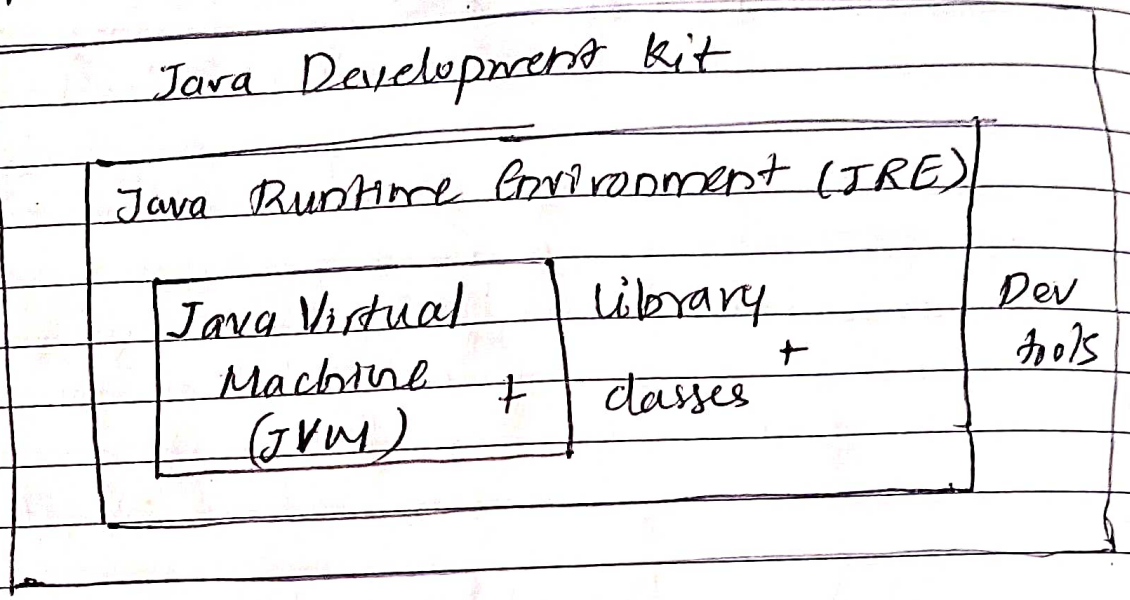
① Explain the components of the JDK.



- The java compiler translates java source code into bytecode that can run on any platform with a Java Virtual Machine (JVM).

- The java Virtual machine (JVM) provides an execution environment for Java bytecode

- The Java Class library is a collection of pre written code developers can use to build java appl^n

- The java debugger is a tool developers can use to find & fix errors in their code.

NOTES - JavaDoc provides an easy way to generate documentation from Java source code.

② Difference b/w JDK, JVM & JRE.

JVM:

08.00 – It is called a virtual machine because it doesn't physically exist. It is a specification that
09.00 provides a runtime environment in which Java bytecode can be executed.

10.00 task performed:
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

13.00 JRE: Java run-time env. is a set of software tools which are used for developing Java appl^n.
14.00 - It is used to provide the runtime env..
- It is the implementation of JVM.
15.00 - It physically exists & contains a set of libraries + other files that JVM uses at
16.00 runtime. SUNDAY 18
↓ rt.jar

17.00

JDK: Java development Kit is a software
18.00 development environment which is used to develop Java appl^n & applets. It physically exists. It contains JRE + development tools.

JDK is an implementation of any of the below given Java platforms

JULY '11 Su Mo Tu We Th Fr Sa 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
AUGUST '11 Su Mo Tu We Th Fr Sa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

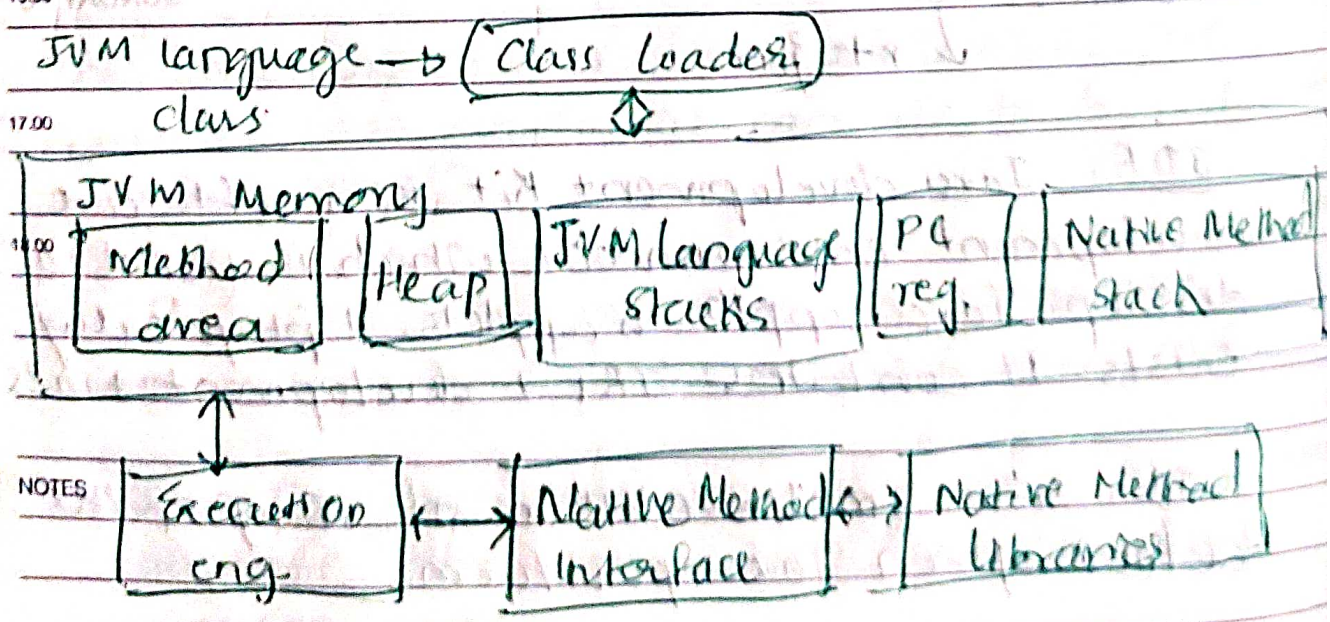
- Standard Edition Java Platform
- Enterprise Edition Java platform
- Micro Edition Java platform

The JDK contains a private JVM & a few other resources such as an interpreter (java), a compiler (javac) an archiver (jar) a documentation generator (Javadoc) etc. to complete the development of a Java Appln.

(3) Role of JVM & how JVM exe. java code?

JVM runs Java appln as a run time engine. JVM is the one that calls the main method present in a Java code.

The .class files (containing byte code) goes into various steps when we run it. These steps together describe the whole JVM.



NOTES

Execution engine executes the class (bytecode). It reads the byte code line by line, use data & info. present in various memory area & exe. Instr. It can be classified as:

- Interpreter: It interprets the bytecode line by line and then executes. The disadvantage here is that when one method is called multiple times, every time interpretation is req.

- Just in time compiler (JIT): It is used to increase the efficiency of an interpreter. It compiles the entire bytecode and changes it to native code so whenever the interpreter sees repeated method calls, JIT provides direct native code for that part so re-interpretation is not req. thus efficiency is improved.

- Garbage Collector: It destroys un-referenced objects.

NOTES

JULY '11
Su Mo Tu We Th Fi Sa
31 1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

AUGUST '11
Su Mo Tu We Th Fi Sa
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

(4) mem. mgm. in JVM.

→ Method area: In the method area, all class level information like class name, immediate parent class name, methods & variables info. etc. are stored including static variables. There is only one method area per JVM, & it is a shared resource.

Heap area: Information of all objects is stored in the heap area. There is also one Heap area per JVM. It is also a shared resource.

Stack area: For every thread, JVM creates one run-time stack which is stored here. Every block of this stack is called activation record / stack frame which stores methods calls. All local variables of that method are stored in their corresponding frame. It is not a shared resource.

PC reg.: Store address of current exe. instr. of a thread, each thread has seperate PC register.

Native method stacks: For every thread, a seperate native stack is created. It stores native method information.

NOTES

(5) What are the JIT compiler & its role in the JVM? What is the bytecode & why is it important for Java?

— The JIT compiler in the JVM improves Java appl^n performance by converting frequently exe. bytecode into machine code at runtime, allowing faster execution. It works alongside the JVM interpreter, compiling hot spots of code and optimizing them for efficiency.

Bytecode is platform-independent intermediate code generated by the Java compiler. It is executed by the JVM allowing Java to maintain its "WORA" promise.

Bytecode is crucial because it enables platform independence, enhances security through JVM checks, and allows runtime optimization by the JIT compiler for better performance on different platforms.

23 September FRIDAY 266-099 39 Week '11

JULY '11 | AUGUST '11

(9) What are the four access modifiers in Java, and how do they differ from each other?

(10)

(12)

Access modifiers help to restrict the scope of a class, constructor, variable, method or data member. It provides security, accessibility etc to the user depending upon the access modifier used with the element.

(1) Default / package level private - No keyword required
- accessible only within the same package.

(2) Private Keyword: private
- accessible only within the class in which they are declared.
Top lvl classes or interfaces can not be declared as private.

(3) Protected: accessible within the same pkg. or subclasses in different packages

(4) Public: accessible from everywhere

NOTES

	Same package			Different package	
	Same class	Sub class	Non sub class	Sub class	Non sub class
private	A	NA	NA	NA	NA
default	A	A	A	NA	NA
protected	A	A	A	A	NA
public	A	A	A	A	A

08.00

10.00

11.00

(11) Can you override a method with a different access modifier in a subclass? For example, can a protected method in a ~~sp~~ superclass be overridden with a private method in a subclass? Explain.

Yes,

- Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by its super-class or parent class.

14.00

15.00

When a method in a subclass has the same name, same parameters or signature & same return type as a method in its super class, then the method in the subclass is said to override the method in the super class.

16.00

SUNDAY 25

17.00

No, a protected method in a super class can not be overridden with a private method in a subclass as private is more restricted than protected.

NOTES

JULY '11
Su Mo Tu We Th Fi Sa
31 1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

AUGUST '11
Su Mo Tu We Th Fi Sa
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

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

— Inner classes (nested classes) can use private access modifiers. It allows for encapsulation i.e hiding implementation details. Limitation: accessible only within the same class.

14) Can a top-level class in Java be declared as protected or private? Why or why not?

— No, a top-level class in Java cannot be declared 'protected' or 'private'. Top-lvl classes can only be 'public' or 'package lvl private' (default). This is because Java requires top-level classes to be accessible at the package or global level for proper compilation & runtime mgm. 'protected' & 'private' access modifiers are meant for controlling access to class members (fields, methods) rather than the classes themselves.

NOTES

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 pkg?

- If you declare a variable or method as 'private' in a class and try to access it from another class within the same pkg., you'll encounter a compilation error. The 'private' modifier restricts access to the declaring class only regardless of pkg. This means other classes even within the same pkg. cannot access private members.

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

- It grants visibility to all classes within the same package (sub class, same class, non sub class) but restricts access from classes outside the package:
(Sub class, Non-sub class)

NOTES

JULY '11
Su Mo Tu We Th Fr Sa
31 1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

AUGUST '11
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

(8) What is the significance of the class loader in Java? What is the process of garbage collection in Java?

Garbage collection in java is an automatic process that manages memory by reclaiming memory used by objects no longer referenced by the program. The JVM's garbage collector identifies these objects (unreferenced) & frees up memory ensuring efficient memory use without manual intervention.

Steps:

(1) Marking: The GC identifies which objects are still reachable (in use) and marks them alive.

~~(2) Sweeping: It identifies unreferenced obj. the~~

(2) Sweeping: Now we have the memory space which is occupied by live & dead obj. The sweep phase releases the memory fragments which contains these dead objects.

(3) Compacting: After reclaiming memory from unreferenced obj. the remaining objects are compacted to reduce ~~fg~~ fragmentation.

GC process runs in the background.

Significance of the Class Loader in Java:

- The class loader is a crucial part of the JVM responsible for dynamically loading Java classes at runtime. Its significance is:

1. Dynamic Class loading: Classes are loaded only when req., saving memory & improving efficiency

2. Namespace Mgm: It ensures that each class is loaded within its own namespace, avoiding conflicts.

3. Security: The class loader ensures that classes are loaded from trusted sources and can enforce access restriction.

4. Extensibility: Developers can define custom class loaders for special class-loading mechanism (eg: loading classes from non-standard sources like databases or network locations).

NOTES

JULY '11
Su Mo Tu We Th Fr Sa
31 1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

AUGUST '11
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

(7) How does Java achieve platform independence through the JVM?

- When a java program is compiled, the java compiler converts the source code into bytecode, which is platform independent.

- The bytecode is not specific to any m/c or OS. Instead it is executed by the JVM, which is available for various platforms (windows, linux, macOS, etc.). Each platform has its own implementation of the JVM, which translates the bytecode into m/c code for that specific system.

- Since the JVM abstracts away the underlying h/w & OS differences, a java program can run on any m/c that has a compatible JVM installed, achieving the WORA. "write once ~~read~~ run anywhere!"

NOTES

(6) Describe the architecture of the JVM?

08.00 - The architectural components of JavaVM:

09.00 (1) Class loader: Responsible for loading Java classes (bytecode) into the JVM. It loads classes as needed and handles tasks like loading, linking and initializing classes dynamically during runtime.

10.00

11.00

(2) Memory Areas:

- Method area: Stores class structures, including methods, fields and constant pool data.

13.00

14.00

- Heap: Used for dynamic memory allocation for objects

15.00

- Stack: Stores method call frames, including local variables and method return values.

16.00

17.00

- Program Counter (PC) Register: Keeps track of the address of the next instruction to be executed

18.00

- Native Method stack: Holds native method calls (non-Java code).

NOTES

AUGUST '11
Su Mo Tu We Th Fi Sa
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

SEPTEMBER '11
Su Mo Tu We Th Fi Sa
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30

(3) Execution Engine: Responsible for executing bytecode. It includes:

08.00

- Interpreter: Reads & executes bytecode
09.00 instructions line by line.

10.00 - Just-in-Time (JIT) Compiler: Compiles frequently executed bytecode into
11.00 native machine code for better performance

12.00 - Garbage collector: Manages automatic memory mgm. by reclaiming memory used by
13.00 objects that are no longer in use.

14.00 (4) Native Method Interface (JNI): Allows the JVM to interface with native code
15.00 written in languages like C or C++.

16.00 (5) Native Libraries: External libraries that the JVM uses to interact with the
17.00 underlying operating system & h/w.

18.00 The JVM architecture ensures platform independence, memory mgm. & runtime efficiency for Java program.

NOTES