Q. Components of JDK

Appletviewer
- used to run & debug java applets without a web browser

apt
- It is an annotation processing tool

extcheck
- It is a utility that detects JAR file conflicts

idlj
- An IDL to Java Compiler. This utility generates Java buildings from a given JAVA IDL file.

jabswitch - It is a java access bridge. Exposes assistive technologies on Microsoft Windows system.

java - loader for java application.
- it is an intepreter which interprets class files that are generated by javac compiler.

javac - converts source code into bytecode.

javadoc - documentation generator
- generates documention from source code comments.

jar - It packages all related class libraries into single JAR file
- manage jar files.

javafxpackager - it is tool used to package & sign JavaFX application

jarsigner - jar signing & verification tool.

javah - used to write native methods.

javap - class file dessembler.

javaws - java web start launcher for JNLP application.

JConsole - Java monitoring & management console.

jdb - debugger.

jhat - java heap analysis tool

jinfo - get configuration info from running java process

jmc - java Mission control

jmap - outputs memory map for java.

jps - JVM Process Status tool.
- lists instrumented Hotspot JVM on target system.

jrunscript - java command line script shell.

jstack - prints java stack traces of java threads.

jstat - JVM statistics monitoring tool

jstatd - jstat daemon

keytool - tool to manipulate keystore

Pack200 - jar compression tool.

Policytool - determine policy for Java runtime.

* Difference betⁿ JDK, JRE & JVM

1) JDK - Java Developement Kit
- JDK is core component of JAVA environment, provides cui tools, executables & binaries required to compile, debug & execute a java program.
- JDK is platform dependent.
- JDK contain JRE with java compiler, debugger & core classes

2) JVM - It is a heart of Java programming language.
- converts byte code into ~~user~~ machine code.
- JVM is platform dependent
- It provides functions like memory management, garbage collection & security

3) JRE - It provides platform to execute Java program.

- JRE consist of JVM, Java binaries, & other classes to execute program succesfully.

1) JDK is for development purpose whereeas JRE for running program

2) JDK & JRE both contain JVM to run java program.

3) JVM is heart of Java & it ~~is~~ provides platform independence

* Role of JVM

- JVM allows java code to run on any device.
- Java source code is compiled by bytecode by the JVM.
- JVM manages memory by the process Garbage collection
- JVM provides secure execution environment.
- It provides robust mechanism for handling exception.

How JVM executes Java code

- When java appln, starts JVM loads bytecode into memory.
- Class loader loads the classes according to need.
- After loading JVM links classes.
- JVM initialize classes by executing static initializer.

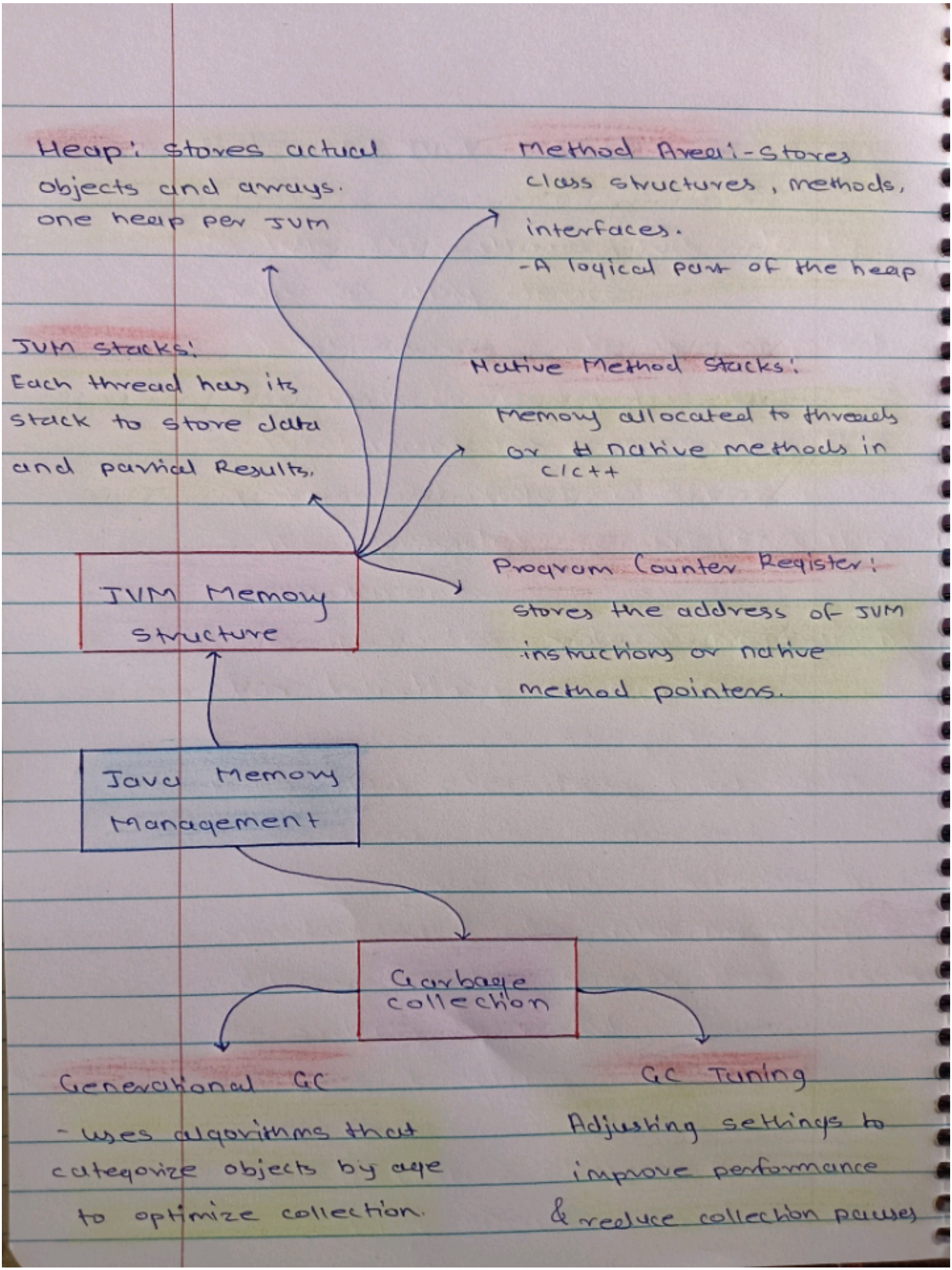
- JVM execute the bytecode by using either inteprebion or JIT compiler.

- Using Garbage collector, JVM continuosly manages memory.

- The JVM provides run time environment that includes support for Java libraries, APIs & standard execution environment.

* How does Java achive platform independence through JVM?

→ It first turns code into bytecode. This bytecode can run on any computer using JVM, which translates into specific machine code needed for that computer's OS & hardware. So, the Java programs work the same way on any device.


Heap: stores actual objects and arrays. one heap per JVM
Method Area: - Stores class structures, methods, interfaces.
- A logical part of the heap
JVM stacks: Each thread has its stack to store data and partial Results.
Native Method Stacks: Memory allocated to threads for native methods in C/C++
JVM Memory Structure
Program Counter Register: stores the address of JVM instructions or native method pointers.
Java Memory Management
Garbage collection
Generational GC
- uses algorithms that categorize objects by age to optimize collection.
GC Tuning
Adjusting settings to improve performance & reduce collection pauses

* JIT Compiler :-

1) Java Bytecode : Java programs are compiled into platform - independent bytecode.

2) JVM :- The JVM interprets this bytecode and converts it into native machine code

3) JIT compiler :- To improve performance, the JIT compiler kicks in at runtime. It compiles frequently executed bytecode sequences into native machine code

4) Native Code Execution :- Once compiled, the native code runs directly on the hardware, which is faster than interpreting bytecode repeatedly.

JVM : Java Virtual Machine

- JVM is platform independent execution environment.
- convert Java byte code into Mchine code & runs it on host OS.
- It is used to run java application on any system with compatible JVM.

JVM Architecture

1) class loader subsystem
   - loads Java classes (compiled byte code) into JVM.

loading - Find compiled java code (byte code) & loads it into memory.

Linking - it prepares that loaded code for execution and check every thing is running well.

Initialization - run static initializers & assign initial values to static field.

2) Runtime Data Areas :-

- used by

Memory areas that are used by JVM to manage data, while program is running.

Method Area

- stores class level information
- like class definations, method information and constants.

Heap

- objects and arrays are stored here
- It is shared across all parts of running program

Stack

- every time method is called, a new stack frame is created to store information
- like local variables & method's execution states.

PC registers :-

- keep track of current instruction bee being executed for each thread

Native method stack

- used when your java program calls methods written in other languages like c, c++.

3) Execution Engine :-

- it is part of JVM that actually runs your code.

Interpreter

- Reads & Execute bytecode line by line.

Just in time (JIT) compiler

- converts byte code into machine code at runtime to speed up the execution.

Garbage collector - manages memory by removing objects, that not needed.

- Java Native Interface (JNI)
  - allows java code to interact with code written in languages like c or c++.
  - useful when platform specific features are needed.
  - use to optimize performance.

Native method libraries
  - libraries written in languages other than Java, used by JVM through JNI to provide specific functionality.

- JVM implementation and performance Optimization

Hotspot (JVM
  - makes code run faster
  - makes memory management more efficient.

Open Jg JVM

- lightweight
- have small memory footprint
- Fast
- Scalable
- deal with environment with limited resources.

Graal VM

- support multiple programming languages (Java, JS, python).
- mixing multiple languages in one program
- converting bytecode to machine code before run time for Faster execution.

Summary :-

JVM helps in write better, more efficient, optimize code, better use of memory, solve problems come during development.

* What is the significance of class loader in Java? What is process of garbage collection?

→ ①

The class loader in java is responsible for loading classes into memory at runtime - It helps and find and bring in the necessary class files from file system, when they are needed. This process ensures that Java programs can run smoothly by dynamic loading classes as they are required & it also verifies & initializes these classes to ensure they are safe & ready for use.

(11) Garbage collection in Java automatically manages memory by cleaning up objects that are no longer needed by program. It identifies and marks these unused objects, then reclaims their memory for future use. This process helps prevent memory leaks & keeps the program running efficiently by ensuring that memory is used only for active object.

* four access modifier & how they differ from each other

->

Public - access from any other class.
protected - accessible within its own package & by subclasses
default - Accessible only within its own package.
private - Accessible only within the same class.

Q.* Can you overide a method with different access modifier in a subclass

→

- No, because of the rules of access modifier overiding.
- When method is overridden, the overriding method should in the subclass must have the same or more permissive access modifier than the method in superclass.
- Private methods are not accessible outside the class they are defined in.
- Protected methods are accessed within the same package & by subclass.

*

protected Access Modifier	default (Package-private) access modifier
- Accessible within its own package & subclass	- Accessible within only its subpackage.
- Accessed across the packages	- can not be accessed outside its own package
- can be accessed in subclasses, regardless of package	- can not be accessed by subclasses in different packages

* It is possible to make class private in Java?

→

- No, if it is top level class.
- A toplevel classes declared public, which makes it accessible from any other class
- A nested classes can declare as private. This means nested class is only accessible within enclosing class.

* Can top level class declared as protected or privat-

→ No, a top level class is declared as public so it can be accessible from any other class.

* What happens if you declare variable or method as a private & try to access it from another class.

→

- Then it can only accessed within the same class.
- Even if another class is in the same package, it can not directly access or modify private members of the class.
- This ensures that sensitive data & methods are protected and only accessible through controlled means within the class itself.

* default or package private access modifier.

→

If no access modifier is specified, the class members are accessible only within same package. This means other classes in the same package can see & use these members, but classes in different package can not access them. It provides a way to restrict access to package-specific classes & members, keeping them hidden from external packages with allowing interaction within the package.

Q. Components of JDK

Appletviewer
- used to run & debug java applets without a web browser

apt
- It is an annotation processing tool

extcheck
- It is a utility that detects JAR file conflicts

idlj
- An IDL to Java Compiler. This utility generates Java buildings from a given JAVA IDL file.

jabswitch - It is a java access bridge. Exposes assistive technologies on Microsoft Windows system.

java - loader for java application.
- it is an intepreter which interprets class files that are generated by javac compiler.

javac - converts source code into bytecode.

javadoc - documentation generator
- generates documention from source code comments.

jar - It packages all related class libraries into single JAR file
- manage jar files.

javafxpackager - it is tool used to package & sign JavaFX application

jarsigner - jar signing & verification tool.

javah - used to write native methods.

javap - class file dessembler.

javaws - java web start launcher for JNLP application.

JConsole - Java monitoring & management console.

jdb - debugger.

jhat - java heap analysis tool

jinfo - get configuration info from running java process

jmc - java Mission control

jmap - outputs memory map for java.

jps - JVM Process Status tool.
- lists instrumented Hotspot JVM on target system.

jrunscript - java command line script shell.

jstack - prints java stack traces of java threads.

jstat - JVM statistics monitoring tool

jstatd - jstat daemon

keytool - tool to manipulate keystore

Pack200 - jar compression tool.

Policytool - determine policy for Java runtime.

* Difference betⁿ JDK, JRE & JVM

1) JDK - Java Developement Kit
- JDK is core component of JAVA environment, provides cui tools, executables & binaries required to compile, debug & execute a java program.
- JDK is platform dependent.
- JDK contain JRE with java compiler, debugger & core classes

2) JVM - It is a heart of Java programming language.
- converts byte code into ~~user~~ machine code.
- JVM is platform dependent
- It provides functions like memory management, garbage collection & security

3) JRE - It provides platform to execute Java program.

- JRE consist of JVM, Java binaries, & other classes to execute program succesfully.

1) JDK is for development purpose whereeas JRE for running program

2) JDK & JRE both contain JVM to run java program.

3) JVM is heart of Java & it ~~is~~ provides platform independence

* Role of JVM

- JVM allows java code to run on any device.
- Java source code is compiled by bytecode by the JVM.
- JVM manages memory by the process Garbage collection
- JVM provides secure execution environment.
- It provides robust mechanism for handling exception.

How JVM executes Java code

- When java appln, starts JVM loads bytecode into memory.
- Class loader loads the classes according to need.
- After loading JVM links classes.
- JVM initialize classes by executing static initializer.

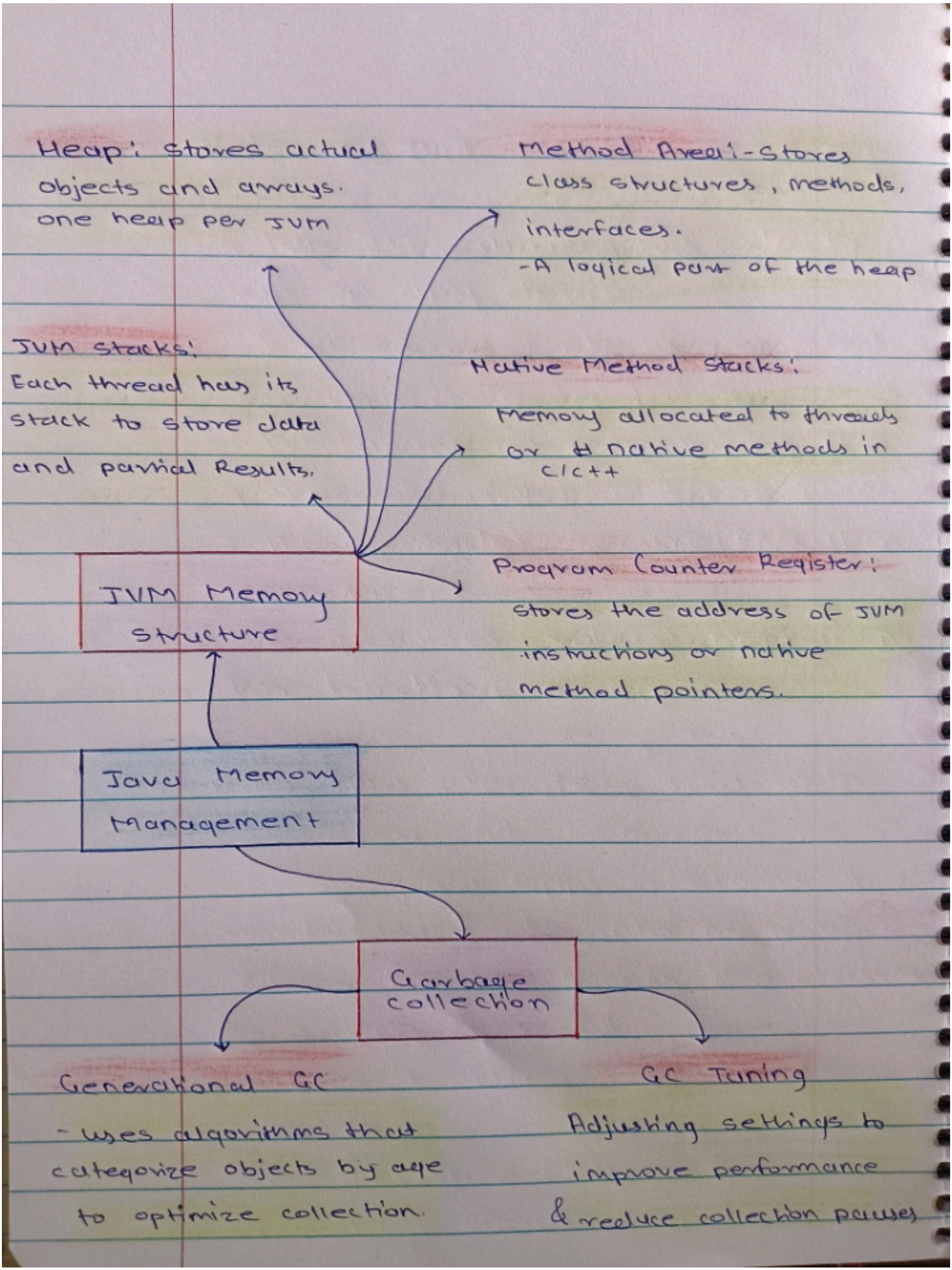
- JVM execute the bytecode by using either inteprebion or JIT compiler.

- Using Garbage collector, JVM continuosly manages memory.

- The JVM provides run time environment that includes support for Java libraries, APIs & standard execution environment.

* How does Java achive platform independence through JVM?

→ It first turns code into bytecode. This bytecode can run on any computer using JVM, which translates into specific machine code needed for that computer's OS & hardware. So, the Java programs work the same way on any device.


Heap: stores actual objects and arrays. one heap per JVM
Method Area: - Stores class structures, methods, interfaces.
- A logical part of the heap
JVM stacks: Each thread has its stack to store data and partial Results.
Native Method Stacks: Memory allocated to threads for native methods in C/C++
JVM Memory Structure
Program Counter Register: stores the address of JVM instructions or native method pointers.
Java Memory Management
Garbage collection
Generational GC
- uses algorithms that categorize objects by age to optimize collection.
GC Tuning
Adjusting settings to improve performance & reduce collection pauses

* JIT Compiler :-

1) Java Bytecode : Java programs are compiled into platform - independent bytecode.

2) JVM :- The JVM interprets this bytecode and converts it into native machine code

3) JIT compiler :- To improve performance, the JIT compiler kicks in at runtime. It compiles frequently executed bytecode sequences into native machine code

4) Native Code Execution :- Once compiled, the native code runs directly on the hardware, which is faster than interpreting bytecode repeatedly.

JVM : Java Virtual Machine

- JVM is platform independent execution environment.
- convert Java byte code into Mchine code & runs it on host OS.
- It is used to run java application on any system with compatible JVM.

JVM Architecture

1) class loader subsystem
   - loads Java classes (compiled byte code) into JVM.

loading - Find compiled java code (byte code) & loads it into memory.

Linking - it prepares that loaded code for execution and check every thing is running well.

Initialization - run static initializers & assign initial values to static field.

2) Runtime Data Areas :-

- used by

Memory areas that are used by JVM to manage data, while program is running.

Method Area

- stores class level information
- like class definations, method information and constants.

Heap

- objects and arrays are stored here
- It is shared across all parts of running program

Stack

- every time method is called, a new stack frame is created to store information
- like local variables & method's execution states.

PC registers :-

- keep track of current instruction bee being executed for each thread

Native method stack

- used when your java program calls methods written in other languages like c, c++.

3) Execution Engine :-

- it is part of JVM that actually runs your code.

Interpreter

- Reads & Execute bytecode line by line.

Just in time (JIT) compiler

- converts byte code into machine code at runtime to speed up the execution.

Garbage collector - manages memory by removing objects, that not needed.

- Java Native Interface (JNI)
  - allows java code to interact with code written in languages like c or c++.
  - useful when platform specific features are needed.
  - use to optimize performance.

Native method libraries
  - libraries written in languages other than Java, used by JVM through JNI to provide specific functionality.

- JVM implementation and performance Optimization

Hotspot (JVM
  - makes code run faster
  - makes memory management more efficient.

Open Jg JVM

- lightweight
- have small memory footprint
- Fast
- Scalable
- deal with environment with limited resources.

Graal VM

- support multiple programming languages (Java, JS, python).
- mixing multiple languages in one program
- converting bytecode to machine code before run time for Faster execution.

Summary :-

JVM helps in write better, more efficient, optimize code, better use of memory, solve problems come during development.

* What is the significance of class loader in Java? What is process of garbage collection?

→ ①

The class loader in java is responsible for loading classes into memory at runtime - It helps and find and bring in the necessary class files from file system, when they are needed. This process ensures that Java programs can run smoothly by dynamic loading classes as they are required & it also verifies & initializes these classes to ensure they are safe & ready for use.

(11) Garbage collection in Java automatically manages memory by cleaning up objects that are no longer needed by program. It identifies and marks these unused objects, then reclaims their memory for future use. This process helps prevent memory leaks & keeps the program running efficiently by ensuring that memory is used only for active object.

* four access modifier & how they differ from each other

->

Public - access from any other class.
protected - accessible within its own package & by subclasses
default - Accessible only within its own package.
private - Accessible only within the same class.

Q.* Can you overide a method with different access modifier in a subclass

→

- No, because of the rules of access modifier overiding.
- When method is overridden, the overriding method should in the subclass must have the same or more permissive access modifier than the method in superclass.
- Private methods are not accessible outside the class they are defined in.
- Protected methods are accessed within the same package & by subclass.

*

protected Access Modifier	default (Package-private) access modifier
- Accessible within its own package & subclass	- Accessible within only its subpackage.
- Accessed across the packages	- can not be accessed outside its own package
- can be accessed in subclasses, regardless of package	- can not be accessed by subclasses in different packages

* It is possible to make class private in Java?

→

- No, if it is top level class.
- A toplevel classes declared public, which makes it accessible from any other class
- A nested classes can declare as private. This means nested class is only accessible within enclosing class.

* Can top level class declared as protected or privat-

→ No, a top level class is declared as public so it can be accessible from any other class.

* What happens if you declare variable or method as a private & try to access it from another class.

→

- Then it can only accessed within the same class.
- Even if another class is in the same package, it can not directly access or modify private members of the class.
- This ensures that sensitive data & methods are protected and only accessible through controlled means within the class itself.

* default or package private access modifier.

→

If no access modifier is specified, the class members are accessible only within same package. This means other classes in the same package can see & use these members, but classes in different package can not access them. It provides a way to restrict access to package-specific classes & members, keeping them hidden from external packages with allowing interaction within the package.