Assignment 3:

3.1 Explain components of the JDK.

→ - The JAVA Development Kit (JDK) is a software development kit (SDK) is used to develope Java application.
- It contains number of tools and components that are esschtial for java programming.

① Java compiler :- (javac)
- ~~This~~ used to compile java source code files int bytecode files

.Java → To → .class

- Bytecod is platform independent intermediate representaion of java programs that can be executed on any java virtual machine (JVM)

② Java Runtime Environment (JRE)
- It is Environment for executing java programs.
- It consist JVM & Java class library.
- JVM is executing bytecode instructions and interacting with OS
- Libraries are collection of interfaces & prewritten classes which support no various task such as input/output networking, graphics

③ Java debugger (.jdb)
- used to debug programs.
- allow step by step iteration of code by setting breakpoint, inspect variables.

and examine the call stack.

(4) Java Documentation Generator (Javadoc)
- generates HTML document from Java source code.
- extract comments from the code & creates a set of web pages that describe the classes, interfaces, methods, & fields

(5) Java Archives (JAR)
- it contains multiple classes and other files which support to execution
- JAR files used to package Java applications for deployment & distribution

(6) Java web start
- this not allows to launch Java application from web browser .. it downloads the necessary files including JRE & launches the application.

(7) Javafx
- set of tools & API for desktop apps creation

Q.2 Differences b/w JDK, JVM & JRE.

JDK	JVM	JRE
① complete package for java development include JRE & development tools used by developers	① Engine that runs java application converts byte code to machine code Ensures platform independence	① Environment to run java applications includes JVM and core libraries used by end-users to run Java programs.
② purpose - Development	② purpose - Execution	② purpose - Execution
③ components - Compiler, debugger, libraries.	③ components - Interpreter.	③ components JVM, libraries
④ used by:- Developers	④ used by: End-users	④ used by: End-users.
⑤ Developers use JDK to write, compile and debug Java applications	⑤ JVM only executes the compiled Java bytecode. It doesn't include tools for deployement	⑤ JRE is required to run java applications, not to develop them.
⑥ JDK = JRE + development tools	⑥ JVM is a part of JRE	⑥ JRE = JVM + libraries and files needed to run Java Programs.

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

→ Role JVM:-

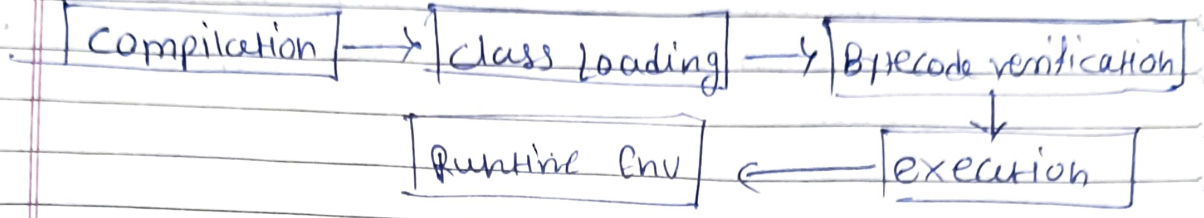
① platform Independence :-
the JVM allows Java applications to run on any device or operating system without modification.

② memory management
- JVM Handles memory allocation and garbage collection, ensuring efficient use of memory and preventing memory leaks

③ security:-
the JVM provides a secure execution environment by enforcing access controls and preventing unauthorized access to System Resources.

④ performance optimization:-
- JVM include JIT compiler (Just in time compiler). which optimizes the performance of Java applications by converting bytecode into native machine code at runtime.

JVM Execution



① Compilation
- compiled .java files into bytecode .class file. which platform independent & can be execute on any JVM.

② class loading :-
= after the compilation compiled bytecode are load into memory.
- This done by the class loader which loads classes as needed during the execution of the program.

③ Bytecode verification :-
- The JVM verifies the bytecode to ensure it adheres to Java security & integrity constraints.
- this prevents malicious code from running.

④ Execution :- The JVM interprets the bytecode or uses the JIT compiler to convert it into native machine code.
- the native code is then executed by the Host machines processor.

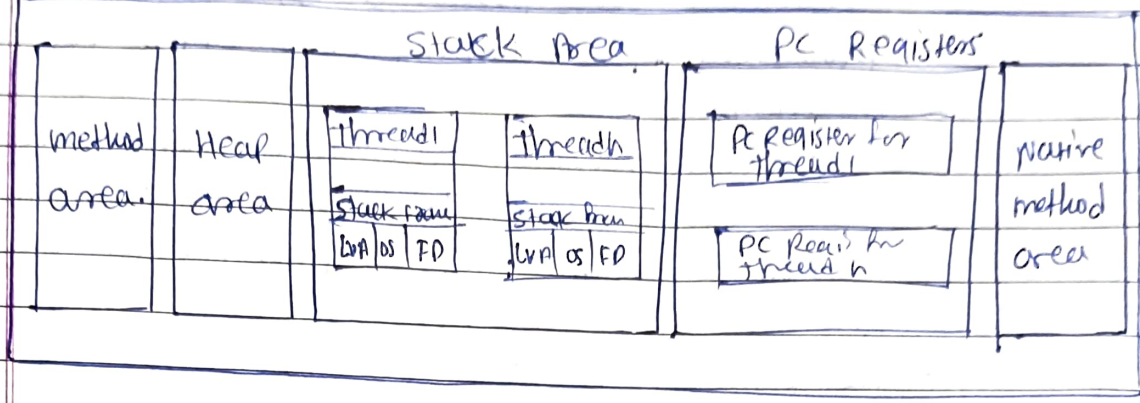
Date
Page

(5) Runtime Environment :-

- JVM provides the runtime environment including the necessary libraries and resources for the Java application to run.

Date
Page

(4) Explain the memory management system of the JVM



JVM memory structure

* Method Area

- the heap memory which is shared by all threads includes a section called method area, when the JVM starts it creats.
- the name of superclass, the name of the interface & the constructors are all stored there.
- It includes elements specific to each class, including fields constant pools, method local data, method codes, constructor codes etc, that are used to initialize objects and interfaces

- JVM throws an ~~out of~~ OutOfMemoryError if the memory in the method area cannot be made accessible to satisfy an allocation request.

Heap area

- objects are produced while a java program is running are kept in heap memory.
- the reference to newly formed objects is kept in stack memory. Dynamic memory allocation is followed by heap
- memory is allocated during the runtime or execution
- Heap memory is larger than stack.
- the garbage collector Automatically removes any unnecessary objects from the heap memory
- the heap memory does not have to be contiguous.
- JVM gives the users the option to initialize or change the heap size as needed.
- JVM throws OutOfMemoryError whenever a calculation needs more heap than the default.
- object is get placed into the heap when new keyword is used.

- Heap is divided into parts:-
  - ~~Young~~ young Generation
    - All new objects are assigned and aged in this location. when this is full there is a small garbage collection
  - old generation.
    - long lasting items are kept in this location. A threshold for the object's age is specified when it is placed in the young Generation and once it is met, the object is transformed to the the old Generation.
  - Perment generation.
    - this comprises JVM metadata for the application methods and runtime classes.

* Stack Area:
- when a thread starts, the stack area is generated. It might have a fixed size or a variable size.
- Each thread has its allocation of stack memory. Data & unfinished results are stored there.
- there are heap object references in it.
- Furthermore, it stores the value directly rather than a reference to an item from the heap.
- the term "scope" refers to the visiability of the variables that are kept in the stack.

- It is not necessary for stack memory to be continous.

* Native method stack:
- Native method stacks often known as c stacks are not created in the java programming language.
- each thread recieves a memory allotment when it is formed.

* program Counter (PC) Registers:

- A program Counter (pc) Register is linked to every thread. A native pointer or the return address is kept in the pc register. It also includes the location of the JVM instruction that are now being carried out.

* JIT compiler & it's Role in the JVM.? what is the bytecode and why is it important for java

- JIT (Just-in-time) Compiler

- the JIT compiler is a part of the java Runtime Environment (JRE) that enhances the performance of java applications by converting bytecode into native machine code at runtime.

Role in the JVM.

- performance optimization:
  - By compiling bytecode into native code, the JIT compiler allows the JVM to execute code faster than interpreting it repeatedly

- Dynamic compilation.
  - the JIT compiler compiles code on the fly, meaning it compiles methods as they are called optimizing frequently used methods for better performance

- Reduced overhead :-
  - Once a method is compiled the JVM can directly execute the native code, Reducing the overhead of interpreting bytecode.

* Bytecode & its importance in Java

- Bytecode is an intermediate, platform-independ... Code generated by the Java compiler from Java source code
- It is stored in .class files and executed by the JVM.

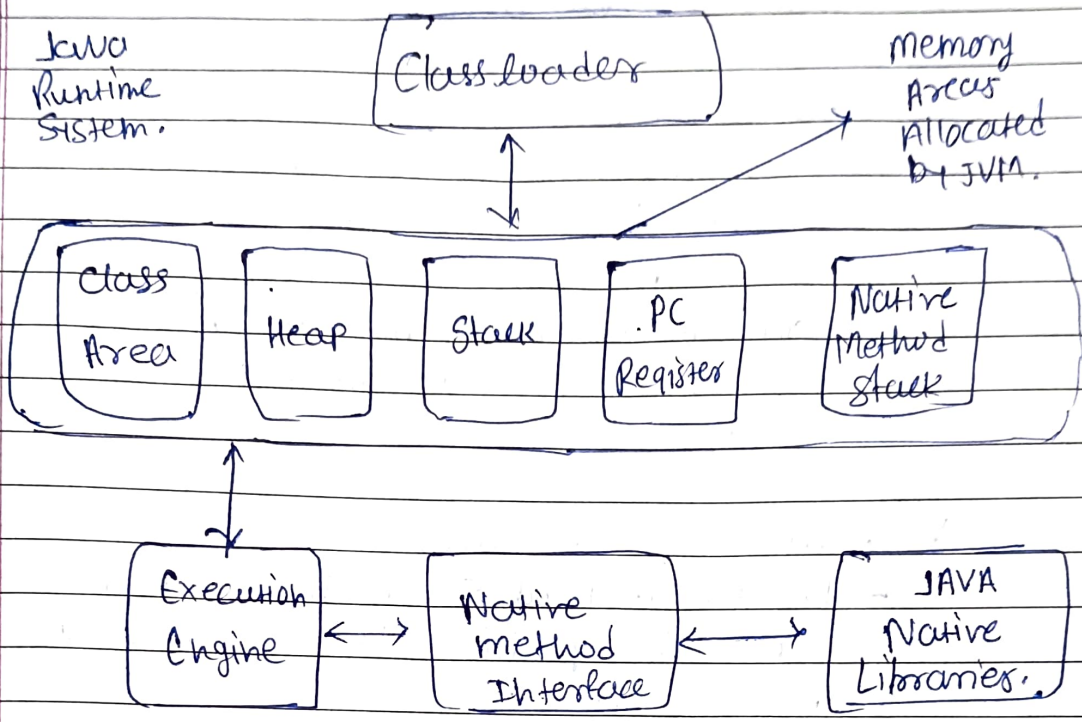
* Importance

- Platform Independence -
Byte allows Java programs to be "write once, run anywhere". The same bytecode can run on any device or operating system that has a JVM.

- Security:-
Bytecode is verified by the JVM before execution, ensuring that it adheres to Java's security

- Efficiency:-
Bytecode designed to be compact and efficient, making it easier for the JVM to load and execute.

Q6 JVM Architecture.



Class loader:-
- loading Java class fiels into JVM at runtime
- verifying bytecode & defining the classes in JVM.

Runtime Data Area:
- Runtime Data Area is the Memory area where the JVM manages data during program execution. It consist of seven Components.

Method Area.
- Store class level data including bytecode of methods, constant pool, static variables, & method metadata

* Heap:-
- Heap is the runtime data area where objects are allowed.

* 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 & returns.

* Native Method Stack:
- 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:-
- 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.

Q7 How does Java achieve platform independence through the JVM.

* Compilation to Bytecode:-
  - Java compiler (javac) compiles the sources code into an intermediate form called bytecode.
  - This bytecode is stored in .class files ~~that~~ and is platform independent.

* JVM Execution:-
  - The JVM is a platform-specific implementation that reads & executes the bytecode. Each operating system has its own JVM, which translates the bytecode into machine code that the specific platform can understand.

* Write once, Run Anywhere:
  - bytecode is the same regardless of the underlying hardware or operating system. You can write your Java program once and run it anywhere that has a compatible JVM.

* Portability:-
  - This setup ensures that Java programs are highly portable.
  - The JVM abstracts the underlying hardware and operating system details, allowing the same bytecode to run on different platforms without modification.

Q. 8 X what is the significance of the class loader in Java ? what is the process of garbage collection in Java?

→ - The class loader is responsible for locating library, reading their contents and loading the classes contained in the libraries.

- Garbage collection in java is the automated process of deleting code that's no longer needed or used this automatically frees up memory space and ideally makes coding java apps easier for developers.

Q. 9. what are access modifies in Java? How do they differ from each other?

- the four access modifies in Java, are public, private, protected & default.

- public :- can be accessed from any class regardless of package.

- private :- Are only accessible within the class where they are declared.

protected : can be accessed within same package and by subclasses

in other packages.

- Default :- If no access modifier is specified the member is considered as package private access within same packages.

Q.10) What is the difference between public protected & default access?

→ (1) public :-
- can be accessed by any class regardless of package
- considered at most open access level

(2) Protected :-
- can be accessed within the class itself and its subclasses, even if they are in a different package.
- used to control access when inheritance is solved.

(3) Default :- (package level private)
- can be accessed by classes within the same package
- considered the default access level if no explicit access modifiers is specified.

(11) Can you override a method with a diff acess modifier in a subclass? Can a protected method in superclass be overridden with put method subclass Explain.

→ yes. The protected method of a Superclass can be overridden by a subclass, If the superclass method is protected the subclass overridden method can have protected or public which means subclass over. ridden method can not have a weaker access specifier.

Q.12 what is the diff bet^n protected & default access?

→ 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 the 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

Q.13 Is it possible to make a class private in Java? If yes where it can be done & what are the limitations?

→ Yes private classes are allowed, but only as inner or nested classes. If you have a private / nested class, then the access is restricted to the scope of that outer class. If you have a private class on own as a top level class then you can't get across to it from anywhere.

Q.14 Can a top-levels class in Java be declared as protected or private? why or why not?

→ - No. we cannot declare a top level class as private or protected. It can be either public or default (no modifier). If it does not have a modifier it is supposed to have a default class.

- Because the whole point of private is to disable the usage of the item outside the class they have been defined in.

Q'15 What happen if you declare a variable or method as private in class and try to access it from another class within the same package.

→ The methods or data members declared as private are accessible only within the class from the same package will not be able to access these members private means "only visible within the enclosing class".

Q.16) Explain the concept of package private or default access. How does it affect the visibility of class members?

→ "private" restricts access to the elements only within the class they are declared.

- protected - allows access within the same package or in subclasses which might be in different packages.

- lastly, the "default" access (no modifier) limits the visibility to classes within the same package.