Basics of Java

What is Java ?

Java is a programming language and a platform. Java is a high level, object oriented and secured programming language.

Features of Java :-

The primary objective of Java Programming language creation was to make it portable, simple and secure programming language.

a) Simple - Java is easy to learn and its Syntax is simple clean and easy to understand. Java has removed many complicated and rarely-used features like pointers.

b) Object-oriented - Java is an object oriented programming language. Everything in Java is object. It follows the principles like : Encapsulation, Abstraction, Inheritance and Polymorphism.

c) Platform Independent - Java is platform independent i.e. WORA (Write once Run Anywhere). Platform Independent means it can compile on one platform and can get executed on other platform.

d) Secured - Java is secured because it has no concept of pointers and Java programs run inside a virtual machin

e) Robust - Java is robust because it has strong memory management, it provides automatic garbage collection and it has exception handling and type checking mechanism.

f) Portable - Java is portable because it facilitates us to carry the Java byte code to any platform.

g) Architecture-neutral - Java is architecture neutral because there are no implementation dependent features, for example size of primitive datatype is fixed.

h) High Performance - Java is faster than other traditional interpreted programming language because Java byte code is "close" to native code.

i) Distributed - Java is distributed because it facilitates users to create distributed application in Java. This features of Java makes us able to access files by calling the methods from any machine on the internet

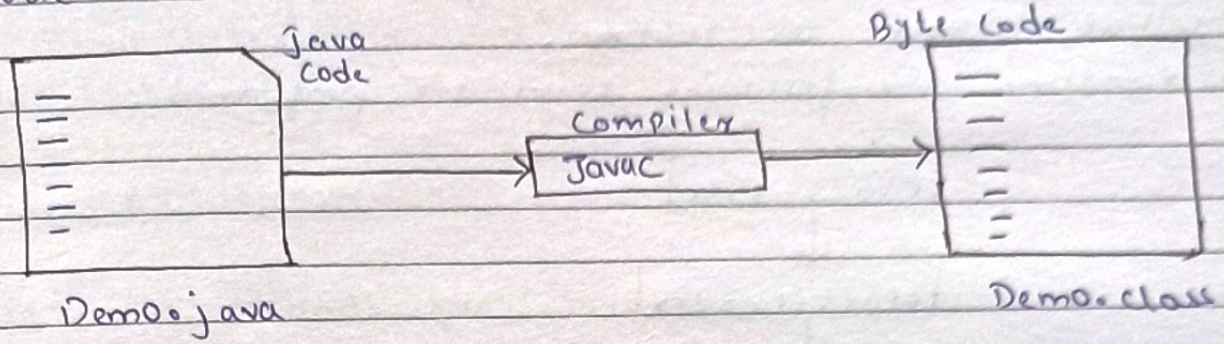
j) MultiThreaded - We can write Java programs that deal with many tasks at once by defining multiple threads.

k) Dynamic - Java is a dynamic language as it supports dynamic loading of classes. It also supports dynamic compilation and automatic memory management.

First Java Program:

```
Class Demo {
    public static void main (String [] args) {
        System.out.println ("Hello Java");
    }
}
```

Compilation flow:
When we compile Java Program using Javac tool, the java compiler converts the source code into byte code.



Parameters used in first Program:-

a) class - It is a keyword used to declare a class in Java.

b) Public - It is a keyword, which is also an access modifier that represents visibility.

c) static - It is a keyword whose advantage is if we make any method as static than there is no need to create an object to invoke the static method. The main() method is executed by JVM, so it doesn't require creating an object to invoke the main method.

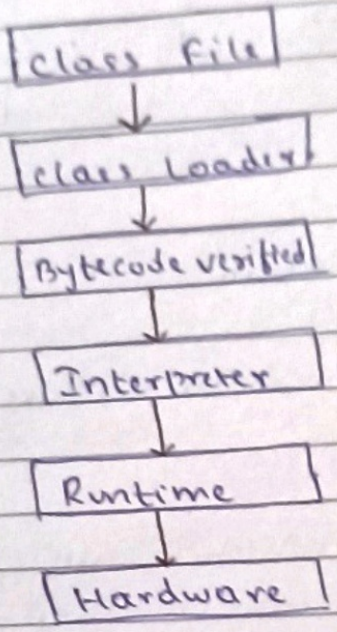
d) void - It is a return type. It means it does not return any value.

e) main - It represents the starting point of the program.

f) String[] args or string args[] - It is used for command line argument.

g) System.out.println() - It is print statement. Here System is a class, out is an object of the PrintStream class and println() is a method of Printstream class

Q. What happens at runtime?
At runtime the following steps are performed:



- Class Loader - It is the subsystem of JVM that is used to load class file.

- Bytecode verifier - checks the code fragments for illegal code that can violate access rights to objects.

- Interpreter - Read bytecode stream then execute the instructions.

Q. Why Class Name and File name should be same?
We are allowed to use any name for file name only when class is not public.
Suppose when you create a program in which more than

one class resides. And after compiling Java source file it will generate same number of .class file as classes resides in your program. In this condition you will not able to easily identify which class need to interpret by Java interpreter and which class consisting Entry point (Main Method).

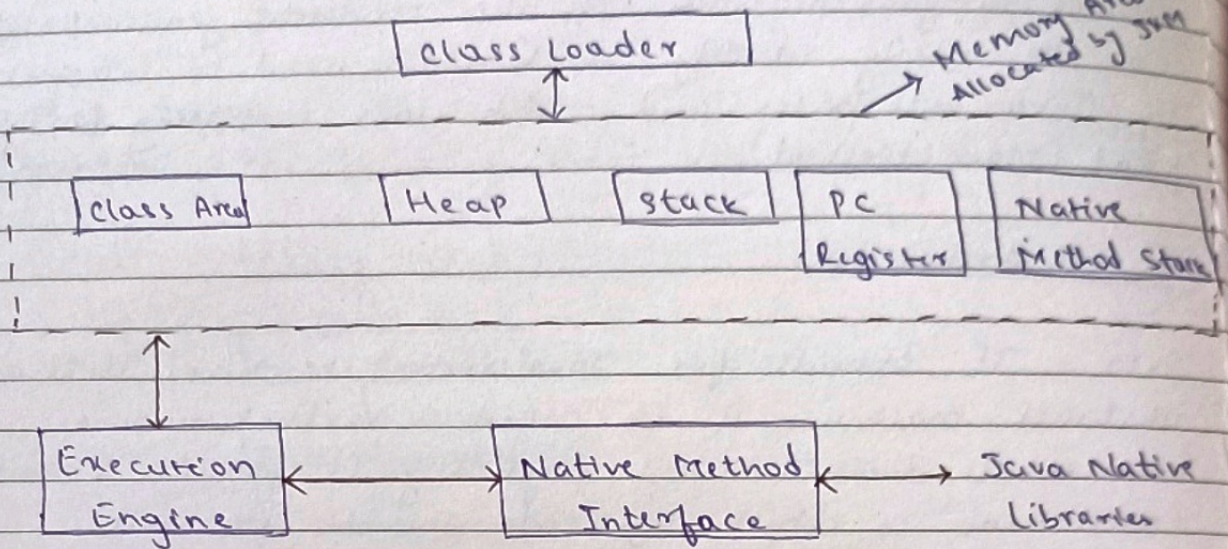
Java Architecture :-

Jvm - It stands for Java virtual Machine. It is an abstract machine. It is called a virtual machine because it doesn't exist physically. It provides a runtime environment in which byte code can be executed. Jvm are available for many hardware and software platforms.

What is does?
Jvm performs following operations :
a) Loads code
b) verifies code
c) Executes code
d) Provides runtime environment

Jvm provides definitions for the :
a) Memory area
b) class file format
c) Register set
d) Garbage - collected heap
e) Fatal error reporting etc.

JVM Architecture :-



a) Class Loader - It is a subsystem of JVM which is used to load class file. Whenever we run the java program it is loaded first by the class loader. There are three built-in classloaders in Java -

i) Bootstrap class Loader : This is the first class loader which is the super class of Extension class loader. It loades rt.jar file which contains java.lang package, java.net package, java.util package, java.io package, java.sql package, etc

ii) Extension class Loader : This is the child class loader of Bootstrap and parent class loader of system class loader. It loades the jar file.

iii) System class Loader : This is a child class loader of Extension class Loader. It loades the classfiles from classpath. By default, classpath is set to current

directory.

Example:

```
public class classLoaderExample {
    public static void main (String[] args) {
        Class c = classLoaderExample.class;
        System.out.println ( c.getClassLoader ());
        System.out.println (String.class.getclass
                                Loader ());

        }
    }
```

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

c) Heap - It is the runtime data area in which objects are allocated.

d) Stack - It stores frames. It holds local variables and partial results, and plays a part in method invocation and return. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

e) Program counter Register - It contains the address of the JVM instruction currently being executed.

f) Native Method stack - It contains all the native methods used in the application.

g) Execute Engine - It contains

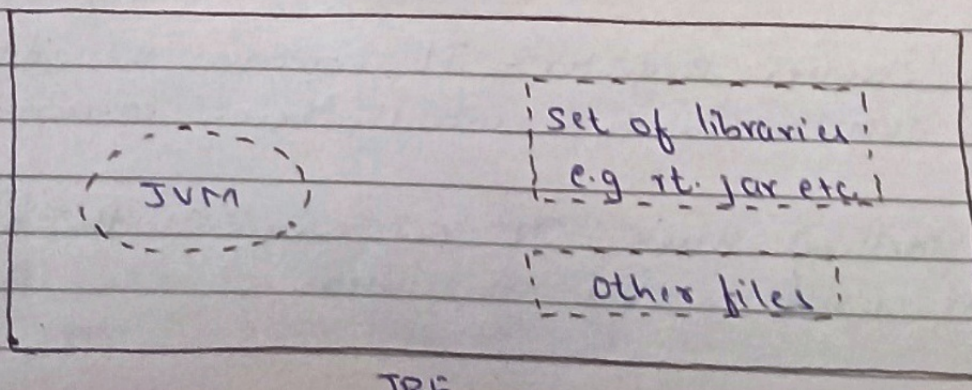
i) A virtual Processor.

ii) Interpreter : Read bytecode stream then execute the instructions.

iii) Just-in-Time (JIT compiler) : It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term 'compiler' refers to the translator from the instruction set of a JVM to the instruction set of a specific CPU.

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

JRE - It stands for Java Runtime Environment. It is also written as Java RTE. It is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.



JDK - It stands for Java Development Kit It is a software development enviroment which is used to develop java applications and applets. It physically exists. It contains JRE + development tools.
JDK contains a private JVM and few other resources such as an 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:
Following is a list of primary components of JDK -

a) appletviewer - This tool is used to run and debug Java applets without a web browser.

b) ~~appt~~ apt - It is an annotation - processing tool

c) extcheck - It is a utility that detects JAR file conflicts.

d) idlj - An IDL-to-Java compiler. This utility generates Java bindings from a given Java IDL.

e) jabswitch - It is a Java Access Bridge. Exposes assistive technologies on Microsoft Windows System.

f) java - The loader for Java Application. This tool is an interpreter and can interpret the class files generated by the javac compiler. Now a single launcher is used for both development and deployment. The old deployment launcher, jre, no longer comes with Sun JDKs, and instead it has been replaced by this new

java loader.

g) javac - It specifies the java compiler, which converts source code into java bytecode

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

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

j) javafxpackager - It is a tool to package and sign JavaFx applications.

k) jarsigner - the jar signing and verification tool.

l) javah - the c header and sub generator, used to write native methods.

m) javap - The class file disassembler

n) javaws - The java web start launcher for JNLP applications

o) Jconsole - Java monitoring and Management Console

p) Jdb - The debugger

q) jhat - Java Heap Analysis Tool

r) jinfo - This utility gets configuration information from

a running Java process or crash dump.

s) jmap – oracle jmap - Memory Map - This utility outputs the memory map for Java and can print shared object memory maps or heap memory details of a given process or core dump.

t) jmc - Java Mission Control

u) jps - Java virtual Machine Process Status Tool lists the instrumented HotSpot Java virtual Machine on the target System.

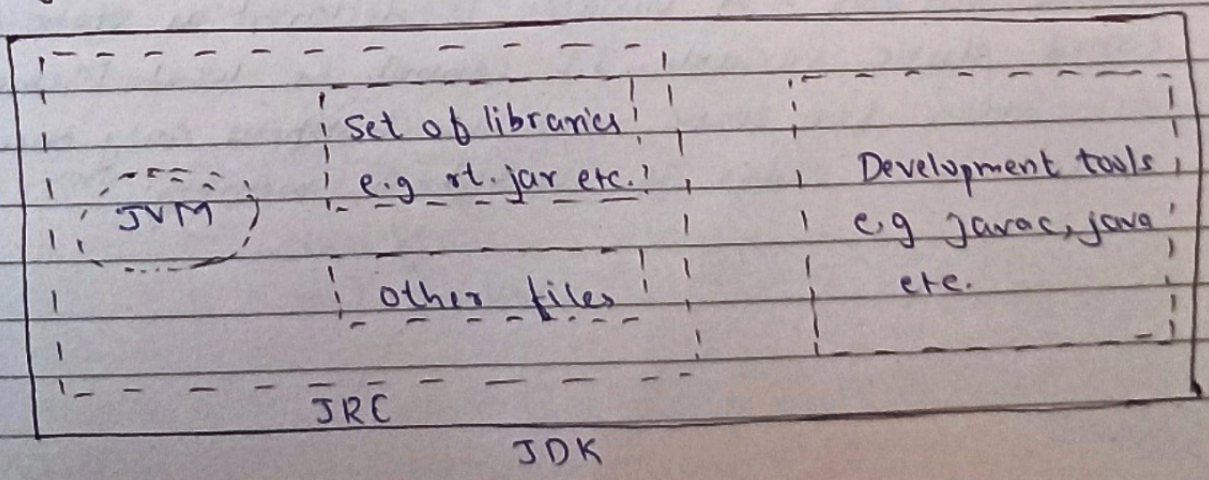
v) jrunscript - Java command line script shell.

w) jstack - It is a utility that prints Java stack traces of Java threads.

x) jstat - Java virtual Machine statistics monitoring tool.

y) jstad - jstat daemon

z) keytool - It is a tool for manipulating the keystore.



PAGE NO.:
DATE.

Java variables :-

A variable is a container that holds the value while the Java Program is executed. A variable is assigned with a datatype.
Variable is a name of memory location. There are three types of variables in java - local, instance and static.

Variable - It is a name of a reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary" + "able" which means its value can be changed.

```
int data = 50; // Here data is variable
```

Types of variables :-

a) Local variable - A variable declared inside the body of the method is called local variable. A local variable cannot be defined with "static" keyword

b) Instance variable - A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

c) Static variable - A variable is declared as static is called static variable. It cannot be local. Memory allocation for static variables happens only once when the class is loaded in memory.

Example:

```
public class Demo {
        static int m = 100; // static variable
        void method () {
                int n = 90; // local variable
            }
        public static void main (String[] args) {
                int data = 50; // local variable
            }
}
```

Data Types in Java :-

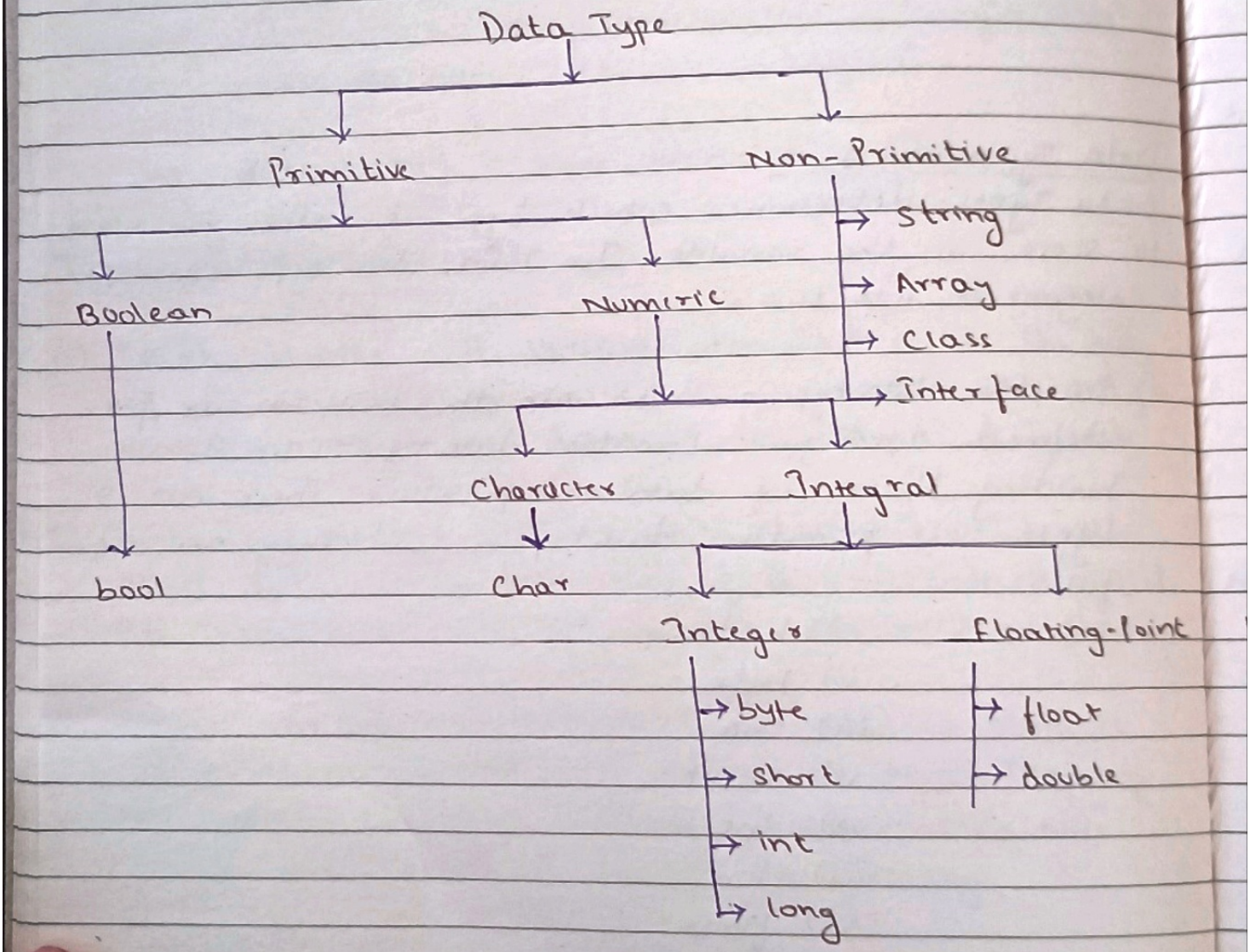
Data Types determines which type of value you want to store in the variable. In Java, data-types are categorized into two :-

a) Primitive data type - These are the built-in or pre-defined datatypes. Primitive datatypes are the building blocks of data manipulation. There are 8 types of primitive datatypes, and they are as follows -

i) boolean
ii) byte
iii) char
iv) short
v) int
vi) long
vii) float
viii) double

b) Non-primitive datatypes - These are user-defined datatypes in Java. Some of the non-primitive data types are -

i) String
ii) Array
iii) class
iv) Interface etc.



Data Type	Size	Default value	Range
boolean	Undefined (depends on JVM)	false	true or false
char	2 bytes	'\u0000'	0 to 65536
byte	1 byte	0	-128 to 127
short	2 bytes	0	-32768 to 32767
int	4 bytes	0	-2147483648 to 2147483647
long	8 bytes	0L	-2^{63} to $2^{63}-1$
float	4 bytes	0.0f	—
double	8 bytes	0.0	—

Boolean Data Type : It is used to store only two possible values true or false. It is used for simple flags that track true / false condition. The size of boolean datatype is virtual machine - dependent. values of boolean are not converted implicitly or explicitly to any other type.

Syntax:- boolean variable_name = value;
Example:- boolean flag = true;

PAGE NO.:
DATE:

Byte Data Type: It is an 8-bit signed two's complement integer. Its minimum value is -128 and maximum value is 127. Its default value is 0.
This data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer.

Syntax: byte variable_name = value;
Example: byte a = 10;

Short Data Type: It is a 16-bit signed two's complement integer. Its minimum value is -32768 and maximum value is 32767. Its default value is 0.
This data type can also be used to save memory just like byte data type, it is 2 times smaller than an integer.

Syntax: short variable_name = value;
Example: short a = 10000;

Int Data Type: It is a 32-bit signed two's complement integer. Its minimum value is -2147483648 (-2^{31}) and maximum value is 2147483647 ($2^{31}-1$). Its default value is 0.
This data type is generally used as a default data type for integral values unless if there is no problem about memory.

Syntax: int variable_name = value;
Example: int a = 12473;

Long Data Type: It is a 64-bit two's complement integer. Its minimum value is -2^{63} and its maximum value is 2^{63-1}. Its default value is 0.
The long datatype is used when we need a range of values more than those provided by int.

Syntax: long variable_name = value(L);
Example: long a = 41236790L;

Float Data Type: It is a single-precision 32-bit IEEE 754 floating point. Its value range is unlimited. If we need to save memory in large arrays of floating point numbers then we use float data type.

Syntax: float variable_name = value(f);
Example: float a = 234.5f;

Double Data Type: It is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited.

Syntax: double variable_name = value;
Example: double a = 12.3;

Char Data Type: It is a single 16-bit unicode character. Its value-range lies between '\u0000' (or 0) to '\ffff' (or 65,535). It is used to store characters. Characters should always be enclosed in single quote (' ').

Syntax: char variable_name = value;
Example: char a = 'A';

Q. Why char uses 2 byte in Java and what is \u0000?

Other languages like C / C++ use only ASCII characters, and to represent all ASCII characters 8 bits is ~~not~~ enough. But Java uses the unicode system not the ASCII code system and to represent the unicode system 8 bit is not enough to represent all character so Java uses 2 bytes of characters.

\uxxxx, it is the lowest range of unicode system.

Example :-

```
public class DataTypes {
    public static void main (String[] args){
        char a = 'G';
        int i = 89;
        byte b = 4;
        short s = 56;
        double d = 4.3554533532;
        float f = 4.7333434f;
        long l = 12121L;
        System.out.println ("char :" +a);
        System.out.println ("integer :" +i);
        System.out.println ("byte :" + b);
        System.out.println ("short :" + s);
        System.out.println ("float :" + f);
        System.out.println ("double :" + d);
        System.out.println ("long :" + l);
    }
}
```

PAGE NO:
DATE:

Unicode System :-

Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.

Lowest value : \u0000
Highest value : \uffff

Q) Why Java uses both unicode and ASCII values?
Java uses both Unicode and ASCII because it is a globally used programming language that needs to support a wide range of characters.

ASCII uses 7 bits representation that means it can only represent ~~over~~ 128 characters, including letters, numbers and some symbols. Unicode uses 16 bits representation that means it can represent over 65,000 characters, including ~~letters~~ characters from most of the world's languages.

Java uses unicode internally to represent characters. However, it also uses ASCII for some purposes, such as when it is communicating with other programs that only supports ASCII.

Operators in Java :-

Operators are the special symbols used to perform a specific task only when placed atleast in between two operands.

Types of operators:
There are many types of operators in Java:
a) unary operator
b) Arithmetic operator
c) Relational operator
d) Logical operator
e) Ternary operator
f) Assignment operator
g) Bitwise operator.

Unary operator - Java unary operators require only one operand. These are of two types -
i) increment
ii) decrement

increment :- It is categorized into two types post-increment and pre-increment.

```
int a = 10;
System.out.println(a++); // 10
system.out.println(++a); // 12
```

a++ (means post-increment) - It says first assign the value then increment it by 1 and store in memory

++a (means pre-increment) - It says first increment the value by 1 and then store it in memory.

decrement :- It is categorized into two types post-decrement and pre-decrement.

```
int a = 10;
System.out.println(a--); 10
System.out.println(--a); 8
```

a-- (means post-decrement) - It says first assign the value then decrement it by 1 and store in memory.

--a (means pre-decrement) - It says first decrement the value by 1 and then store it in memory.

Arithmetic Operator - These operators involve the mathematical operators that can be used to perform various simple or advance arithmetic operations on the primitive datatypes referred to as the operands.

i) Addition (+) :- This operator is a binary operator and is used to add two operands.

```
int num1 = 10;
int num2 = 20;
int sum = num1 + num2;
System.out.println(sum);
```

ii) Subtraction (-) :- This operator is a binary operator and is used to subtract two operands.

```
int num1 = 20;
int num2 = 10;
int sub = num1 - num2;
System.out.println(sub);
```

iii) Multiplication(*) :- This operator is a binary operator and is used to multiply two operands.

```
int num1 = 10;
int num2 = 20;
int mult = num1 * num2;
System.out.println(mult);
```

iv) Division(/) :- This is a binary operator that is used to divide the first operand by the second operand and give the quotient as result.

```
int num1 = 20;
int num2 = 10;
int div = num1 / num2;
System.out.println(div);
```

v) Modulus (%) :- This is a binary operator that is used to return the remainder when the first operand is divided by the second operand.

```
int num1 = 5;
int num2 = 2;
int mod = num1 % num2;
System.out.println(mod);
```

Assignment operator :- These operators are used to assign values to the variable. The left side operand of the assignment operator is a variable and the right side operand of the assignment operator is a value.

Types of Assignment operator:

i) Simple Assignment operator :- It is used with the "=" sign where the left side consist of the operand and right side consist of a value.

```
variable = value;
int x = 10;
```

ii) Compound Assignment operator :- The compound operator is used where +, -, *, and / is used along with the = operator.

a) += operator - This operator is a compound of '+' and '=' operators. It operates by adding the current value of the variable on the left to the value on the right and then assigning the result to the operand on the left.

```
num1 += num2; => num1 = num1 + num2.
```

b) -= operator - This operator is a compound of '-' and '=' operators. It operates by subtracting the variables value on the right from the current value of the ~~value~~ variable on the left and then assigning the result to the operand on the left.

```
num1 -= num2; => num1 = num1 - num2
```

c) *= operator - This operator is a compound of "*" and "=" operators. It operates by multiplying the current value of the variable on the left

to the value on the right and then assigning the result to the operand on the left.

num1 *= num2 ; => num1 = num1 * num2

d) /= operator - This operator is a compound of "/" and "=" operators. It operates by dividing the current value of the variable on the left by the value on the right and then assigning the quotient to the operand on the left.

num1 /= num2 ; => num1 = num1 / num2

Relational operators - These are the bunch of binary operators used to checks for relations between two operands, including equality, greater than, less than, etc.

i) Equal to (==) :- This operator is used to check whether the two given operands are equal or not. The operator returns true if the operand at the left-hand side is equal to the right-hand side, else false.

var1 == var2

```
int var1 = 10;
int var2 = 20;
int var3 = 10;
System.out.println (var1 == var2); // false
System.out.println (var1 == var3); // true
```

ii) Not equal to (!=) :- This operator is used to check whether the two given operands are equal or not. It functions opposite to that of the equal-to-operator. It returns true if the operand at the left-hand-side is not equal to the right-hand side, else false.

var1 != var2

```
int var1 = 10;
int var2 = 20;
int var3 = 10;
System.out.println(var1 != var2); // true
System.out.println(var1 != var3); // false
```

iii) Greater Than (>) :- This checks whether the first operand is greater than the second operand or not. The operator returns true when the operand at the left-hand side is greater than the right-hand side.

var1 > var2

```
int var1 = 10;
int var2 = 20;
int var3 = 10;
System.out.println(var1 > var2); // false
System.out.println(var2 > var3); // true
```

iv) Lesser Than (<) :- This checks whether the first operand is less than the second operand or not. The operator returns true when the operand on the left-hand side is less than the right-hand side.

var1 < var2

```
int var1 = 10;
int var2 = 20;
int var3 = 10;
System.out.println(var1 < var2); // true
System.out.println(var2 < var3); // false
```

v) Greater Than or Equal to (>=) :- This checks whether the first operand is greater than or equal to the second operand or not. The operand returns true when the operand at the left-hand side is greater than or equal to the right-hand side.

```
var1 >= var2
```

```
int var1 = 10;
int var2 = 20;
int var3 = 10;
System.out.println(var1 >= var2); // true
System.out.println(var1 >= var3); // true
```

vi) Lesser Than or Equal to (<=) :- This checks whether the first operand is less than or equal to the second operand or not. The operator returns true when the operand at the left-hand side is less than or equal to the right-hand side.

```
var1 <= var2
```

```
int var1 = 10;
int var2 = 20;
int var3 = 10;
System.out.println(var1 <= var2); // true
System.out.println(var1 <= var3); // true
```

Logical Operators :- It is used to perform Logical operations. They are used to combine two or more conditions.

i) AND operator (&&) :- This operator returns true when both the conditions under consideration are satisfied or are true. If even one of the two yields false, the operator returns false.

A	B	A&&B
True	True	True
True	False	False
False	True	False
False	False	false

```
int a = 10;
int b = 20;
int c = 20;
if ((a<b) && (b==c)){
    int d = a+b+c;
    }
System.out.println(d); //50
```

ii) OR operator (||) :- This operator returns true when one of the two conditions under consideration is satisfied or is true. If even one of the two yields true, the operator results true. To make the result false, both the constraints need to return false.

A	B	A||B
True	True	True
True	False	True
false	True	True
False	Fala	False

```
int a = 10;
int b = 20;
int c = 20;
if ((a<b) || (b>c)) {
        d = a+b+c;
    }
System.out.println(d); // 50
```

iii) NOT operator (!) :- This is a unary operator and returns true when the condition under consideration is not satisfied or is a false condition. If the condition is false, the operation returns true and when the condition is true, the operation returns false.

A	!A
True	False
False	True

```
int a = 10;
int b = 20;
System.out.println(!(a<b)); // false
System.out.println(!(a>b)); // true
```

Keywords - These are known as reserved words. These are the predefined words by Java so they cannot be used as variables or object name or class name. There are 52 keywords in Java.

49 keywords are used and 2 are not currently in use they are const and goto.
true, false and null look like keywords, but in actual they are literals.

A list of Java keywords are:

abstract	else	interface	strictfp
boolean	enum	long	super
break	extends	native	switch
byte	final	new	Synchronized
case	finally	null	this
catch	float	package	throw
char	for	private	throws
class	if	protected	transient
continue	implements	public	try
default	import	return	void
do	instanceOf	short	volatile
double	int	static	while

Java Control Statements :-

Java compiler executes the code from top to bottom. Java provides statements that can be used to control the flow of java code. It is one of the fundamental features of java, which provides a smooth flow of program.

Java provides three types of control flow statements

a) Decision Making statements
   i) if statements
   ii) switch statement

b) Loop statements
   i) do while loop
   ii) while loop
   iii) for loop
   iv) for-each loop.

c) Jump statements
   i) break statement
   ii) continue statement

Decision Making statements - These statements decide which statement to execute and when. These statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided.

Types of Decision Making statements :

1. If statement - The control of the program is diverted depending upon the specific condition. The condition of the if-statement gives a Boolean value, either true or false. There are four types of if-statements -

a) Simple if-statement : It is the most basic statement among all control flow statements in java. It evaluates a Boolean expression and enable the program to enter a block of code if the expression evaluates to true.

Syntax -

```
if (condition) {
    Statement; // Executes when condition
}                  is true
```

Example -

```
public class Student {
    public static void main (String[] args)
        int x = 10;
        int y = 12;
```

```
            if (x+y>20){
                System.out.println("x+y is greater");
            }
        }
    }
```

output: x+y is greater.

b) if-else statement: It is an extension of the simple if-statement, which uses another block of code, i.e., else-block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax -

```
if (condition){
    Statement 1; // executes when condition is
}                    true
else {
    Statement 2; // executes when condition is
}                    false.
```

Example -

```
public class Student {
    public static void main (String[] args){
        int x=10;
        int y=12;
        if (x+y<20) {
            System.out.println("x+y is greater");
        }
        else {
            System.out.println("x+y is lesser");
        }
    }
}
```

Output : x ty is lesser

c) if-else-if ladder : This statement contains the if statement followed by multiple else-if statements. We can say it is a chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true.

Syntax -

```
if (condition 1) {
        statement 1; // executes when condition 1 is
    }                        true
else if (condition 2) {
        statement 2; // executes when condition 2 is
    }                        true
else if (condition 3) {
        statement 3; // executes when condition 3 is
    }                        true
    .
    .
    .
else if (condition n) {
        statement n; // executes when condition n is
    }                        true
else {
        statement; // executes when all the
    }                        conditions are false
```

Example -

```
public class Student {
    public static void main (String [] args) {
        String city = "Delhi";
```

```
if ( city == "Meerut") {
        System.out.println(" city is Meerut");
    }
else if (city == "Noida") {
        System.out.println(" city is Noida");
    }
else if ( city == "Agra") {
        System.out.println(" city is Agra");
    }
else {
        system.out.println (city);
    }
}
```

Output : Delhi

d) Nested if-Statement : In nested if-statements, the if statement can contain if or if-else statement inside another if or else-if statement.

Syntax -

```
if (condition 1) {
    statement 1; // executes when condition 1 is true
  if (condition 2) {
        statement 2; // executes when both
    }                  condition 1 & 2 is true.
else {
        statement 3; // executes when
    }                  condition 1 is true and
}                      condition 2 is false.
```

Example -

```
public class Student {
    public static void main (String[] args) {
        String address = "Delhi, India";
        if (address.endsWith("India")) {
            if (address.contains("Meerut")) {
                System.out.println("City is Meerut");
            }
            else if (address.contains("Noida")) {
                System.out.println("City is Noida");
            }
            else {
                System.out.println(address.split(",")[0]);
            }
        }
        else {
            System.out.println("You are not living in India");
        }
    }
}
```

output: Delhi

Q. WAP to check a year is leap or not.

→ Leap year are the year which is divisible by 4 and 400 but not by 100.

CODE:

```
public class LeapYear {
    public static void main (String[] args) {
        int year = 2020;
```

```
if (((year%4==0)&&(year%100!=0))||(year%400==0))
{
    System.out.println("The year is Leap");
}
else {
    System.out.println("The year is not Leap");
}
}
}
```

Q. WAP to check a number is positive, negative or zero.

```
public class CheckNumber {
    public static void main (String[] args) {
        int num = -13;
        if (num > 0){
            System.out.println("Positive");
        }
        else if (num < 0) {
            System.out.println("Negative");
        }
        else {
            System.out.println("zero");
        }
    }
}
```

e) Switch Statement: The Java switch statement executes one statement from multiple condition. It is like if-else-if ladder statement.

Important points for switch statement :-

a) There can be one or N number of case values for a switch expression.

b) The case value must be of switch expression type only. The case value must be literal or constant. It doesn't allow variables.

c) The case values must be unique. In case of duplicate value, it renders compile-time error.

d) Java switch case expression must be of byte, short, int, long (with its wrapper class), enums and string.

e) Each case statement can have a break statement which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.

f) The case value have a default label ~~with~~ which is optional.

Syntax -

```
switch (expression) {
    case value1:
        statement1;
        break;  // optional
    case value2:
        statement2;
        break;  // optional
    .
    .
    .
    case valuen:
        statementn;
        break;  // optional
```

```
    default:
        statement; // executes when all cases are
    }                  not matched.
```

Example -

```
public class SwitchExample {
    public static void main (String[] args) {
        int num = 20;
        switch (num) {
            case 10:
                System.out.println ("10");
                break;
            case 20:
                System.out.println ("20");
                break;
            case 30:
                System.out.println ("30");
                break;
            default:
                System.out.println ("Enter a
                                    valid number");
            }
        }
    }
```

Output - 20

Q. WAP to find month based on the given input

```
public class SwitchMonth {
    public static void main (String[] args) {
        int month = 7;
        String monthString = "";
```

```
switch (month) {
    case 1:
        monthString = "1 - January";
        break;
    case 2:
        monthString = "2 - February";
        break;
    case 3:
        monthString = "3 - March";
        break;
    case 4:
        monthString = "4 - April";
        break;
    case 5:
        monthString = "5 - May";
        break;
    case 6:
        monthString = "6 - June";
        break;
    case 7:
        monthString = "7 - July";
        break;
    case 8:
        monthString = "8 - August";
        break;
    case 9:
        monthString = "9 - September";
        break;
    case 10:
        monthString = "10 - october";
        break;
    case 11:
```

```
            monthString = "11 - November";
            break;
        case 12:
            monthString = "12 - December";
            break;
        default:
            System.out.println("Enter a valid
                                month");
        }
        System.out.println(monthString);
    }
}
```

output - 7 - July

Q. WAP to check the given alphabet is vowel or consonant.

```
public class SwitchVowel {
    public static void main(String[] args) {
        char ch = 'o';
        switch (ch) {
            case 'a':
                System.out.println("Vowel");
                break;
            case 'e':
                System.out.println("Vowel");
                break;
            case 'i':
                System.out.println("Vowel");
                break;
            case 'o':
                System.out.println("vowel");
                break;
```

```
        Case 'U':
            System.out.println("vowel");
            break;
        Case 'A':
            System.out.println("vowel");
            break;
        Case 'E':
            System.out.println("vowel");
            break;
        case 'I':
            System.out.println("vowel");
            break;
        case 'O':
            System.out.println("vowel");
            break;
        case 'U':
            System.out.println("vowel");
            break;
        default:
            System.out.println("Consonant");
        }
    }
}
```

Output - vowel

Q. WAP to check if break statement is not used.

```
public class SwitchExample{
    public static void main(String[] args){
        int num = 20;
        switch(num){
            Case 10:
```

```
            System.out.println("10");
        case 20:
            system.out.println("20");
        case 30:
            system.out.println("30");
        default:
            System.out.println("Enter a valid number");
        }
    }
}
```

output - 20
30
Enter a valid number

Q. WAP to check if String value is taken in switch statement.

```
public class SwitchExample {
    public static void main (String[] args){
        String level = "Expert";
        int levelCheck = 0;
        switch (level) {
            case "Beginner":
                level = 1;
                break;
            case "Intermediate":
                level = 2;
                break;
            case "Expert":
                level = 3;
                break;
            default:
                level = 0;
```

```
        }
        System.out.println("Your level is :" + level);
    }
}
```

Output - Your level is : 3

Q. WAP to understand Nested switch statement.

```
public class NestedSwitch {
    public static void main (String[] args) {
        char branch = 'C';
        int collegeYear = 4;
        switch (collegeYear) {
            case 1:
                System.out.println("English, Maths, Science");
                break;
            case 2:
                switch (branch) {
                    case 'C':
                        System.out.println("Java");
                        break;
                    case 'E':
                        System.out.println("Microprocessor");
                        break;
                    case 'M':
                        System.out.println("Drawing");
                        break;
                }
                break;
            case 3:
                switch (branch) {
                    case 'C':
```

```
                System.out.println("Multimedia");
                break;
            case 'E':
                System.out.println("Micro Electronics");
                break;
            case 'M':
                System.out.println("ICE");
                break;
            }
            break;
        case 4:
            switch(branch){
                case 'C':
                    System.out.println("Data Communication");
                    break;
                case 'E':
                    System.out.println("Image Processing");
                    break;
                case 'M':
                    System.out.println("Thermal Engineering");
                    break;
                }
            break;
        }
    }
}
```

Output - Data Communication.

Q. WAP to print switch statement using wrapper class.

```
public class WrapperSwitch {
    public static void main(String[] args) {
        Integer age = 18;
        switch (age) {
        case (16):
            System.out.println("You are underage");
            break;
        case (18):
            System.out.println("You are eligible for vote");
            break;
        case (65):
            System.out.println("You are senior citizen");
            break;
        default:
            System.out.print("Please give the valid age");
        }
    }
}
```

Output - You are eligible for vote

Loop Statements - If we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. We have three types of loops in Java -

a) do-while loop
b) while loop
c) for loop

a) do-while loop : This statement is used to iterate a part of the program repeatedly, until the specified condition is true. If the number of iteration is not fixed and you must have to execute the loop a least once, it is recomended to use do-while loop.

Q. Why do-while loop is called as exit control loop?
Java do-while loop is called as exit control loop because it checks the condition at the end of loop body. It executes at least once because condition is checked after loop body.

Syntax -

```
initialization;
do {
    statement;
    expression;
  }
while (condition);
```

i) Condition is an expression which is tested. If the condition is true, the loop body is executed and control goes to update expression. As soon as the condition becomes false, loop breaks automatically.
Example:

```
i <= 100
```

ii) Expression is either incremented or decremented while every time the loop body is executed.
Example:

```
i++;
```

Example -

```
public class DowhileExample {
    public static void main (String[] args) {
        int i=1;
        do {
            System.out.println (i);
            i++;
        }
        while (i<=10);
    }
}
```

Output -

```
1
2
3
4
5
6
7
8
9
10
```

Q. How do-while enters into infinite loop?

```
public class Example {
    public static void main (String[] args) {
        do {
            System.out.println ("Infinite loop");
        }
        while (true);
    }
}
```

If we pass true in the do-while loop, it will be infinitive do-while loop.

b) while loop : It is used to iterate a part of the program repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops. If the number of iteration is not fixed, it is recomended to use the while loop.

Syntax -

```
initialization;
while (condition){
    Statement;
    expression;
    }
```

Example -

```
public class WhileExample {
    public static void main (String[] args){
        int i=1;
        while (i<=10){
            System.out.println(i);
            i++;
        }
    }
}
```

Output -

```
1
2
3
4
5
6
7
8
9
10
```

Q. How while loop enters into infinite loop?

```
public class Example {
    public static void main (String [] args) {
        while (true) {
            System.out.println ("Infinite loop");
        }
    }
}
```

If we pass true in the while loop, it will be infinitive while loop.

c) for loop : It is used to iterate a part of the program several times. If the number of iteration is fixed, it is recomended to use for loop. There are three types of for loop -

i) Simple for loop
ii) for-each loop
iii) labeled for loop.

i) Simple for loop - This loop is similar as C/C++. We can initialize the variable, check condition and increment/decrement value.

Syntax -

```
for (initialization ; condition ; increment/decrement){
        Statement ;
}
```

Initialization is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable.

It is an optional condition.

condition is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.

Increment / Decrement is used to update the value of a variable. It is an optional condition.

statement is executed only when the condition of the loop is true. If the condition is false the loop terminates automatically without executing the statement.

Example -

```
public class ForExample {
    public static void main (String[] args) {
        for (int i=1; i<=10; i++) {
            System.out.println (i);
        }
    }
}
```

output - 1
2
3
4
5
6
7
8
9
10

ii) Nested for loop : If we have a for loop inside the another loop, is known as nested for loop. The inner loop executes completely whenever outer loop executes.

Example -

```
public class NestedForExample1
    public static void main (String[] args){
        for(int i=1; i<=3; i++) {
            for (int j=1; j<=3; j++) {
                System.out.println(i+" "+j);
            }
        }
    }
}
```

output -

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

iii) for-each loop : This loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation. It works on the basis of elements and not index. It returns element one by one in the defined variable.

Syntax-

```
for ( data_type variable : array_name) {
        statement;
    }
```

Example-

```
public class ForEachExample {
    public static void main (String[] args) {
        int arr[] = {12, 23, 44, 56, 78};
        for (int i : arr) {
            System.out.println(i);
        }
    }
}
```

output- 12
23
44
56
78

iv) Java Labeled for loop : We can have a name of each Java for loop. To do so, we use label before the for loop. It is useful while using the nested for loop as we can break / continue specific for loop.

Syntax-

```
labelname :
    for (initialization ; condition ; inc/dec){
        statement;
    }
```

Example -

```
public class Example {
    public static void main (String [] args) {
        aa:
        for (int i=1; i<=3; i++) {
            bb:
            for (int j=1; j<=3; j++) {
                if (i==2 && j==2) {
                    break aa;
                }
                System.out.println (i+" "+j);
            }
        }
    }
```

output -

```
1 1
1 2
1 3
2 1
```

If we use break bb, it will break inner loop only which is the default behaviour of any loop.

Example -

```
public class Example2 {
    public static void main (String [] args) {
        aa:
        for (int i=1; i<=3; i++) {
            bb:
            for (int j=1; j<=3; j++) {
                if (i==2 && j==2) {
                    break bb;
                }
```

```
                System.out.println(i+" "+j);
            }
        }
    }
}
```

output -
```
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

Q. How for loop enter into infinite loop?

```
public class ForExample{
    public static void main(String[] args){
        for(;;){
            System.out.println("Infinite loop");
        }
    }
}
```

If we use two semicolons ;; in the for loop, it will be infinitive for loop.

Jump Statements - It is used to transfer the control of the program to the specific statements. There are two types of jump statements -

a) break
b) continue

a) break : When this statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement.

In Java, the break is majorly used for -
i) Terminate a sequence in a switch statement
ii) To exit a loop
iii) Used as a "civilized" form of goto.

Example -

```
public class Example1 {
    public static void main (String[] args) {
        for (int i=1; i<=10; i++) {
            if (i==5) {
                break;
            }
            System.out.println(i);
        }
    }
}
```

Output - 1
2
3
4

Example -

```
public class Example2 {
    public static void main (String[] args) {
        for (int i=1; i<=3; i++) {
            for (int j=1; j<=3; j++) {
                if (i==2 && j==2) {
                    break;
                }
```

```
            System.out.println(i+" "+j);
        }
    }
}
```

Output -

```
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

b) Continue : This statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.

Example -

```
public class Example1 {
    public static void main (String[] args){
        for (int i = 1 ; i<=10; i++) {
            if (i==5) {
                continue;
            }
            System.out.println(i);
        }
    }
}
```

Output :

1
2
3
4
6
7
8
9
10

Example2-

```
public class Example2{
    public static void main(String[] args){
        for(int i=1; i<=3; i++){
            for(int j=1; j<=3; j++){
                if(i==2 && j==2){
                    continue;
                }
                System.out.println(i+" "+j);
            }
        }
    }
}
```

output -

1 1
1 2
1 3
2 1
2 3
3 1
3 2
3 3

Example -

```
public class Example3 {
    public static void main (String[] args) {
        aa:
            for (int i=1; i<=3; i++) {
                bb:
                    for (int j=1; j<=3; j++) {
                        if (i==2 && j==2) {
                            continue aa;
                        }
                        System.out.println(i+" "+j);
                    }
            }
    }
}
```

Output -

```
1  1
1  2
1  3
2  1
3  1
3  2
3  3
```

Example -

```
public class Example4 {
    public static void main (String[] args) {
        int i=1;
        while (i<=10) {
            if (i==5) {
                i++;
                continue;
            }
```

```
            System.out.println(i);
            i++;
        }
    }
}
```

output -

1
2
3
4
6
7
8
9
10

Java comments - These are the statements in a program that are not executed by the compiler and interpreter.

These are used -

i) To make the program more readable by adding the details of the code.

ii) To ~~make~~ maintain the code and to find the errors easily.

iii) To provide information or explanation about the variable, method, class or any statement.

iv) To prevent the execution of program code while testing the alternative code.

Types of Java comments -

a) Single Line comment

b) Multi Line comment

c) Documentation comment

a) Single Line comment – It is used to comment only one line of the code. It is the easiest way of commenting the statements.
It starts with two forward slashes (//).

Example –

```
public class Example{
    public static void main (String[] args){
        int i = 10; // i is a variable with value 10
        System.out.println(i); // printing the variable i
        }
    }
```

output –

10

b) Multi Line comment – It is used to comment multiple lines of code. It is used to explain a complex code snippet or to comment multiple lines of code at a time. Multi-line comments are placed between /* and */.

Example –

```
public class Example {
    public static void main (String[] args){
        /* Let's declare and print
           variable in Java.*/
        int i = 10;
        System.out.println(i);
        /* float j = 5.9;
           float k = 4.4;
           System.out.println(j+k); */
        }
    }
```

Output -
10

c) Documentation Comment - This is usually used to write large programs for a project or software application as it helps to create documentation API.
To create documentation API, we need to use the javadoc tool. The documentation comments are placed between /** and */.

Syntax -

```
/**
*we can use various tags to depict the parameter
*or heading or author name
*We can use HTML tags
*
*/
```

javadoc tags -

1. { @docRoot } - used to depict relative path to root directory of generated document from any page.
2. @author - To add author of the class.
3. @code - To show the text in code font without interpreting it as html markup or nested javadoc tag.
4. @version - To specify "version" subheading and version-text when -version option is used.
5. @since - To add 'since' heading with since text to

generated documentation.

6. @param - To add a parameter with given name and description to "parameters" section.

7. @return - Required for every method that returns something.

Example -

```
import java.io.*;
/**
 * <h2> Calculation of number </h2>
 * This program implements an application
 * to perform operation such as addition of numbers
 * and print the result.
 * <p>
 * <b> Note: </b> Comments make the code readable
 * and easy to understand.
 *
 * @author Aman
 * @version 17.0
 * @since 2023-01-22
 */
public class Calculate {
/**
 * This method calculates the sum of two integers
 * @param input 1 This is the first parameter to
 * Sum() method.
 * @param input 2 This is the second parameter to
 * the sum() method
 * @return int this returns the addition of input 1
 * and input 2.
 */
```

```
public int sum (int input1, int input2){
        return input1+input2;
    }
/**
* This is the main method uses of sum() method.
* @param args unused
* @see IOException
*/
public static void main (String[] args){
    Calculate c = new Calculate();
    int res = c.sum(40,20);
    System.out.println("Addition of numbers:"+res);
    }
}
```

Q. Are Java Comments executable?

Java ~~codes~~ comments are not executed by compiler or interpreter. However before the lexical transformation of code in compiler, contents the code are encoded into ASCII in order to make the processing easy.

Example -

```
public class Demo{
    public static void main (String[] args){
        // the below comment will be executed
        // uuuuu System.out.println("Java comment");
    }
}
```

Output -

```
Java comment
```

The above code generated the output because the compiler

parse the unicode character \u000d as a newline before the lexical transformation, and thus the code is transformed as below :-

```
public class Demo{
    public static void main (String[] args){
        // the below comment will be executed
        //
        System.out.println (" Java comment ");
    }
}
```

Thus, the unicode character shifts the print statement to next line and it is executed as a normal java code.

Strings in Java

Q. What is String?

String is a sequence of characters. But in Java, String is an object that represents a sequence of characters enclosed inside double quote (" "). String class is present in java.lang package. Strings are generally immutable in nature.

Syntax -

a) By String literal

```
String variable name = "value";
```

b) By new keyword

```
String variable_name = new String ("value");
```