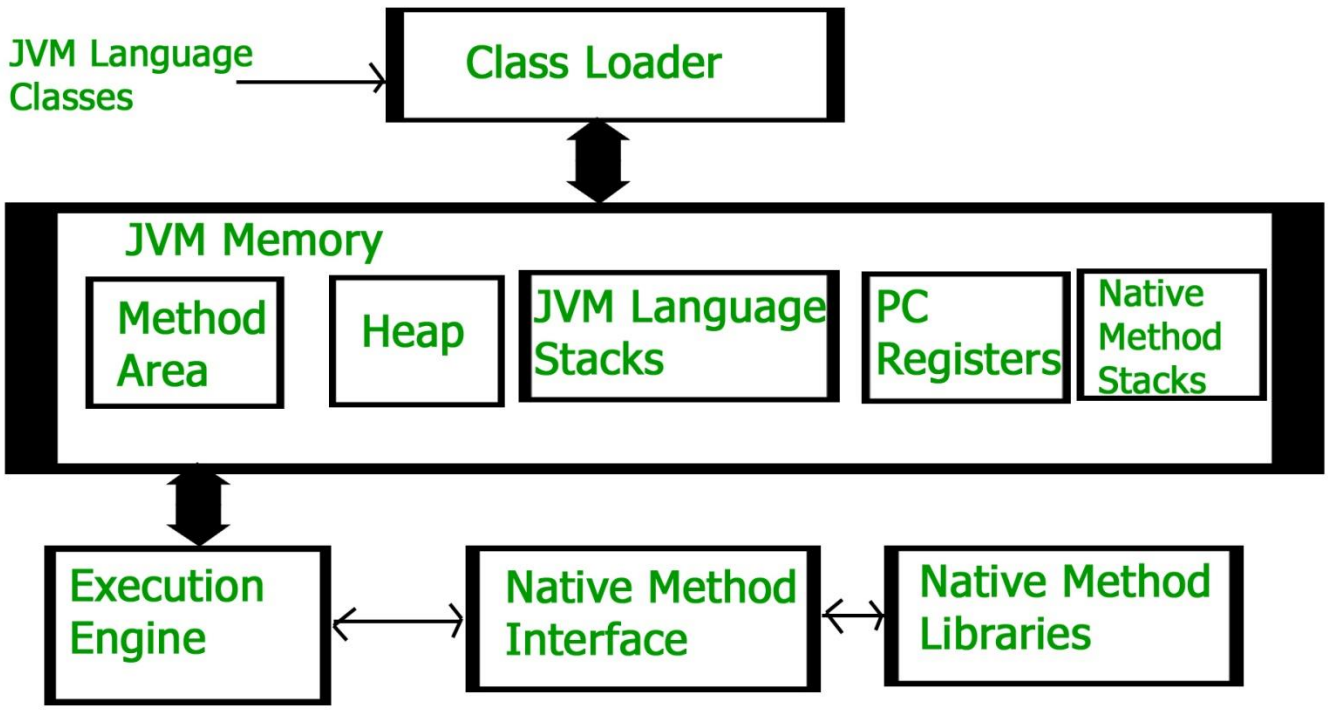
Q3) (i) What is the role of the JVM in Java?

Java Virtual Machine Threads

- When a Java application starts, several threads are automatically created and started by the JVM to handle various tasks and provide runtime
  - **Main Thread**
    - It executes the main() method .
  - **Reference Handler Thread**
    - It is responsible for handling reference objects that are enqueued by the garbage collector.
  - **Finalizer Thread**
    - It is responsible for running the finalize() method of objects that are eligible for finalization.
  - **Signal Dispatcher Thread**
    - is responsible for handling operating system signals, such as SIGINT (interrupt signal) and SIGTERM (termination signal).
  - **Compiler Threads**
    - The JVM may create and start one or more compiler threads to compile Java bytecode into native machine code.
  - **Garbage Collector Threads**
    - To perform garbage collection.
  - **Other System Threads**
- jstack is command line tool which is used to take thread dump.
  - First get the process id:
    - jcmd //or
    - ps -ef | grep java
  - Use process id with jstack
    - jstack pid

Q3 (ii) How does the JVM execute Java code?

Ans: When we compile a *.java* file, *.class* files(contains byte-code) 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.



Q4) Explain the memory management system of the JVM.

Ans: In Java, memory management is an automatic process that is managed by the

Java Virtual Machine (JVM), and one that does not need explicit intervention. Java, being a block-structured language, uses a model where its memory is divided into two main types: stack and heap.

Q5(i) What is the JIT compiler and its role in the JVM?

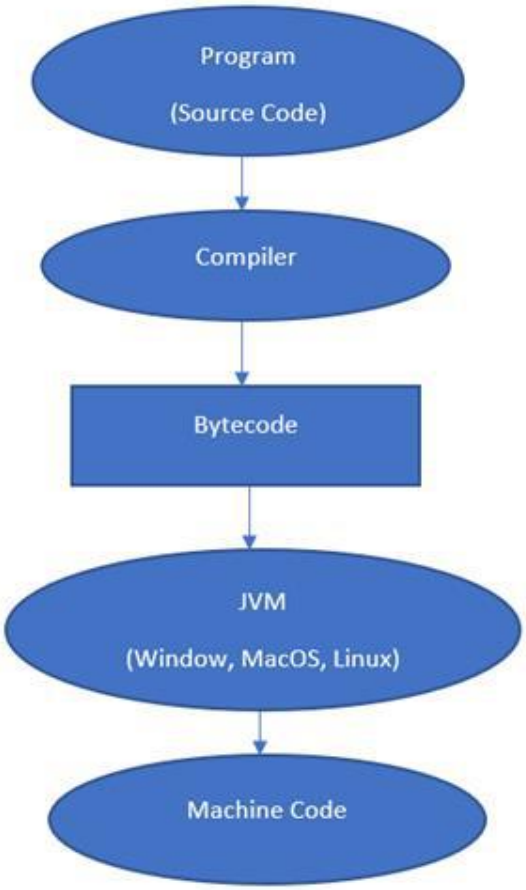
Ans: The Just-In-Time (JIT) compiler is a key component of the Java Virtual Machine (JVM) that plays a crucial role in improving the performance of Java applications.
- The JIT compiler is a part of the JVM responsible for translating Java bytecode into native machine code at runtime.
- It takes the bytecode generated by the Java compiler and converts it into native machine code that is executable by the underlying hardware.

Q5 (ii) What is the bytecode and why is it important for Java?

Ans: ByteCode: Byte Code can be defined as an intermediate code generated by the compiler after the compilation of source code(JAVA Program). This intermediate code makes Java a platform-independent language.

Byte Code generation:

Compiler converts the source code or the Java program into the Byte Code(or machine code), and secondly, the Interpreter executes the byte code on the system. The Interpreter can also be called JVM(Java Virtual Machine). The byte code is the common piece between the compiler(which creates it) and the Interpreter (which runs it).



Q 6) Describe the architecture of the JVM.

JVM Architecture

Java Architecture is internally basically divided into seven segments. It contains a classloader, memory area, execution, etc.

- **Classloader:** it's a subsystem of JVM used to load class files. When we execute the code, it's loaded by the classloader. There are built-in classloaders in Java.

1. **Bootstrap ClassLoader:** This is the first classloader, the superclass of the Extension classloader. It loads the rt.jar file, which contains all files of Java Standard Edition like Java.lang package classes, java.net package classes.

2. **Extension ClassLoader:** This is the child classloader of Bootstrap and parent classloader of System classloader.

3. **System/Application Classloader:** It loads the class files from classpath. It is also known as the Application classloader.

- **Class(Method)Area:** Class (Method)Area stores per-class structures such as the constant runtime pool, field and method data, and the form code.
- **Heap:** It is the runtime method where objects are allocated.
- **Stack:** It stores frames and holds local variables and partial results and plays a part in method invocation and return.

- **Program Counter Register:** This register contains the address of the Java Virtual machine instruction, which is currently being executed.

- **Native Method Stack:** It has all the native methods applicable in the application.

- **Execution Engine:** This consists of three parts:
1. **A virtual processor**
2. **Interpreter**
3. **Just-In-Time(JIT) compiler**

- **Java Native Interface:** It is a framework that facilitates an interface to communicate with another application coded in other languages like C++, C, Assembly, etc. Java uses this framework to send output to the console to interact with OS libraries.

Q 7) How does Java achieve platform independence through the JVM?

Ans: The meaning of Java platform-independent is that the Java compiled code(byte code) can run on all operating systems. A program is written in a language that is a human-readable language. It may contain words, phrases, etc which the machine does not understand. For the source code to be understood by the machine, it needs to be in a language understood by machines, typically a machine-level language. So, here comes the role of a compiler. The compiler converts the high-level language (human language) into a format understood by the machines.
Therefore, a compiler is a program that translates the source code for another program from a programming language into executable code. This executable code may be a sequence of machine instructions that can be executed by the CPU directly, or it may be an intermediate representation that is interpreted by a virtual machine. This intermediate representation in Java is the **Java Byte Code.**

Q 8(i) What is the significance of the class loader in Java?
Ans: In Java, the class loader plays a crucial role in the Java Virtual Machine (JVM) environment.

1. Dynamic Loading: Class loaders are responsible for dynamically loading Java classes into the JVM as they are referenced by the running Java program. This dynamic loading mechanism allows Java applications to load classes on-demand, which improves memory efficiency and reduces startup time.

4. Hierarchical Structure: Java class loaders follow a hierarchical structure, allowing them to delegate class loading responsibilities to parent class loaders. This hierarchical arrangement enables classes to be loaded from various sources such as the file system, network, or custom repositories.
5. Isolation and Security: Class loaders provide a level of isolation between different Java applications and components. Each class loader has its own namespace, ensuring that classes loaded by one class loader do not interfere with classes loaded by other class loaders. This isolation enhances security by preventing unauthorized access to classes and resources.

6. Customization and Extensibility: Java applications can implement custom class loaders to extend or modify the default class loading behavior. Custom class loaders are often used in frameworks and application servers to implement features like hot swapping, dynamic module loading, and runtime bytecode generation.
7. Class Loading Mechanism: The class loading process consists of three steps: loading, linking, and initialization. The class loader loads the bytecode of a class into memory, performs bytecode verification, resolves symbolic references, and initializes the class before it can be used.
8. Dynamic Updates: Class loaders support dynamic updates and reloading of classes at runtime. This feature is particularly useful in long-running applications that require updates or patches without downtime.

Q 8(ii) What is the process of garbage collection in Java?

Ans: The process of garbage collection in Java is the automatic process of reclaiming memory occupied by objects that are no longer reachable or in use by the program. The process of garbage collection in Java typically involves the following steps:

1) Identification of Unreachable objects
2) Marking Phase
3) Sweeping phase
4) Compaction
5) Finalization.