Assignment 3.

(1) Explain the components of JDK?

→ The Java development kit (JDK) is a comprehensive package that provide everything needed to develop Java application.

Java Compiler :- converts Java source code into bytecode which the JVM can execute.

Java Runtime Environment (JRE) :- provides the libraries Java virtual machine (JVM).

Java Debugger :- a tool for debugging Java programs.

Java Documentation :- a tool that generate API documentation from Java source code
- Comments.

Java Archie (Jar) :- A utility to bundle multiple files into a single Jar file for easier distribution.

Java libraries (rt.jar) :- A set of pre compiled class libraries that provide standard functionalities for Java progress.

(2) ~~2~~ Differentiation

PAGE No.
DATE / /

(2) Differentiation bet^n JDK, JVM & JRE

→ JDK (Java Developed kit) :-
A full Developement environment that includes the JRE tool for developing Java application. (compiler, debugger etc) & other utilities.

JVM (Java Virtual Machine) :-
A virtual machine that Java bytecode is executed.

JRE (Java Runtime Environment) :-
A subset of the JDK that include. The JVM, core libraries & other components needed to run java applications.

(3) Role of the JVM in Java and How it executes Java code.

→ The JVM plays a critical role in running Java app^. It abstract the underlying Hardware & operating system details providing a platform independant environment.

(i) Compilation :- Java Source code is compiled into bytecode by the Javac compiler.

(ii) Class loading :- The JVM loads the bytecode using the class loader.

(iii) Bytecode Verification :- The bytecode is verified for security & correctness.

(iv) Execution :- The JVM interpret the bytecode or uses the just in time.

(4) memory management system of JVM?

→ The JVM manage memory in Rwl region.

Heap :- stores object & their associated data.
It is divided into
young gen :- where new objects are allocated & aged.
old gen :- where long surviving objects are stored.

stack :- Each threads in Java has its own stack storing method call & local variable.

Method Area :- stores class structures, method data & constant pool.

Program Counter :- keeps track of the JVM instruption currently being executed.

Native method stack :- used for native method calls via

(5) JIT Compiler, Bytecode & their importance.

→ JIT Compiler (Just in time) is a part of the JVM that optimize bytecode execution by compiling it into native machine code by runtime, improving performance.

Byte Code :- A platform independent, intermediate representation of Java code that is executed by the JVM. Bytecode is crucial because it enable Java's "write once, run anywhere" capability.

(6) Architecture of JVM?

→ Class loader subsytem :- loads, links & initialize classes during runtime.

Runtime Data Areas :- includes the method area, heap, stack, program counter and native method stack.

Execution engine :- Consist of Interpreter, Execute bytecode ~~instor~~ instructions.

JIT Compiler :- Compile bytecode to native for performance optimization.

Garbage Collector :- Automatically manages by reclaiming memory used by objects no longer in use.

Native Method Interface :- Allow Java to interact with native appn written in other language like C or C++.

(7) Java's platform independance through the JVM.

→ Java achieve platform independance through the JVM by compiling Java source code into platform independent byte code. The JVM on each platform interpret this byte code and execute on the specific hardware ensuring that the same Java program can run on any platform without modification.

(8) significance of the class loader & Garbage collection in Java?

→ Class loader :- Responsible for dynamically loading Java classes into the JVM at runtime. It separates the loading of classes into three phases :- loading, linking and initialization. the classloader also helps with namespace management & ensure classes are not loaded more than once.

Garbage Collection :- Java use garbage collection to automatically manage memory. The JVM identifies objects that are no longer referenced by any part of the program & reclaim their memory preventing memory leaks & improving efficiency.

(9) four Access Modifiers in Java?

→ public :- The member or class is accessible from any other class.

protected :- The member ~~or class~~ is accessible within it own package and subclasses.

Default :- The member is accessible only within its own package. It is the default level if no modifiers is specified.

private :- The member is accessible only within the class it is declared in.

(10) Diff^n bet^n ~~from~~ ~~on~~ public, protected & default Access modifiers?

→ public :- Accessible from any other class regardless of the package.

procted :- Accessible within the same package and by subclasses, even if they are in diff^n package.

Default :- Accessible only within the same package not visible to classes in other package.

(11) overriding a method with different Access Modifier?

→ you cannot override a method in a ~~class~~ subclass with a more restrictive access modifiers. for ex. protected method in superclass cannot be overriden with a private method in a subclass. However you can override it with a method having protected or public access.

(12) Diff^n bet^n protected & default access?

→ protected :- Allow access within the same package and also to subclasses, even if they are in diff^n package.

Default :- Restrict, access to classes within the same package, subclasses in diff. packages cannot access members with default access.

(13) Making a class private in Java?

→ You cannot declare a top-level class as private. However you can declare inner classes (classes within class) as private. A private inner class is accessible only within it outer class.

(14) Declaring a Top-level class as protected or private?

→ A top level class in Java cannot be declared as protected or private. Top level classes can only be public or have default access. This is because protected and private access would restrict the visibility of the class too severly, conplicting with Java's package-based access control.

(15) Accessing private variable or methods form another class in the some package.

→ If a variable or method is declared as private. it cannot be accessed from another class even if that class is within the same package. private members are only accessible within the class they are declared in.

(16) Concept of "package-private" or "Default" Access.

→ "package-private" or "Default" access in Java means that the member is accessible only within its own package. If access modifier is specified, the member is treated as having default access. This restricts visibility to other classes in the same package and prevents access from classes in other package.