Assignment - 3

1] Explain the components of JDK.

→ The Java Development kit is a software development environment provided by Oracle corporation, enabling developers to build Java applications and applets. It includes various tools, libraries, and executables required for Java development.
The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) etc. to complete the development of a Java application.

Components of JDK -
Primary Components of JDK -

① appletviewer - This tool is used to run and debug Java applets without a web browser.
② apt - It is an annotation-processing tool.
③ extcheck - It is a utility that detects JAR file conflicts.
④ idlj - An IDL-to-Java compiler. This utility generates Java bindings from a given Java IDL file.
⑤ jabswitch - It is a java access bridge. Exposes assistive technologies on Microsoft Windows systems.
⑥ java - The loader for Java applications. This tool is an interpreter and can interpret the class files generated by the javac compiler.

(7) javac - It specifies the Java compiler, which converts source code into Java bytecode.

(8) javadoc - The documentation generator, which automatically generates documentation from source code comments.

(9) jar - The specifies the archiver, which packages related class libraries into a single JAR file. This tool also helps to manage JAR files.

(10) javafxpacker - It is a tool to package and sign JavaFX applications.

(11) javap - the class file disassembler.

2] Differentiate between JDK, JVM, JRE -

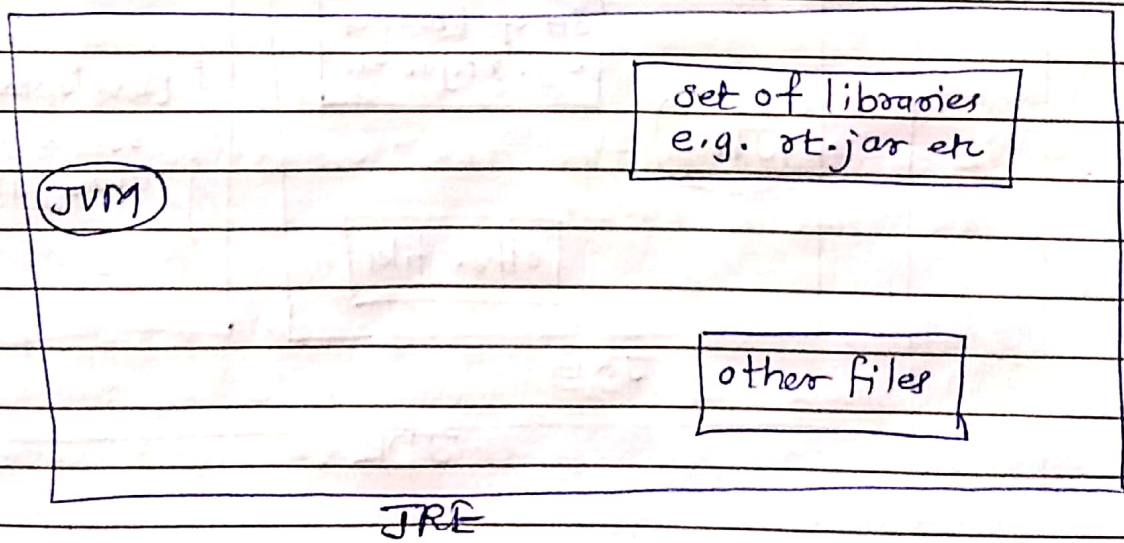
* JVM -

i) JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. ii) It is a specification that provides a runtime environment in which Java bytecode can be executed.
iii) It can also run those programs which are written in other languages and compiled to Java bytecode.
iv) JVMs are available for many hardware & software platforms.
v) JVM, JRE and JDK are platform dependent because the configuration of each OS is different from each other.
vi) However, Java is platform independent.

vii) JVM performs the following main tasks -
① Loads code
② Verifies code
③ Executes code.
④ Provides the runtime environment.

* JRE -

i) JRE is an ~~acronym~~ acronym for Java Runtime Environment.
ii) It is also written as Java RTE.
iii) The Java Runtime Environment. It is the implementation of JVM.
iv) It physically exists. It contains a set of libraries + other files that JVM uses at runtime.
v) The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



* JDK -

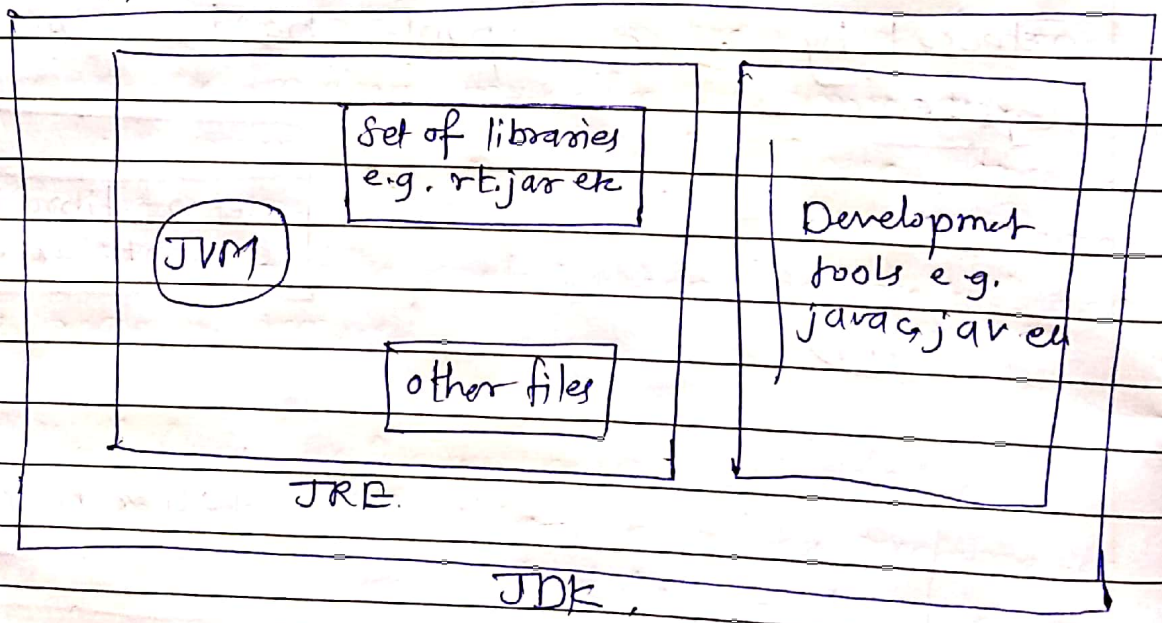
i) JDK is an acronym for Java Development kit.
ii) The Java Development Kit (JDK) is a software development Environment which is used to develope Java applications and applets. It physically exists.

iii) It physically exists. It contains JRE + development tools.

iv) JDK is an implementation of any one of the below given Java Platform release by Oracle Corporation.

Java SE
Java EE
Java ME

v) The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter / loader (java), a compiler (javac), an archiever (jar), a documentation generator (javadoc) etc. to complete the development of a Java Application.



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

→ i) JVM (Java Virtual Machine) runs Java applications as a run-time engine.

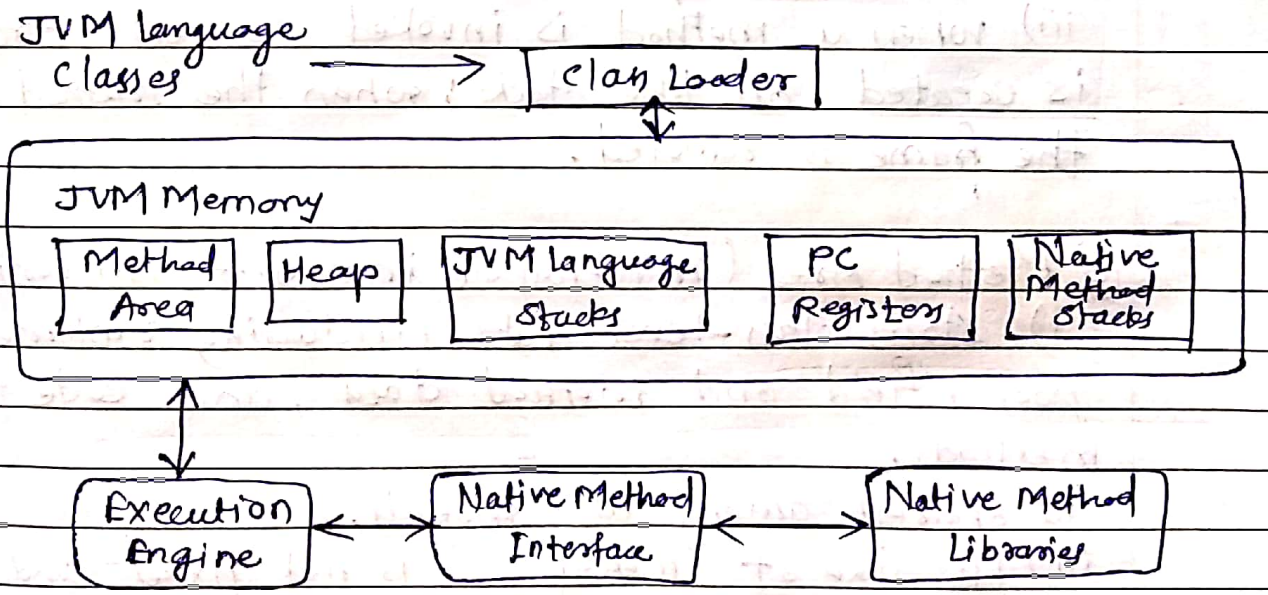
ii) JVM is one that calls the main method present in a Java code.

iii) JVM is a part of JRE (Java Runtime Environment).

iv) Java applications are called WORA (write Once Run Anywhere). This means a programmer can develope Java code one one system and expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM.

v) When we compile .java file, .class file (contains byte-code) with the same class names present in .java file are generated by the Java Compiler.

vi) This .class file goes into various steps when we run it.



4) Explain the memory management system of the JVM?

→ The JVM memory is divided into several areas, each serving a different purpose:

① Heap :
i) The heap is the runtime data area from which memory for all class instances and arrays is allocated.
ii) It's shared among all threads.
iii) Garbage Collection (GC) happens here, which automatically reclaims memory used by objects that are no longer reachable.

② Stack :
i) Each thread has its own stack, used for storing frames. A frame holds local variables, partial results, and data for method invocation and return.
ii) Stack memory is not shared between threads.
iii) When a method is invoked, a new frame is created on the stack; when the method completes, the frame is removed.

③ Method Area (or Metaspace in newer JVM versions) -
i) Stores class-level data, including runtime constant pool, field and method data, and code for methods.
ii) Shared among all threads.
iii) The size of metaspace is not fixed and can grow dynamically.

④ Program Counter (PC) Register :-
i) Each thread has its own PC register that keeps track of the JVM instruction being executed.

ii) If a thread is executing a Java method, the PC register holds the address of the JVM instruction. If it's a native method, the PC register is undefined.

⑤ Native Method stack –
i) Used for native methods (methods written in other than Java, like C/C++).
ii) Each thread has its own native method stack.

* Garbage Collection –
The JVM automatically manages memory using a process called Garbage Collection (GC), which reclaims memory by identifying and disposing of objects that are no longer referenced by the application.

GC Algorithms –
(1) Mark and Sweep
(2) Generational GC.
(3) G1 (Garbage-First)

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

→ i) JIT is a part of JVM that optimizes the performance of the application.
ii) JIT stands for Just In Time compiler.
iv) The JIT compilation is also known as dynamic compilation.
v) JIT is an integral part of JVM.
vi) It optimizes the performance of the Java application at compile time or run time.

* Bytecode - i) Bytecode is an intermediate, platform-independent code generated by the Java compiler after you compile your code .java files.

ii) Bytecode is not specific to any operating system or hardware, and it has the following key roles -

① Portability - Java's "Write Once, Run Anywhere" feature relies on the bytecode. Since bytecode is platform independent, it can run on any machine that has JVM.

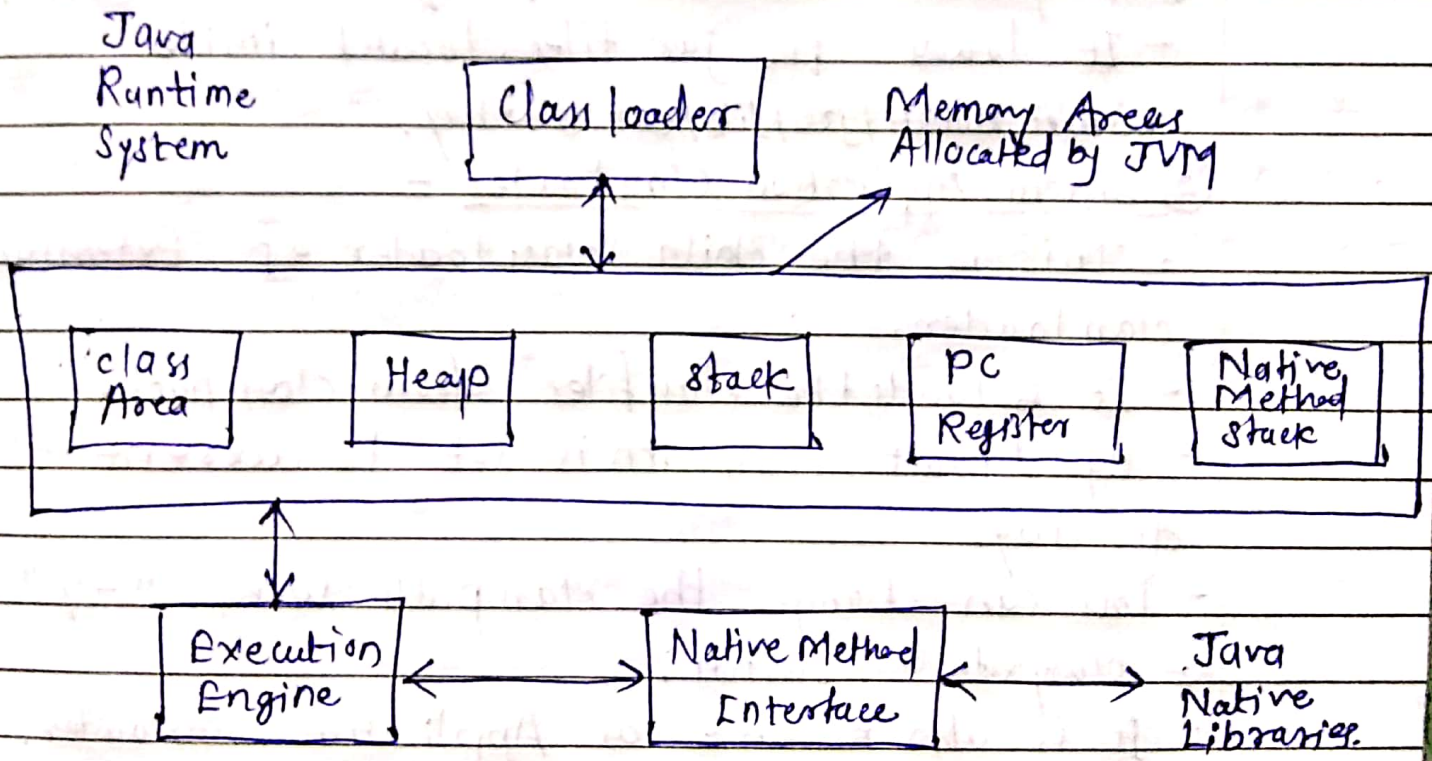
② Execution by JVM - The JVM reads bytecode and translates it into machine specific code (either through interpretation or with the help of JIT compiler). So your java program can run on any device that supports JVM

③ Security - Bytecode allows JVM to verify and monitor the execution of code, making Java more secure against malicious programs.

81.3

6] * Overview of JVM Architecture -

- JVM (Java Virtual Machine) is an abstract machine.
- It is a specification that provides runtime environment in which java bytecode can be executed.



- It contains classloader, memory area, execution engine etc.

"You cannot afford to take a break"

Page No.: Date: YOUVA

Ass 3

Q. 8 1] Class loader –

- classloader is a subsystem JVM which is used to load class files.
- whenever we run the java program, it is loaded first by classloader.
- There are three built-in class loaders in java :–

① Bootstrap classloader – (Highest Priority will be given to this)
- This is the first class loader which is the super class of Extension class loader.
- It loads the rt.jar file which contains all class files of Java standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes. etc.

② Extension class loader – (jre/lib) ← loading classes inside.
- This is the child class loader of Bootstrap and parent classloader of System class loader.
- It loads the jar files located inside $JAVA_HOME/jre/lib/ext directory.

③ System/Application Class Loader –
- This is the child class loader of Extension class loader.
- It is loads the class files from class path.
- By default, class path is set to current directory.
- You can change the class path using "-cp" or "-classpath" switch.
- It is also known as Application classloader.

* There are the internal class loaders provided by Java. If you want to create your own class loader, you need to extend the ClassLoader class.

2] Class (Method) Area -
Class (Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3] Heap -
It is the runtime data area in which objects are allowed.

4] Stack -
- Java stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.
- Each thread has a privat JVM stack, created at the same time as thread.
- A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5] Program Counter Register -
- Pc (program counter) register contains the address of the Java Virtual Machine instruction currently being executed.

6] Native Method stack -
- It contains all the native methods used in the application.

7] Execution Engine -
It contains →
① A virtual processor
② Interpreter - Read bytecode stream then execute the instructions.

③ Just-In-Time (JIT) compiler -
- It is used to improve the performance.
- JIT compiles parts of the bytecode that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.
- Here the term "compiler" refers to a translator from the instruction set of a specific CPU.

8] Java Native Interface - (JNI)
- JNI is a framework which provides an interface to communicate with another appl^n written in another language like C, C++, Assembly etc.
- Java uses JNI framework to send output to the console or interact with os libraries.

imp for ccbp exam.

Wrapper class Hierarchy —

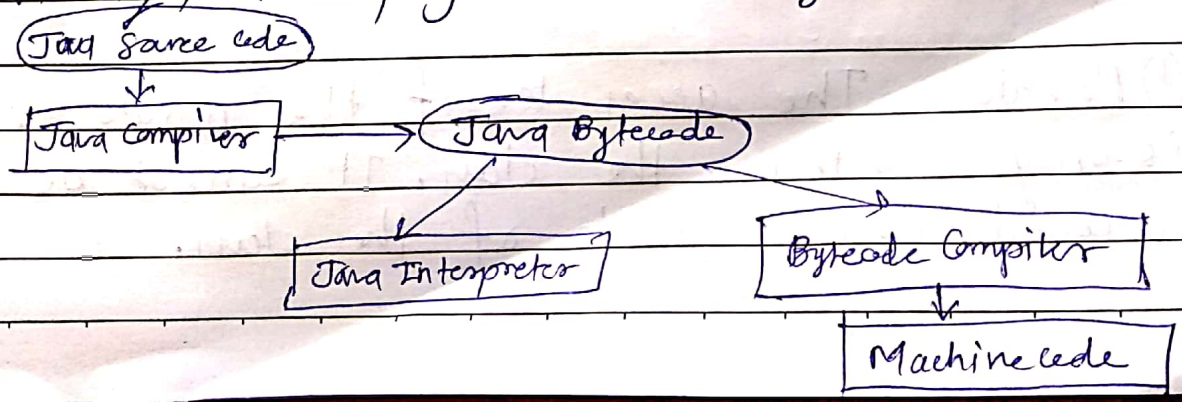
* Boxing conversion - Process of converting primitive to non-primitive.
* UnBoxing conversion - converting non-primitive to primitive.

7@] How does the Java achieve platform independence through JVM?

→ The meaning of Java platform-Independent is that the Java compiled (bytecode) can run on all operating systems.

Step-by step execution of Java program —

① whenever a program is written in Java, the java compiler it.
② The result of java compiler is the .class file or the bytecode.
③ The bytecode generated is a non-executable code and needs an interpreter to execute on a machine.
④ This Interpreter is the JVM and thus the Bytecode is executed by JVM.
⑤ Finally, the program runs to give the desired o/p.



why →

i) ① In Java, main point here is that JVM depends on the OS - so if you are running Mac OS X you will have a diff. JVM than if you are running Windows or some other OS.

② This fact can be verified by trying to download the JVM for your particular machine - when trying to download it, you will be given a list of JVM corresponding to different OS, and you'll obviously pick whichever JVM is targeted for the OS that you are running.

③ So, we can conclude that JVM is platform-dependent and it is the reason why Java is able to become "Platform Independent".

* In case of Java, it is the magic of Bytecode that makes it platform-independent.

Q.9. What are the four access ~~access~~ modifiers in Java?
Q.10. and how do they differ from each other?

→ There are 4 access modifiers in Java which is used to control visibility of the members declared inside Type (interface / class / enum).

① Private
② package level private (default)
③ protected.
④ public.

① Private - The access level of the private modifier is only within the class. It cannot be accessed from outside of the class.

② Default - The access level of the default modifier is only within the package. It cannot be accessed from outside of the package.

③ Protected - The access level of the protected modifier is within the package. It cannot be accessed from outside the package.

④ Public - The access level of the public modifier is everywhere. It can be accessed from within the class, outside the class, within the package & outside the package.

Q.11. Can you override a method with a different access modifier in a subclass? for example, can a protected method in a superclass be overriden with a private method in a subclass? Explain?

→ No, you cannot override a method with a different access modifier in a way that reduces its visibility in a subclass.

* When overriding a method in a subclass, the access modifier must either be the same or more permissive than the method in the superclass. You cannot make the access level more restrictive.

e.g. If ~~you~~ method in the superclass is protected, the overriden method in the subclass must be either protected or public, but not private.

Why? —

This is because overriding a method with a more restrictive access modifier would break the contract of object-oriented design.

Page No.:
Date:
YOUVA

Q12. What is the difference between protected and default (package-private) access?

→ The main difference between protected and default (also called package-private) access in Java lies in who can access the class members (methods or variables):

1. Protected Access Modifier -
   A member with the protected keyword can be accessed:
   ↳ within the same package (just like default/package private access)
   ↳ In subclasses, even if the subclass is in different package.

2. Default (Package-Private) Access :-
   A member without any access modifier (default access) is only accessible:
   ↳ within the same package.
   ↳ It cannot be accessed from subclasses if they are in different packages.

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

→ i) Yes, it is possible to make a class private in Java but with certain limitations. Specifically, only nested classes (also known as inner classes) can be declared private.

ii) You cannot make a top level class (a class not defined inside another class) private.

* Private Nested class →

A nested class can be declared private if it is inside another class.

This restricts the visibility of the nested class to only within the enclosing class.

No other class can directly access it, even if they are in the same package.

```
e.g. class OuterClass {
        private class InnerClass {
                public void display() {
                        sop ("this is private Inner class");
                }
        }
        public void accessInnerClass() {
                InnerClass inner = new InnerClass();
                inner.display();
        }
}

public class Test {
        psvm() {
                OuterClass outer = new OuterClass();
                outer.accessInnerClass();
        }
}
```

* Top-level Classes –

A top level class in Java cannot be private. It can only have the following access modifiers –

① public – can be accessed from anywhere.
② default – within same package.

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. Only public and default (package-private) access levels are allowed for top level classes.

Why -

① Top level classes are Global in scope -
↳ A top level class is visible to the entire package or globally (if declared public). Declaring it as private or protected would violate the accessibility principles of these modifiers.

② Private and Protected modifiers restrict accessibility.

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?

→ If you declare a variable or method as private in a class and try to access it from another class within the same package, you will get a compilation error because private members are only accessible within the same class where they are declared, and not from other classes, even if they are in the same package.

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

→ In Java, "package-private" (also known as "default" access) is the access level that

applies when no explicit access modifier (public, protected or private) is specified for a class, method or variable. This means that the member is accessible only within the same package.

* If in a class, method or variable has no access modifier (i.e. neither public, protected nor ~~default~~ private) it has default (package private) access.
* Members with the package-private access are visible within the same package but not outside the package, even to subclasses.
* Applied to classes: Only top-level classes and class members (variables, methods, constructors) can have package-private access. Nested classes can also be package private.