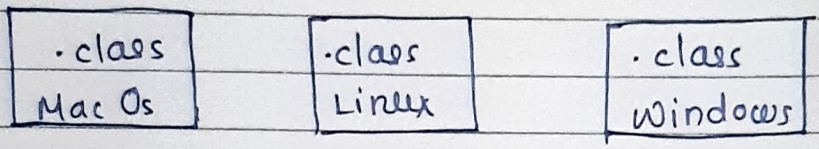
Assignment -3

Q.1) Components of the JDK.

→ - JDK stands for Java Development kit.
- Used for developing Java Programming Applications.
- It contains tools such as, Java compiler which converts Java source code to the byte code.
- Basically the bytecode genered by Java compiler can run on any platform with the help of JVM (Java Virtual Machine)



- The above .class files are in various platforms, basically we can use/execute the .class file which is generated by the compiler on any platform.

components :

1) JRE (Java Runtime Environment) :
- It's a component in JDK which includes JVM, core classes & supporting classes
- It's used to run the Java applications

2) JavaC (Java compiler) :
- Converts Java source code into byte code that can be understood by the JVM.

3) Java Debugger :
- A tool which is used to identify & fix errors in java code.

Q2) Difference betn JDK, JRE, JVM.

→ • JDK (Java Development Kit):

purpose: Used for developing Java applications

Components: JVM + JRE + compiler + debugger + other development tools

Functionality: Provides comprehensive set of tools & libraries necessary for developing, compiling, testing & deploying Java Application.

• JRE (Java Runtime Environment):

purpose: It is used to compile the Java applications

Components: JVM + Java class library

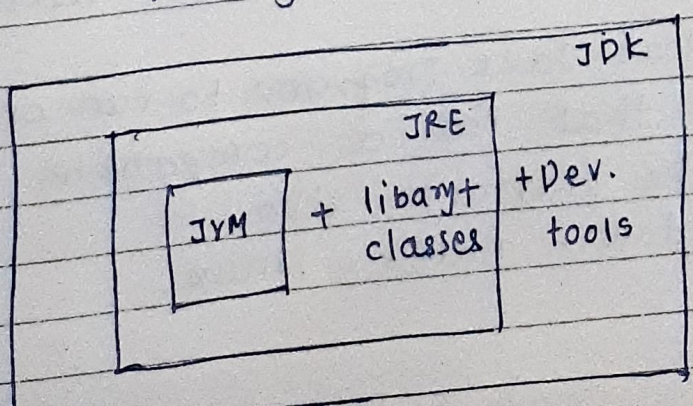
Functionality: Provide Java applications can run on different operating systems without modification.

• JVM (Java Virtual Machine):

Purpose: It interprets & executes Java bytecode

Functionality: JIT (Just In Time) compilation optimizes Java bytecode for faster execution by translating it into machine code at runtime

- Automatically allocates & deallocates memory (Garbage collection)



Q.3) Role of JVM in Java? How does JVM executes the Java code?

→ • Main role of JVM is, when a Java compiler compiles, Java source code → javac → Byte code.

The byte code is converted to machine-specific code with JIT compiler.

so,

Byte code → JIT compiler (JVM) → Machine specific code

-JIT compiler is used to executes parts of codes, in a faster way compared to the CPU.

• JVM automatically manages memory, allocation & deallocation with the help of garbage collector - It helps to prevent memory leaks & improves application performance.

• JVM provides thread management where it can create or manage the threads, allows Java applications to perform multiple tasks concurrently.

• JVM allows Java programe to run on any platform that has a compatible JVM installed as per the platform like, windows, macOs, Linux.

Q.4) Memory Management system of JVM.

→ - JVM automatically manages memory using garbage collection.
- It identifies & reclaims the unused memory.
- When an object is created, it acquires some space If an object is no longer reachable, it's considered as a garbage value.
- Some garbage collectors may also perform compaction, which involves moving live objects together in memory to reduce fragmentation & improve memory utilization.

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

→ - A JIT compiler basically converts a byte code into machine specific code.
- It is a part of JVM which is used to analyze & compiles the code and executes in no. of parts. which makes it faster executable than CPU.
- Basically, bytecode is an intermediate language that is generated when Java source code is compiled.
- As JIT is platform independent means it can be executed on any system that has a compatible JVM.

Q.6) Architecture of JVM:

→ JVM is a complex piece of s/w that can be divided into several key components:

1) Class loader : Loads the class into memory.

2) Bytecode Verifier: checks the validity of the loaded bytecode

3) Interpreter : Executes the validity of the loaded bytecode.

4) JIT compiler: Optimizes bytecode by converting it to machine code at runtime.

5) Garbage collector : Manages memory allocation & deallocation.

6) Execution engine : Executes the machine code generated by the JIT compiler.

7) Native Interface: Allows java code to interact with native libraries.

Q.7) How Java achieves platform independent through the JVM?

→ - Java achieves platform independence through the JVM. The JVM acts as an abstract machine that translates Java byte code into machine specific code.

- Same Java bytecode can be executed on any platform that has a compatible JVM.

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

→ significance of class loader :

- It is an integral part of the Java Runtime Environment (JRE) that dynamically loads Java classes into the JVM.
- class loader controls the loading of classes from different resources.
- Helps to prevent malicious code execution.
- Also developers can create custom load classes to implement specific loading strategies.

Process of Garbage Collection:

step - 1 : It is controlled by a thread known as Garbage Collector.

step - 2 : Java provides two methods System.gc() & Runtime.gc() that sends request to the JVM for garbage collection.

step - 3 : If the heap memory is full (where the objects are stored), the JVM will not allow to create a new object & shows an error java.lang.OutOfMemoryError.

step - 4 : When garbage collector removes the object from memory. first, the garbage collector thread calls the finalize() method of that object & then remove.

Q.9) What are four access modifiers in Java? How they differ from each other?

→ - Access modifiers are used to control the visibility of the members of the class / enum / interface.

- There are 4 access modifiers in Java:
  1) private
  2) package level private (default)
  3) protected
  4) public

Public:
Visibility - Accessible from anywhere within the project including other classes, packages, modules.
Usage - Used for public API's, classes, method & variables that need to be accessible to external code.

Private:
Visibility - Accessible only within the same class.
Usage - Used for methods & variable that should be encapsulated within a class and not directly accessible from outside.

protected:
Visibility: Accessible within the same class, its subclasses and other classes within the same package.
Usage: Used for methods and variables that need to be accessible to subclass but should not be directly accessible to unrelated classes.

4) Default (package private):
Visibility - Accessible only within the same package

Q.11) Can you override a method with a different access modifier in subclass? ex. can a protected method in a superclass be overridden with a private method in subclass? explain

→ yes, it is possible to override a method with a different access modifier in subclass.

- Public - Can be overriden by public, protected, default or private.
- Protected - can be overridden by public, protected or default access modifiers.
- private - Cant be overridden.
- Default (package-private) - Can be Overridden by public, protected or default.

Q.12) Difference beth protected and default (package private) in Java?

→ Protected -
Visibility: Accessible within the same class, its subclasses & other classes within the same package.

Default -
Visibility: Accessible only within the 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?

→ - It's not possible to make a class private in Java.

~~limitations~~:
~~You can't access that class within the same package~~
- Because, classes in Java are inherintly public by default.
- This means they are accessible from anywhere in the project, including ~~ot~~ classes, packages, modules.

Q.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 as protected or private
- Top-level classes, which are not nested within any other class, must be declared as public or default.
- Protected & private access modifiers limit the visible of a class to specific contexts.

Q.15) What happens if you declare a variable or method as private in a class and try to access it from the another class within the same package?
→ - Compile error will occur
- In Java, private access modifier restricts the visibility of a variable or method to within the class where it is ~~considered~~ declared.

Q.16) Package private (default) access?
→ - Package private or default access in Java, limits visibility of class members to the same package.
- If no access modifier is specified, it's assumed to be package private.
- This provides a middle ground both public & private access, allows encapsulation.