Page No.
Date

Assignment No. 3

Q.1 Explain the components of the JDK.

⇒ - Java Development Kit (JDK) is a Software development environment used for developing Java applications and applets.

- Components are Java runtime environment, compilers, debuggers that can be used to develop, test and deploy java applications
- Java, Javac, jar, javadoc, jdb, jstack, jmap are basic components of the JDK

i) Java ⇒ command line too used to execute java program and application

ii) javac ⇒ Compile java Source code file into bytecode

iii) jar ⇒ used to package java classes & resources into JAR files.

iv) javadoc ⇒ Generate API documentation from java Source code

v) jdb ⇒ debug Java programs using Java debugger.

vi) jstack ⇒ To print the stack trace of a java process or thread.

Q.2 Differentiate Between JDK, JVM and JRE

⇒ JDK ⇒ i) JDK Stands (Java Development Kit) it is used for developing applets and java applications.

ii) It contain JRE + development tools

JVM ⇒ i) JVM Stands (Java virtual Machine), It provide a runtime environment for driving Java applications or code

ii) JVM is an abstract machine that converts the java bytecode into a machine language.

ii) JVM is essentially a part of the JRE, you cannot separately download and install it.

3) JRE ⇒ JRE stands (Java runtime Environment), It is a set of software tools designed for running other software
- It is an installation package that provides an environment to only run the java program on your machine.

Q 3 What is the role of JVM in Java? & How does the JVM execute Java code?

⇒ - Java code is written in .java files and compiled into bytecode, which is stored in .class files, This bytecode is platform-independent meaning it can be executed on any device that has a JVM.
- It provide a runtime environment for java applicati to run on different platform and operating system.

• Execution
- Java code firstly compiled by java compiler into byte code which stored in .class files.
- JVM class loader load .class in memory, link them
- bytecode verifier checks valid or not
- Then jvm executes bytecode using interpreter or Just-in-time (JIT) Compiler.

Q 4 Explain the memory management System of the JVM.

⇒ - JVM creates various run time data areas in a heap These areas are used during the program execution
- The memory areas are destroyed when JVM exits, data areas destroyed when thread exits.

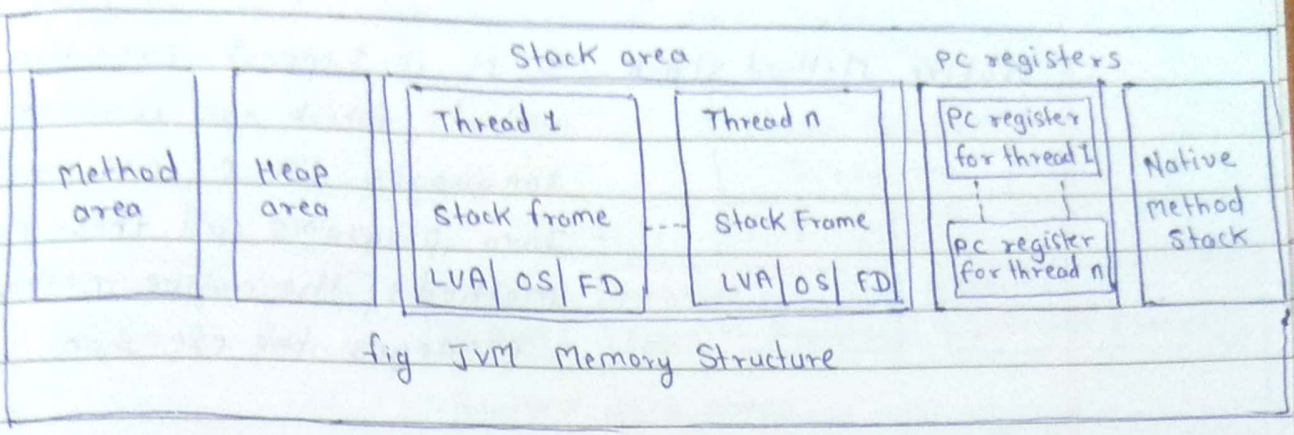


fig JVM Memory Structure

- Method area ⇒ It is part of heap memory which is shared all the threads. It creates when the JVM starts up. It is used to store class structure, Superclass name, interface name & constructor

- Heap area ⇒ - Heap stores the actual object. It creates when the JVM starts up. It can be of fixed size or dynamic size
  - when we use new keyword, the JVM creates an instance for the object in a heap. and the reference of that object stores in the stack

- Stack area ⇒ - Stack area generates when a thread creates. It can be of either fixed or dynamic size.
  - It contains references to heap objects
  - Stack Frame is data structure that contains the thread data, Thread data represents the state of the thread in the current method.

- PC registers ⇒ PC stands (Program Counter), It is small memory area that stores the address or instructions of the bytecode in a thread.

Native Method stack ⇒ It is support execution of native methods, which are written in languages like C, C++ instead of java
- Java programs call these native methods, the native method stack manages the execution.

Q.5 What are the JIT compiler and its role in the JVM? What is the bytecode & why is it important for Java?

⇒ - JIT stands the (Just in time) it is component of java runtime environment
- It improve performance of Java applications, and compiles its using bytecode to machine code at runtime.

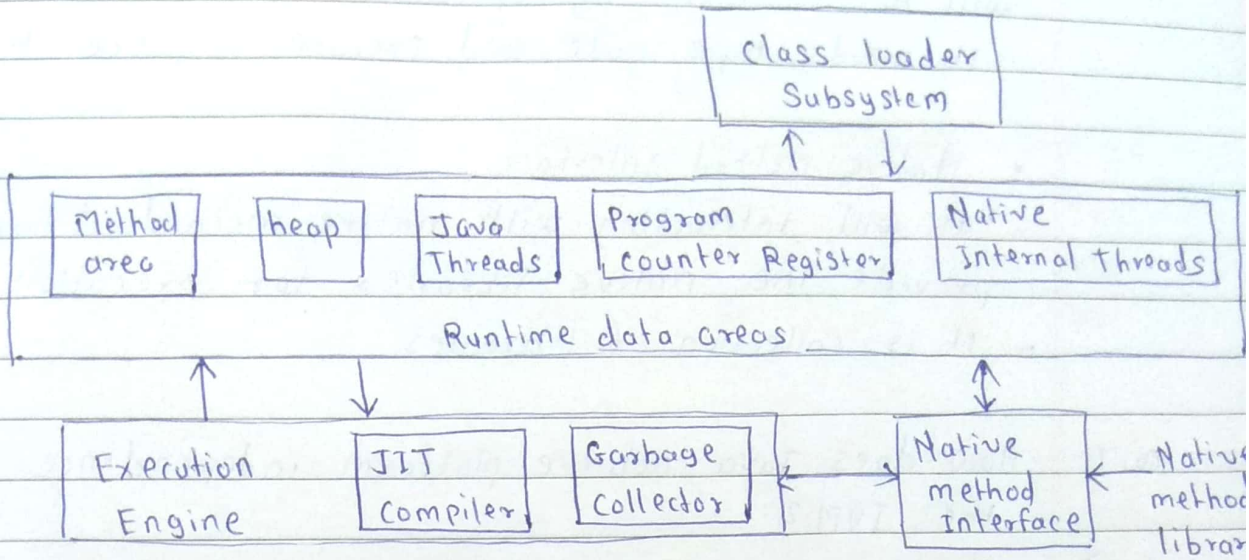
• role
- It is Speed up Java programs.
- It convert program from bytecode into machine code, So computer directly understand it.

• Bytecode
- Java Bytecode is instruction set of Java virtual machine. If we compiled java program, java bytecode is generated.
- bytecode in forms of .class file.

• Importants
- It helps java to achieve Security, efficiency and portability.

Q.6 Describe the architecture of the JVM.

⇒



- Class loader Subsystem
  - It verifies class files using a bytecode, bytecode in forms of .class file
  - If we compiled Program, then bytecode file is generated.

- Run time data areas
  - run time data areas contain Method area, heap, Java Threads, PC, Native Internal Threads
  - Method area stores data like as fields, constant pools & method information.
  - In Heap all objects Stored in JVM, heap contains arrays because arrays are objects.
  - Java thread is lightweight process that allows to execute multiple task concurrently (at the same time).
  - PC is Small register in the CPU, it execute the next instructions in a program.
  - Native Internal Support, execution of native methods.

- Execution Engine
  - The bytecode which is assigned to the runtime data area

will be executed by Execution Engine
- It reads byte code and execute it piece by piece

- Native Method Interface
- It will interacting with native method libraries & provide the native libraries for execution engine
- It is collection of libraries.

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

⇒ Java can run on any device without modification it uses write once, run anywhere approach. It source code is compiled into bytecode, this bytecode can be executed any other platform thats why java is platform independence through JVM.

Q.8 What is the Significance of the class loader in Java? What is the process of garbage collection in Java?

⇒ class loader is abstract class in Java.lang package. It loads the classes at run time, It helps to manage security by ensuring classes come from trusted locations
- class loader load classes in different sources, like files or network.

- Garbage Collection.
- It is automatically manage memory it remove objects / variables in a program that are no longer used.
- free up the memory Space, So that the prevent memory leaks.

Q.9 what are the four access modifiers in Java?

* Java Modifiers
  - Public
  - Private
  - Protected
  - defaut

Access Modifiers

1) • Public
   - It is used to set the access level for classes, attributes, methods and constructors
   - The code is accessible for all classes
   - Prg

```
public class Main
 { public String fname = "Dipti"
   public String lname = "Patil"
   public int age = 23;
 }
class First
 {  public static void main (String []args)
     {
        Main obj = new Main();
        System.out.println ("Fname :"+obj.fname);
        System.out.println ("Lname :" +obj.lname);
        System.out.println ("Age :" +obj.age);
     }
 }
```

2) • Private
   - Private class declared with the private access modifier
   - Only nested (inner) classes can be declared as private
   - A private nested class is a class defined within another class and its access is restricted to the outer class.

Prg

```
public class Main
{
    private String fname = "Dipti";
    private String lname = "Patil";

    public Static void main (String [] args)
    {
        Main obj = new Main ();
        System.out.println ("First Name:" + obj.fname);
        System.out.println ("Last Name:" + obj.lname);
    }
}
```

3) Protected

- Protected access modifier in Java is used to control the visibility of class members (fields, method, constructor)
- Members declared with protected are accessible within the same package., means that any class in same package can access protected members

6) default (Package level Private)

- The class is only accessible by classes in the same package.
- Means that any class within the same package can access members that are declared with default access
- The defaut access modifier is also called package - private

Q 10) what is different between public, protected and defaut access modifier

⇒

Access modifier	Same class	Same Package	Subclass (Different Package)	Outside Package
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No

public ⇒ The member is accessible everywhere
protected ⇒ Accessible within Same class, Same package and Subclass (different Package)
Default ⇒ Accessible only within the Same class and Same package.

Q.11 Can you override a method with a different access modifiers in a Subclass? For example, can a protecte method in a Superclass be overridden with a private method in a Subclass? Explain.

⇒ - Yes, we can override by using ~~protected~~ different access modifiers.
- No, a protected method in a Superclass cannot be overridden with a private method in a Subclass, the access level of overriden method in the Subclas must be Same or less restrictive than the method in the Superclass

Q.12 what is the difference between protected and default (package-private) access?

⇒ • Protected ⇒ It is only accessible within Same class, Same package and Subclass (different Package)

- Default ⇒ It is accessible within same class and same package.

Q.13 It is possible to make a class private in Java? If yes, where can it be done and what are the limitations?

→ - Yes, it is possible, only as inner class or nested classes
- If you have a private class on its own as a top-level class, then you can't get access to it from anywhere

Q.14 Can a top-level 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.
- because private class, member or method is enclosed by the private class also protected class it is only accessible through the owner class and its subclasses. So you can apply private or protected modifiers only on the nested classes.

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

⇒ - Compilation error is occured.
- because if you declare a variable or method as private in a class, it is only accessible within the class where it is defined.

Q.16 Explain the concept of "package-private" or "default" access, how does it a affect the visibility of class members?

⟹
- Package-Private (default) means the member is accessible only within its own package
- If we do not specify any access modifier, java automatically treats the class, field or method as package-private.
- visibility of class is accessible within the same class, or within the same package