ASSIGNMENT No-3

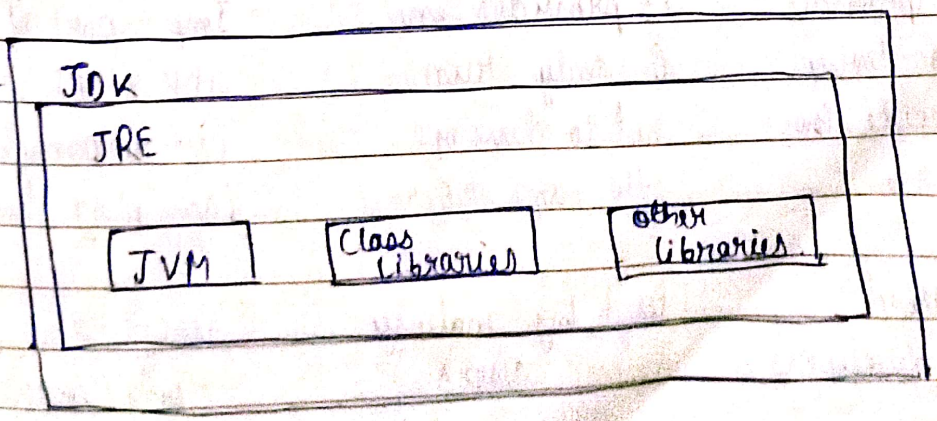
Q1] Explain the components of the JVM

Ans ⇒ There are 3 components of JVM.

1. Class Loader subsystem
   - a. Extension class loader
   - b. System class loader
   - c. Custom class loader
   - d. Bootstrap class loader
2. Runtime data areas
   - a. Method area
   - b. Heap
   - c. Java stack
   - d. PC Register
   - e. Nativ method stacks.
3. Execution engine
   - a. Interpreter
   - b. Just in Time compiler
   - c. Garbage Collector.

Q2] Explain the components of the JDK

Ans :-



JDK consists of :

1. Java Runtime env. (JRE)
2. An interpreter / loader (Java)

③ A Compiler (Javac)
④ An archiver (Jar) and many more.

The JRE in JDK contains a JVM and all the class libraries present in the production environment, as well as additional lib. useful for developers, eg:, IDL Libraries

Java is executable file inside /JDK/bin directory which is used to interpret / load / exute the .class files (executing Byte code files).

Javac is also one application present inside /JDK/bin directory which is used inside console for compiling the .Java (Source file) file.

The jar file is nothing but a pack of java classes.

Q3] Differentiate between JDK, JVM, & JRE.

Ans⇒ JDK	JRE	JVM
① Its a kit provides the env. to develope and execute the java program	It provides env. to only run not to develope the java program	Imp part of both JDK & JRE. it is responsible for java prog. line by line
② Developer used By mainly developers	used by mainly end users	used in process-ing / executing the prog files
③ JDK = JRE + other dev tools	JRE = JVM + other class Libraries	JVM = only Runtime envi. for exe. the java byte code
④ Platform dependent	platform dependent	platform independent

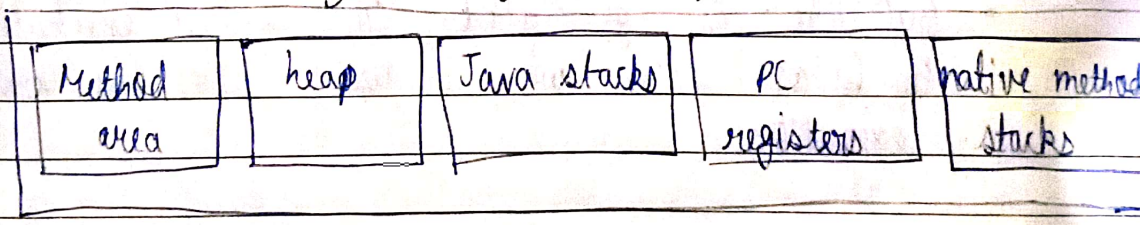
Q4] What is the role of JVM in Java ? & how does the JVM execute java code.

Ans] It's work is to loads code, it verifies the code then helps in the execution by making runtime execution environment.

- first class loader comes into the picture.
- So first (.class) file being imported into class loader and it would be storing all the (.class) file.
- After that it byte code inside (.class) file extracted from the (.class) file and then that byte code imported to the method area of the JVM memory.
- After JVM execution engine does its job.
- This code is then executed by execution engine. And inside that there are Byte code interpretation, JIT and optimizations and garbage collections.

Q5] Explain the memory management system of JVM.

Fig >



- heap => It is a shared runtime data area and stores the actual obj in a memory.
- Method area => The memory is allocated for class structures method data and constructor field data, and also for interfaces or special method used in class.

⊙ Stacks :- It is created at the same time when a thread is created.

⊙ Native method stacks :- Also known as C stack, native method stacks are not written in java language.

⊙ PC register :- Program counter registers is for non native methods stores the address of the available JVM instruction whereas for native method, the value of PC is undefined.

Q6] What is the JIT compiler and its role in the JVM.? What is the bytecode and why is it important for Java?

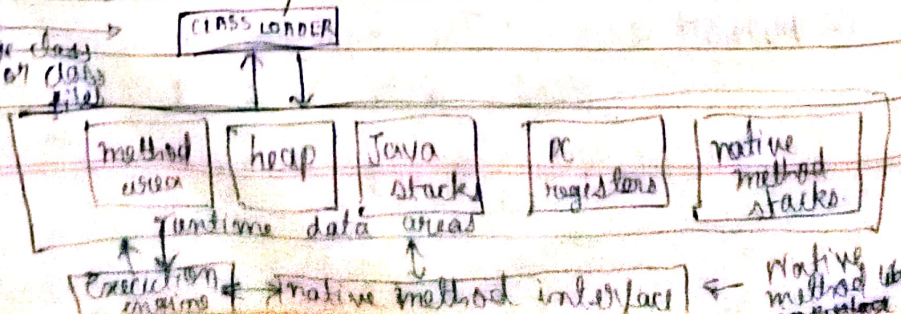
Ans ⇒ • JIT (Just in time compiler) with JVM interacts at the run time and compile suitable bytecode sequences into native machine code.

• Bytecode is generated code after executing the (.class) files which is used for the cross platform execution.

• JIT is important for performance optimization of Java based applications during run time.

Q7] Describe the architecture of JVM.

Ans ⇒



④ The class loader sub system mainly responsible for 3 activities.
① Loading → reads .class file generate binary data and save it in the method area
② Linking → performs verification, preparation & resolution
③ Initialization → all static variables are assigned with their values defined in the code and static block

⇒ Class loader → Bootstrap class loader, extension class loader, system & user defined class loader

⑤ JVM memory areas consist of :-
① heap area
② Method area
③ Stack area
④ PC registers
⑤ Native method stacks.

⑥ Execution engine :-
Executes the .class (bytecode). It read line by line uses data and info. present in various mem. area and executes instructions.

⑦ Native method interface → interface that interacts with native method libraries and provides the native libraries (C, C++) required for the execution.

⑧ Native method libraries → These are collections of Native Libraries required for executing native methods. They include libraries written in Lang. like C & C++.

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

Ans ⇒ As JVM helps to make runtime execution and helps to run / execute code, thus being a platform independent the byte code is then run by the jvm interpreter in other operating system which helps java to achieve platform independence through the JVM.

Q8] What is significance of the class loader in java? What is the process of garbage collection in Java?

Ans ⇒ It helps to load memory dynamically. The process of garbage collection in java is to allow programs to perform automatic garbage memory management. Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting unused obj.

Q9] What are the 4 access modifier in java, and how do they differ from each other.

Ans ⇒

public	protected	private	default
methods and data members are accessible from everywhere in the program	methods and data members declared as protected are accessible within the same package or the subclasses	data members methods declared are accessible only within the class in which they are declared	methods, dat members or classes are accessible only within the same package

Q11] Can you override a methode with different access modifier in a subclass? for example, can a protected method in a superclass be overridden with a private method in a subclass? Explain.

Ans → No, we can't override a methode in different access modifier in a subclass. As a super class is protected the overriding method is the subclass can be protected or public but not private.

Q12] What is the difference between protected and default (package-private) access.

Ans →	Default (package private) access	protected access
	① methods, data members or class are accessible only in the same package	① methods, data members which are protected are accessible only within the same package or subclass in different packages.

Q13] Is it possible to make a class private in Java? If yes, where can it be done, and what are the limitations?

Ans ⇒ Yes, we can make class as private but only only if is an inner or nested class.

Limitations :-

1. Can't declare top level classes or interfaces as private.
2. Private inner class can only be accessed within the outer class in which it is declared.
   - It means we can't make private inner class variable outside the class.

Q 14] Can a top level class in java be declared as protected or private ? why or why not.

Ans ⇒ NO, a top level class in java can't be declared as protected or private. Because a private top-level class would be inaccessible from any other class, making it effectively useless., and protected access is intended for inheritance and package level access, which doesn't apply to top-level classes.

Q 15] What happens if you declare a varible or method as private in a class and try to access it from another class within the same package?

Ans ⇒ It will throw an error because private members is only accessible within the same class in which it is declared.

Q16) Explain the concept of "package-private" or "default" access. How does it affects the visibility of class member?

Ans → The "package-private" or "default" is only accessible only within the same package. So the class fields, methods, and constructors that are not explicitly declared as public, protected, or private are accessible only within the same package. & Thus default access are not accessible outside their own package. This limits their visibility to other classes within the same package.