Page No.
Date

Q1. Explain the component of the JDK

⇒ 1. Java compiler (Javac) :-

The java compiler is a key component of JDK that transforms java source code (.java files) into bytecode (.class files). the generated bytecode can be executed on any problem with JVM installed, ensuring the "write once, run anywhere" philosophy of java

2. Java virtual machine (JVM) :-

The java virtual machine is the runtime engine that execute java bytecode. It provide an abstraction layer betn the java application & underlying system.

3. Java Runtime Environment (JRE) :-

The java runtime Environment (JRE) is a ~~[illegible]~~ subset of JDK that includes the JVM and essential class libraries.

4. Java API libraries :-

Java API libraries provide a vast collection of pre-defined classes and methods that simplify common programming task

5. Java Debugger :-

The java debugger (Jdb) is a powerful tool for debugging java applications it develope to set breakpoints, inspects variables and step through the code to identify and fix issues during development.

6. Java Documentation generator :-

Java documentation automatically generates documentation making it easier for developers to understand & use the classes & methods provided by the application's codebase

7. Additional utilities:-

Apart from the major components mentioned above JDK also includes various utilities that facilitate develop. tasks.

Q2. Differentiate bet^n JDK, JVM and JRE.

=>

JDK	JRE	JVM.
stands for Java development kit.	stands for Java runtime environment	stands for Java virtual machine
It primarily insists in executing codes & functions in development	major responsibility for creating an environment for the execution of code	specifies all the implementations it is responsible for providing all of the implementation to JRE
JDK is platform dependent. it means that for every different platform, you require different JDK.	platform dependent for every different platform, you require a different JRE	platform independent won't require a different JVM for every different platform.
JDK = Development tools + JRE (Java runtime environment)	JRE = Libraries for running application + JVM	only the runtime environment help in execution.

Page No.
Date

3) What is the role of the JVM in java? &
How does the JVM execute java code.

=) JVM is crucial in java as it serves as runtime enviroment for executing java code. It's primary role includes interpreting or compiling bytecode, managing memory, handling exceptions, ensuring platform independence & providing security features such as byte code verification & sandboxing. essentially JVM allows java program to run on any device or operating system that has a compitable JVM implementation.

JVM executes Javacode in steps:-

i) source code is compiled by java compiler into byte code

ii) jvm loads bytecode classes dynamically as needed during execution

iii) JVM verifies bytecode to ensure it is java language specification & does not violate security constraints.

iv) JVM interprets bytecode instructions or use JIT compiler to translate bytecode into machine code.

v) JVM manages memory allocation, de-allocation & garbage collection to ensure efficient memory usage.

vi) Exception handling: JVM handles exception & runtime errors gracefully, allowing java prog. to recover from unexpected situations

vii) JVM implementations does optimization like JIT compilation to improve performance.

Q4. Memory management system of JVM.

⇒ Method area.

- It has 5 parts
- Method area/Heap area -
  - Load all class information. JVM has only one method area & heap area.
  - It is also called as thread ~~method~~ memory

stacks :-

- keep method information, local variables.
- A separate runtime stack is created for every thread.
- All details are stored here until completion of method.

-PC Register :-

- It holds information about next execution.
- It stores address of currently executing JVM instruction.
- separate PC register is created for every thread.

- Native method stack :-
  - Thread creates this kind of memory and threads is at a whole new level.

- Native method Area :-
  - This is stack that can support native method that are written in different language

Date / /

Q5 what are the JIT compiler and its role in the JVM ?. what is the bytecode and why is it important for java ?.

⇒ The JIT compiler is a key component of the JVM responsible for optimizing the execution of java bytecode at runtime.

Roles of JIT compiler :-

- when a java program is compiled, its translated into platform - independent. bytecode which is executed by the JVM.
- The bytecode is platform independent, meaning it can run on any system with a compatible JVM.
- executing bytecode directly can be less efficient than executing native machine code.

Bytecode :-

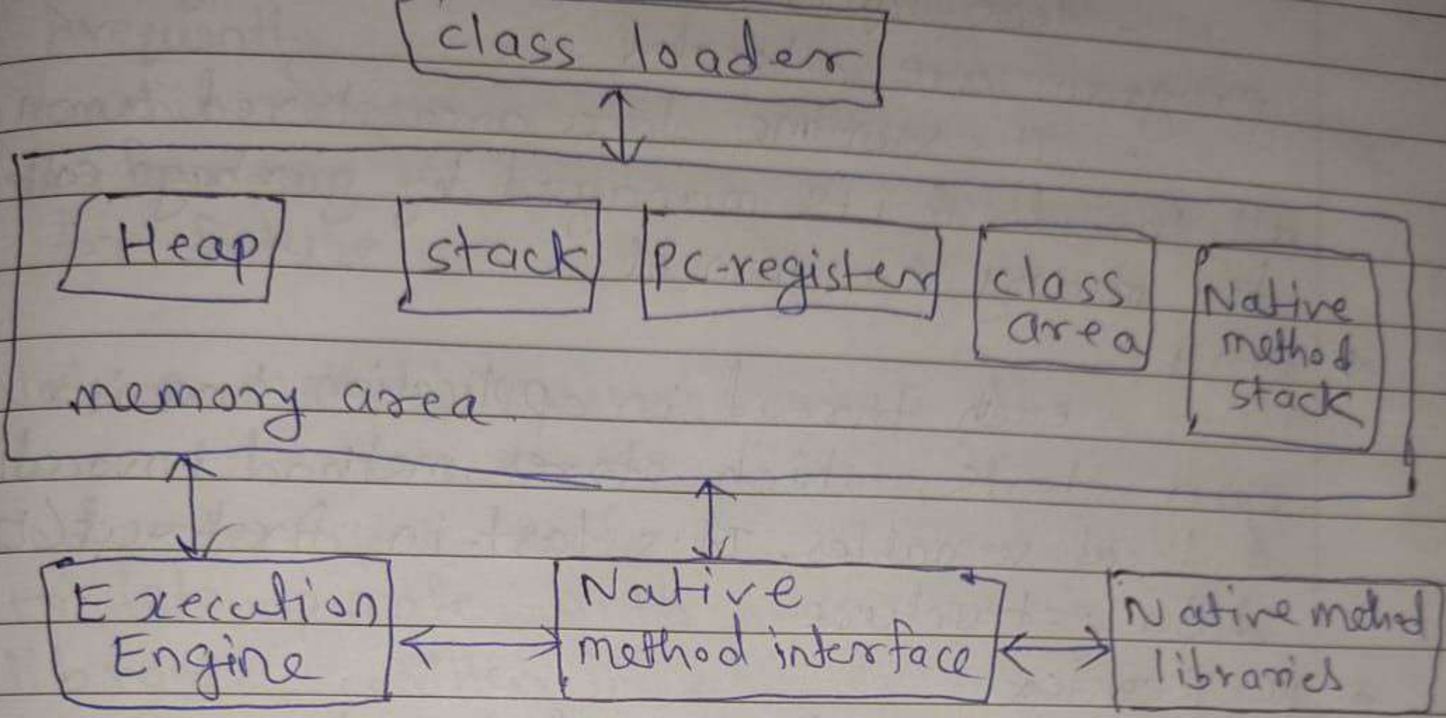
- Bytecode is the intermediate representation of java source code after it compiled by the java compiler.
- Bytecode is portable & platform independent allowing java program to run on any system with a JVM installed without requiring recompletion.

Importance of Bytecode is:-

- portability
- security
- performance
- flexibility.

6) Describe the architecture of the Jvm.

=>



class loader :-

loads class files into memory It is responsible for finding & loading .class files from file system, network & other sources & then converting them into binary system.

Runtime Data Area:-

This is area where data structures used by Jvm are allocated. It consists of several components.

- Method area :-

It stores class metadata, including method bytecode, ~~files~~ ~~fiel~~ method information, constant pool & static variables.

* -Heap-

- Here objects are created by java program are allocated.
- It's runtime data area stored among all threads & its managed by garbage collector for memory management.

* stack -

each thread in application has its own stack, which stores method invocations & local variables. It's last-in, first-out (LIFO) data structure.

* PC-Register -

each thread has its own program counter (PC) register, which holds address of currently executing instruction.

* Native method stack -

It is used for native method execution (methods written in languge other than Java).

* Execution Engine -

It executes java bytecode. it has 2 components.

- Interpreter -

It reads by bytecode & execute the code. producing native machine code instructions. its portable but relatively slow.

- Just-in-Time(JIT) compiler:-

JIT compiles frequently executed bytecode into native machine code at runtime, optimizing performance compiled code is then cached for future use.

Native method Interface -

It enables java code to runs inside a java virtual machine to interoperate with applications and libraries written in other programming languages.

Native method libraries:-

These contains native methods that provide functionality beyond JRE.

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

-> Java achieves platform independence through JVM. and the use of bytecode,

- Bytecode:- The java computer translates the source code into platform - independent bytecode bytecode is a set of instructions designed to be executed by the JVM.

-. JVM interpretation:- The JVM interprets bytecode instructions at runtime. instead of executing the bytecode directly on the underlying hardware the JVM interprets the bytecode instructions & execute them accordingly.

- standard libraries:- Java provides ~~standard~~ a comprehensive set of standard libraries that abstract away system - specific functionalities.

- class loading Mechanism :- Java's class loading mechanism allows classes to be loaded dynamically at runtime. this flexibility enables java application to load classes from various source such as the local file system, network custom resource, Q8. what is the significance of the class loader in Java?- what is the process of garbage collection in Java.

⇒ class loader:- loads class files into memory. It is responsible for finding and loading class files from file system, network & other sources & then converting them into binary form.

In garbage collection process, collector scans different parts of heap looking for objects that are no longer in use. If an object is no longer has references to it from else where in application. the collector removes objects, freeing up memory in the heap. This process continues till all unused objects are successfully reclaimed.

- To ensure garbage collector work efficiently, JVM separates heap into parts & then collectors use mark-and-sweep algorithm to traverse these parts & clear out unused objects.