OBJECT ORIENTED METHODOLOGY

COMPUTER SCIENCE & ENGINEERING

Programming Language :-

Language :-

→ Language is a mode of comm^n i.e used to share ideas, opinions with each other.

→ eg :- If we want to teach someone, we need a language i.e understandable by both communicators.

What is Programing Language ?-

→ A PL is a computer language i.e used by programmers to communicate with computers.

eg :, C, C++, Java, Python etc.

Types of PL :-

a) Low Level Language
b) High Level language
c) Middle Level language

a) Low Level language :-

Low LLL is machine dependent PL.

→ The Processor run low-level language directly without need of compiler or interpreter. (So it runs so fast.)

(2)

* i) ML (Machine Language) :-
  ↳ machine code or object code.
  → easier to read because it ~~is~~ ~~difficult to read~~ is normally displayed in binary or hexadecimal form.

* ii) Assembly Language :-
  Set of instructions in a symbolic & human-understandable form.
  → Assembler to convert a AL to ML.
  → Less memory less execution time to execute program eg :- mov 1
  add 1

b) HLL :→
  Designed for developing user-friendly S/w program & website.
  → Require compiler or interpreter to translate the program into ML.
  eg : Java, C, C++, C#, etc

  i) Procedural oriented PL :-
  ↳ ~~Derived form~~ procedure call concept
  (It divides a program into small procedures called routines or fun^n).
  eg :- IDE, microsoft visual studio.

C, FORTRAN, Basic.

ii) OOP

↳ Based upon the objects.

→ Programs are divided into small parts called objects

eg: Java, Python

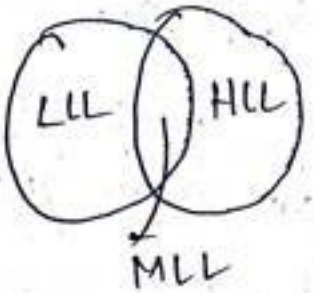
iii) Natural language :—

Part of human languages such as English, Russian, German etc.

c) MLL :→

lies bet^n LLL & HLL

, C, C++.



Description of compiler & Interpreter

compiler :→ computer program converts the HLL program written to MLL

→ It translates the whole program at a time.

eg Java, C.

(A) Interpreter :→

converts instructions line by line written in HLL to MLL

→ It translates the program line one by one.

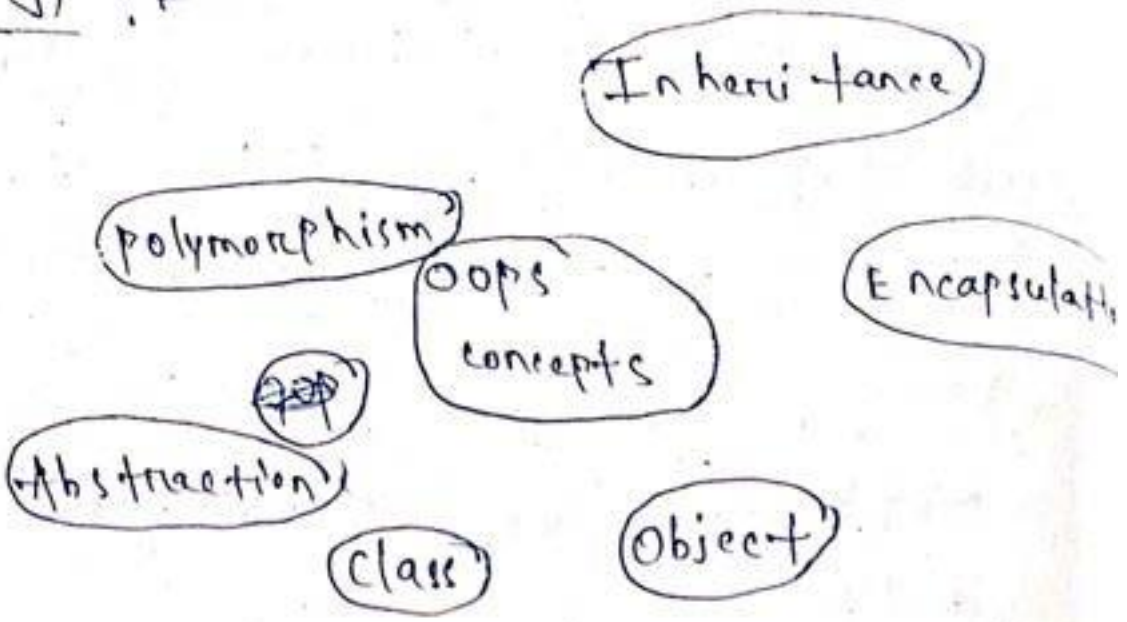
* Difference bet^n compiler & Interpreter

compiler	Interpreter
→ Translate the entire program at a time	→ Translate program line by line.
→ more memory required	→ less memory
→ execution time is less.	→ execution time is more
→ If error detected nothing is executed	→ If error detected it stops & do not go to next line.
→ C, C++ etc	→ Perl, Python etc.

OOP

→ It is a methodology or paradigm to design a program using class & object.

Terminology :-



(Object oriented programming)

* Difference bet' procedural & object op

OOPs concepts & Terminology 20.10.21

→ OOP is a programming paradigm (prototype/model) that relies on the concept of classes & objects.

→ It is used to structure a software program into a simple, reusable piece of code.

imp * Features / characteristics of OOPs :→

There are main OOP concepts in Java, These are :-

a) Object
b) class
c) Inheritance
d) Polymorphism
e) Abstraction
f) encapsulation

a) Object:→

→ Objects are the basic run-time entities that has state & behaviour) in an object oriented system.

→ An object can be defined as an instance of cls.

eg:→ Dogs have state(name, color, breed, hungry) & behavior (barking, fetching, wagging tail).

b) Class:-

→ collection of objects is called class.

→ It is a logical entity

eg:→ Each bicycle was built from the same set of blueprint & therefore contains the same component

c) Inheritance:-

→ When one object acquires all the Properties & behaviours of Parent-object. It is known as inheritance.

→ It Provides code reusability. It is used to achieve run-time polymorphism.

→ eg:→ Dog, cat, cow can be derived class of Animal base class.

Some imp. terminology:-

i) Sub class:- The class that inherits properties from another class.

ii) Super class:- The class whose Properties are inherited by sub class.

iii) Reusability :—

When we want to create a new cls & there is already a clss that includes some of the code that we want, we can derive our new class from the existing cls. (By reusing the fields & methods of the existing cls).

d) Polymorphism :— (More than one form)

→ When one task is performed by different ways i.e known as polymorphism.

eg :→ To convience the customer differently, to draw something eg. shape, or rectangle etc

→ In Java we use Method overloading & method overriding to achieve polymorphism.

* Dynamic Binding :—

When the type of the object is determined at run-time

e) Abstraction :→

→ Hide internal details & showing functionality is known as abstraction.

eg :— Phone call (we don't known the internal processing)

→ In java, we use abstract class & interface to achieve abstraction.

f) Encapsulation :—

Binding (or wrapping) code & data together

into a single unit

→ eg :→ A capsule is wrapped with different medicine

→ Java Bean (All the data members are private here).

→ provides security that keeps data & method safe from inadvertent change.

→ (Hiding data & method with in the object)

Benifits of oops :→

oop has great advantages over other programing styles.

a) We can eliminate redundant code & extend the used of existing class. (Through inheritance).

b) Helps the programmers to build secure program (Principle of data hiding).

c) Possible to have multiple objects to coexist without any interference.

d) The object controls how one interacts with others, Preventing others kinds of errors.

e) Appl^n

1) client - Server Systems (internet

2) Object oriented db

3) Real time System design

4) Simulation & modeling System

What is Java ?

→ ~~James gosling 1995.~~

Java was developed by James Gosling in 1995

→ A general Purpose object-oriented language.

→ Write once Read Anywhere (WORA).

→ Designed for easy web / Internet appl^n.

→ platform independent PL.

→ Java contains extensive library (2500)

* Why platform independent ?

~~Becoz of~~ WORA structure makes Java as platform independent i.e write codes in one or for example windows xp & executer that code in another platform (os) i.e MACOS & Linux.

Why Java

→ Simple

→ object oriented

→ Distributed

→ Interpreted

→ Robust

→ Secure

→ portable

→ multithreaded

History Behind Java :→

→ ~~James Gosling~~ James Gosling is the father of Java. co-founder of Java was Vinod Khosla.

→ Java was introduced in "Oak", may 20, 1995 at sun world.

Java was catagories into 3 basic types

a) ~~Jar~~ J2SE (Java 2 standard edition) :—

⇒ To develop client side standalone appl^n such as Thread, File IO, Swing, Applet.

b) J2ME (Java 2 Micro eddition) :—

To develop appl^n for mobile devices such as cell phones.

c) J2EE (Java 2 enterprise edition) :—

To develop serverside appl^n such as Java Servelet & Java Server pag

→ Java was originally generated by sun microsystem, But now Java was purchased/ owned by oracle corporation.

How is Java different from c

⇒ Maion difference is that c is a structure oriented & Java is an object oriented (class & object

1st, 2nd, 3rd ...

→ Java doesn't support an explicit pointer.

→ Java doesn't have preprocessor, so, we can't use #define,

→ Java doesn't include structures unions & enum data types.

→ Difference bet^n c, c++ & Java

	C	C++	Java
1)	Procedural lang.	→ object oriented PL	→ pure OOPL
2)	Static PL	→ Static PL	→ Dynamic PL (operation done on runtime also).
3)	code is executed directly	→ directly	→ code is executed by JVM
4)	Platform dependent	→ platform dependent	→ Platform independent
5)	uses only compiler	→ compiler	→ Both compiler & interpreter (Interpreted language).
6)	generate .exe & .bax, files	→ .exe	→ .class file
7)	32 keywords	→ 60	→ 52
8)	.c	→ .cpp	→ .java
9)	Support pointer	→ also	→ doesn't

→ Su doesn't support constructor & destructore	→ Both	→ only constructo r
→ malloc(), calloc(), free(), realloc().	→ new & delete operatore	→ garbage collector.
→ An array should be declared with size a[10];	→ Same	→ An array can be declared without declaring the size a[]

~~Execution Model of Java~~

Prerequisite B4 start writing Java program

1) Setup your Work Environment.
2) Any sorce of Notepad- to write java progra
3) Make Sure you have The JDK installed

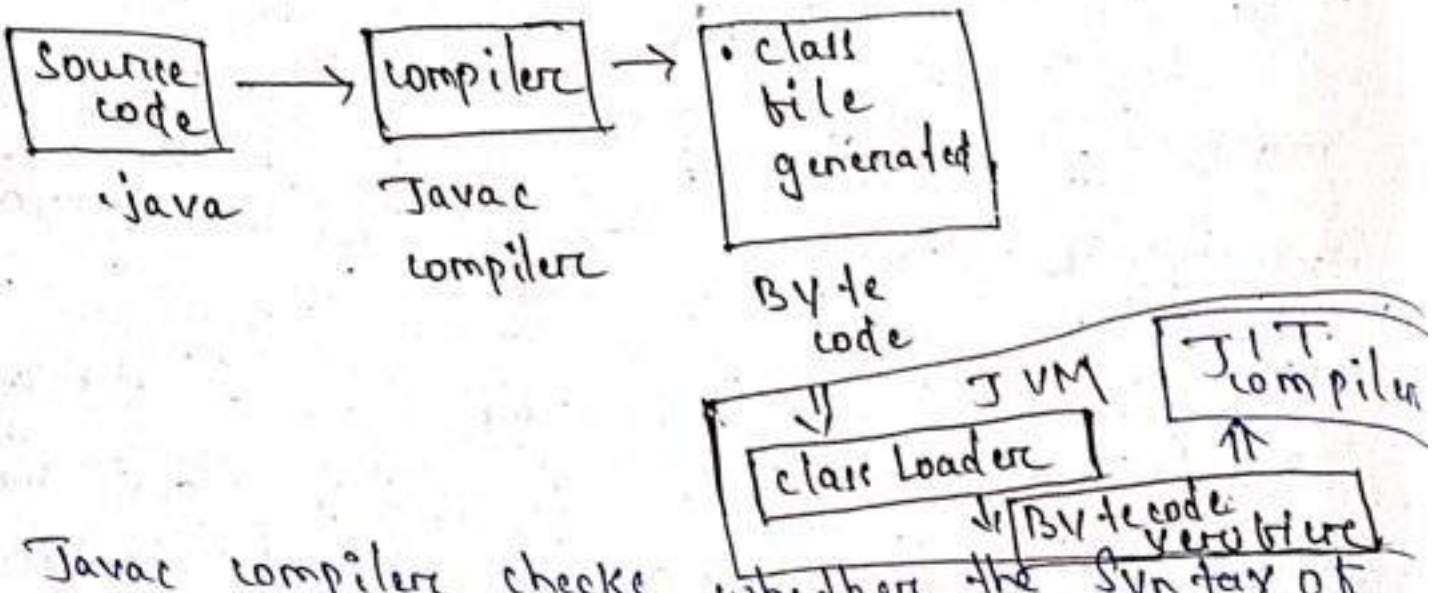
Tools useful for developing & testing programs written in Java & running on the java platform.

Execution model of Java

2 steps

a) compilation → javac compiler

b) Execution → JVM



→ Javac compiler checks whether the syntax of the Program is correct or not.

↳ If it successfull we get .class file (Byte code)

Byte code :⇒ Machine independent code
↳ not readable (we can run the code in any OS)

Run :-

→ Gives the .class file to Java class loader
Javac <name of mainmethod>

→ class loader loads all the classes which is required for complete info of java Program

→ Byte code verifier (Run time errors)

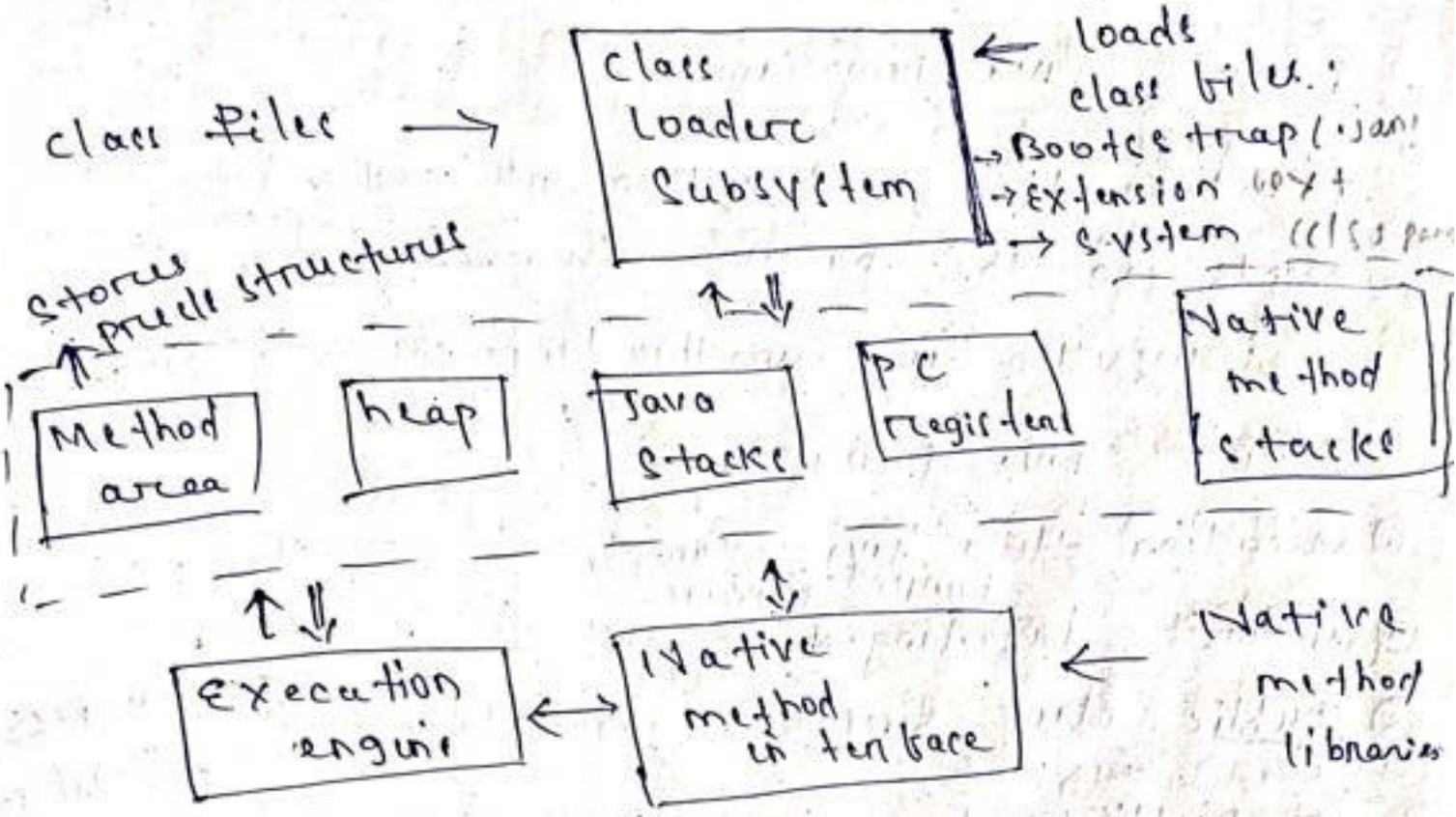
→ JIT (Just in time) compiler converts byte code to machine code.

JVM (Java Virtual Machine)

→ JVM converts Java Byte code into machine lang.

→ Architecture of JVM (Java Virtual Machine)

Architecture of JVM contains classloader, memory area, execution engine etc.



Class/Method Area :—

Store all Preclass structures.

eg :- runtime const. pool

Heap stores the object

Java stacks :— Methods or variables

Program counter register :→

It stores the address of the currently executing instruction by JVM

Native Method stacks — non java code (C or C++)

Native methods (JNI)

Java m() ↔ C++ m() { } → Bridge (JNI)

Execution Engine :-
virtual processor, interpreter, compiler

Native method interface ↔ Java native libr

First Java program :⇒

store/holds the instruction of native code

depends on the native library.

→ C++ is written in another lang. instead of java.

First Java program

① creating Hello.java example —

②① import java.lang.* ; → ① project → package → All cls

② public class Simple /* Hello */

{

③ public static void main(String args[]) {

System.out.println("Hello");

}

}

① Package Declaration : ⇒

① package · package name

Built in — built in or user defined

② import Statement

import built in / user defined library

3> class declaration

public class → any package can be access

Main Method

→ class :→ Key word is used to declare a class in Java

public :→ Access Method i.e usable to all.

static :→ There is no need to create object to create object to invoke the static method.

→ The main method is executed by JVM, so it doesn't required to create object to invoke main()

void → Return type of the method, means it doesn't return any value.

main :— Startup of the program

a) String args[] :→ connect line argument.

c) System.out.println :— print statement

D-t - 25.10.21

Variable → name of memory location whole value varies, that may or may not change

Datatype :→ specifies to the compiler which kind of it is.

'+' is used to join 2 string / concatenate

use 3rd

Data type :⇒

a) Premitive datatype

b) object / reference datatype. / Non premitive

a) Premitive Data type

⇒ Predefined by the programing language called as the Keyword brough by the compiler.

→ There are 8 Premetive datatype

i) Byte :⇒

→ used to save space, may be inplace of integer becoz a byte is 4 time smaller than an integer.

→ 8 bit signed (may be -ve or +ve)

→ -128 to 127 eg :- byte a = 10;

ii) Short

→ 16 bit signed 2s complement integer

→ -32768 to +32767 eg Short a = 324;
Short n = -5000;

→ Save space (2 times smaller than integer)

iii) Integer

→ 32 bit signed 2's complement integer

→ -2147483648 to 2147483

→ Default value is zero → eg int a = 30;

iv) Long

64 bit signed 2c complement integere

→ -9223372036854775808 to 9223372036854775807

→ used in wider range than integere

→ Default value is 0L

→ ex:- 95678901 eg long m = 10000L;

v) Float

→ It is a single prusigion, 32 bit IEEE 754 floating point.

→ Float mainly used to save memory in large array of floating point no.

→ default value is 0.0f

→ Never used for prusized value such as currency

→ eg x = 259.5f Float F1 = 234.5f;

vi) Double

→ Double prusigion IEEE 754 floating point

→ 0.0D double a = 12.3;

vii) Boolean :→

→ one bit information datatype used for logical aspects.

→ 2 value True/False boolean a = True;

→ Default value is false

eg boolean x = True

viii) char

→ Single 16 bit unicode char.
→ \u0000 to \uffff char letter='A';
→ 'char' is used to store any character.

WAP to add two no.

```
class Add
{
  public static void main(String args[])
  {
    int a=10, int b=20;
    int c = a+b;
    System.out.println("Sum is = "+c);
  }
}
```

WAP to Print a character / string

```
Public class A
{
  public static void main(String args[])
  {
    char a1 = 'a';
    char a2 = 'b';
    System.out.println("char 1:" + a1);
    System.out.println("char 2:" +a2);
  }
}
```

b) Non Primitive data type

c) user defined data type

4 types of primitive data type

i) Array :-

→ used to store data in consecutive manner

ii) String :-

→ used to store consecutive characters.

String abc = "Hello";

iii) class

iv) Interface

Variable : →

A name of the reserved area allocated in memory. (Name of the memory location)

in → data = 50;

↳ variable

3 types of variable

a) Local variable → (inside the body of the method)

b) Instance variable / Non-static variable ↳ inside the class but outside the body of the method)

c) static variable

↓

declared as static

```
Public class A
{
    Static int m = 100; // → Static Variable
    void method()
    {
        int n = 90; // local Variable
        int m = 20; int s = m+n;
    }
    public Static void main(String args[]
    {
        int data = 50; // instance Var
        int int b = m+data;
    }
}
```

Operator

Symbol i.e used to perform operation

- Unary operator
- Arithmatic operator
- Shift operator
- Relational operator
- Bitwise operator
- Logical operator
- Ternary operator
- Assignment operator

a) unary operator

→ Java unary operators requires only one operand.

operation :→

a) postfix (exp++ / exp--)

b) prefix (++exp / exp--)

eg :→ int a = 10;

(++a) ; ⇒ 11

(a++) ⇒ 11 (12)

(--a) → 11

(a--) → 11 (10)

a++ = a+1

a++

a--

Pr WAP for unary operator

a = 10 (11)

a++ ⇒ 10

a++ (11)

```
class unary
{
    public static void main (String args[])
    {
        int a = 10;
        System.out.println(a++); → 10 (11)
        System.out.println(++a); → 12
                          (a--); → 12 (11)
                          (--a); → 10
```

b) Arithmatic operator:-

→ It is used to perform addition, sub, multiplication & division (Basic mathematical operations)

+, -, *, /, %.

⇒ Java Arithmatic operation Example

```
public class operators
{
  public static void main (String args[])
  {
    int a = 10, b = 20;
    System.out.println(a+b);
    System.out.println(a-b);
    System.out.println(a*b);
    System.out.println(a/b);
    System.out.println(a%b);
  }
}
```

20 / 20 / 10 / 200 / 2 / 0 ← Reminder

BY using Scanner cls :⇒ ← import Scanner cls

```
import java.util.Scanner;
```

↳ collection of framework, legacy collection class, event model, date & time facilities, string tokenizer, Random no. generator.

```
public class A
{
  p svm (String args[])
  {
    int a, int b, int c;
    Scanner sc = new Scanner (System.in);
    System.out.println("ENTER 1st no.");
    a = sc.nextInt();
    System.out.println("ENTER 2nd no.");
    b = sc.nextInt();
```

← construct Scanner cls object

← define variable to Receive i/p

```
c = a + b;
System.out.println(" " c);
}
}
```

for ~~double~~

nextDouble() → for double
nextFloat() → float
nextLine() → String
nextLong() → long

nextBoolean() ← boolean
nextShort() → short

~~Q:~~ WAP to print a person's Age, Name & salary by using Scanner clc.

c) Logical operator :—

When we want to form compound by combining two or more Relation.

→ &&, ||, !.

→ Logical expression defer a value of true or false.

op1	op2	op1 && op2
T	T	T
T	F	F
F	T	F
F	F	F

op1	op2	op1 || op2
T	T	T
T	F	T
F	T	T
F	F	F

d) Relational operator

→ It is used in the case of comparision

3) else : 3rd))

==, !=, >, <, <=, >=

↳ True / false

```
int a = 5, b = 6;
System.out.println( a > b);
```

Logical OR / || & operations

```
int a = 10, b = 6, c = 15;
System.out.println( a > b && a > c);
```

e) Shift operators :-

<<, >> → Right shift

↳ left shift

```
public class Shift {
    public static void main(String args[])
    {
        int a = 8; → 1000
        int b = a << 2; ⇒ extra '0's
        int b = a >> 2 ⇒ cut the Right
    }
}
```

8 ⇒ 1 0 0 0 (32 16 8 4 2 1) ⇒ 32

1 0 0 0 0 0 → 32

↳ add extra 2 '0's

8 ⇒ 1 0 0 0

>> 2 ⇒ 10.

25

⇒ a = 1 1 0 0 1

a << 2 ⇒ 1 1 0 0 1 0 0 (add 2 0's)

>> 2 ⇒ 1 1 0 (cut 0)

#) Bitwise operators
→ Execute single bit of theire operands.

|	OR (Bitwise)
&	Bitwise AND
&=	Bit wise AND Assignment
|=	Bit wise OR "
^	→ XOR (Exculsive OR)
<<	→ left shift
>>	→ Right shift
~	→ one's complement

int a = 5, int b = 7

5 → 0 1 0 1 → a

7 → 0 1 1 1 → b

a	b	&	|	^
0	0	0	0	0
0	1	1	1	1
1	1	0	1	1
0	1	0	1	1
1	1	1	1	0
		5	7	2

~ → single

~a

0 → 1
1 → 0
0 → 1

1 0 1 0 → b (-ve)

XOR

0 - 0 = 0
1 - 1 = 0

0 1 0 1
+ 1
0 1 1 0

```
int a = 5, b = 7;
System.out.println( a & b);
                  ( a | b);
                  ( a ^ b);
                  (~a);
```

1 0 0 0 0 → 32
32 16 8 4 2 1
1 0 → 2

2) else if 3rd))

g) Ternary operator

conditional operator → if false
→ if true

var=(cond^n) ? value 1 : value 2

a = 2
b = 5

min=(a<b) ? a : b ;

min=2 < 5 : 2

h) Assignment operator :=

Assign value on the right to the operand on the left

a = 10 ; b = 20

a += 4 ; a = a + 4

b -= 4 ; b = b - 4

a + = 4
a = a + 4;

a - = 4
a = a - 4;

Type casting : Type conversion

→ converting one premitive datatype into another premitive data type is called premitive ~~data type~~ casting / conversion.

int a = 5;

double b = 2.7;

int c = a + b; 5 + 2

double c = 5.0 + 2.7

~~WAP to calculate division~~
WAP to find max^m of 3 no. using nested if else statement

```
class Test
{
  Psvm( String args[])
  {
    int a=6, b=5, c=13;
    if (a>b) && (a>c)
    { System.out.println(" a is larg
    }
   else if(b>a) && (b>c)
    { System.out.println(" b is la
    }
  else
    { system.out.println(" c is grea
    }
} }
```

WAP to calculate division obtained by a student with his % mark given

```
a = 76;
if (a>60);
  { 1st;
else if(a>=50 &&
        a<60)
  { 2nd
} else 3rd }
```

9/11/21

Switch Statement :-

The Switch " is Java's multi way branch statement which
→ It is a case/selection oriented statement which we have to choose a
among the alternative.
→ It is better than nested if else and if-else single alternative.
-if ladder statement.

Syntax

```
switch(choice/expression)   ← user define variable or const.
                  → Built in or mandatory
{
    case 1 :
      S1 sequence;
      break;  ← may or may not be
    case 2 :
      // statement sequence;
      break;
       :
       :
    default :
       S 1 ;
       : Sn
}
```

```
int number = 20;
Switch (number)
{
  case 10:
     System.out.println("10");
     break;
  case 20 :
                    ("20");
  break;
  case 30:
                    ("30");
  break;
```

```
default :
    C.O.P("Not 10, 20 or 30");
}
```

Note :→ i) There can be one or N no. of case value must be literal.

ii) The case value must be switch exp. type only. (it doesn't allow variable

iii) The case value must be unique. In case of duplicate, it renders compile time errors.

iv) Each case statement can have a break statement which is optional. when control reach to break it ~~brea~~ jumps the control after switch.

Jump Statement :—

Jump Statement makes the control jump to another section of the program.

i) Break Statement :→ Takes control out of the loop.

ii) continue Statement → Takes control the beginning of the loop.

iii) go to Statement → To desired line of program

iv) Return

Class :⇒

⇒ class is a rule or structure using that structure we give real life existance / implementation.

⇒ class is a template or blueprint from which objects are created.

→ Every thing we wish to represent in a Java program must be encapsulated in cls.

⇒ cls defines the state & behaviour of the basic programing components known as objects.

⇒ clses creates objects & object use methods to communicate bet^ them. (Talk about OOP)

eg :⇒

class	properties	Behaviour
Human	Height	eat
	Weight	talk
	color	sleep
	Name	

Anatomy / Syntax of cls :⇒

```
modifier class classname
{
    Access level   datatype variablename;
                   datatype methodname(p1, p2)
                   {
                   ≡
                   }
```

2

Basic factorial of no.

```
class Fact
{ p s v m ( String{ args[] )
  { int n=5, fact=1,P;
    while (n >= 1)
    { fact = fact *n;
      n--;
    }
  System.out.println(fact);
  }
```

Static block, Method, variable

variable → Static ~~pri~~

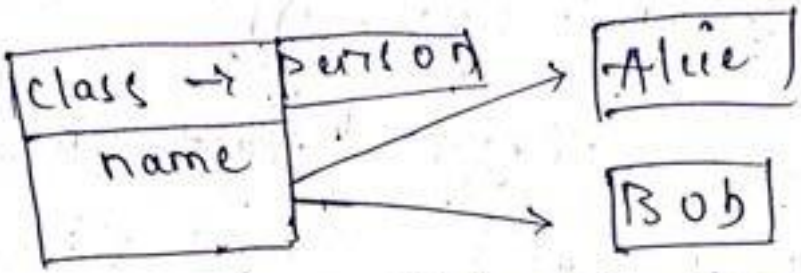
↓ 2 ways

i) we can directly access

ii) with the help of cls nam

Static

Related to cls



→ indicates that the particulare member belongs to a type itself. mainly used to help memory management.

1 copy only (not change in cls.)

eg. name — No value changes in Stati
breed: Homo

applied in variable, cls, method, blocks, nested cls

Final

↳ static keywords can't be declared in non static cls.

* We can write static in main cls

→ We can't ~~not~~ write ~~non~~ static method in static method.

Jump Statement :-

↳ Transfere Program execution control to a specific statement.

a) Break :⇒

```
class A
{
  psvm(...)
  {
    for(int i=0; i<10; i++)
    {
      if(i==5)
      {
        break;
      }
      s.o.p(i);
    }
    s.o.p("outside");
  }
}
```

0 1 2 3 4

b) continue :→ It is used within looping statements.

```
class A
{
  psvm(String args[])
  {
    for(i=1; i<10; i++)
    {
      if(i%2==0)
      {
        continue;
      }
      s.o.p(i);
  } } }
```

1 3 5 7 ... 99

Return

It is mainly used in methods in order to terminate a method in between & return back to the callee method

```
class A
{
    double add(double a, double b)
    {
        double sum = 0;
        sum = (a+b) / 2.0;
        return sum;
    }
```

Syntax :→

```
return returnvalue;
```

program :→

```
public class A
{
    int compare()
    {
        int x = 3;
        int y = 8;
        S.o.p("x= "+x +"\ny= "+y);
        if(x>y)
            return x;
        else return y;
    }
    psvm()
    {
        A ob = new A();
        int R = ob.compare();
        S.o.p(+R);
    }
}
```

Loops in Java

→ Looping repeatedly execute the same set of instruction until a termination cond^n is met.

→ It reduces the programming effort bcz execution of similar statement bcz a no. of time or infinitely time

(or) If we want to repeat one statement multiple times, in that case we can simply use loop / iteration statement.

→ Loops must have a starting point it may or may not have ending point.

Types

a) while loop :

→ It is used to iterate a part of the program several times.

→ If the no. of iteration is not fixed, it is recommended to use while loop.

Syntax :⇒

```
while (condition)
{
    code to be executed ;
    increment / Decrement (may or may not)
}
```

Q Write to print 1 to 10

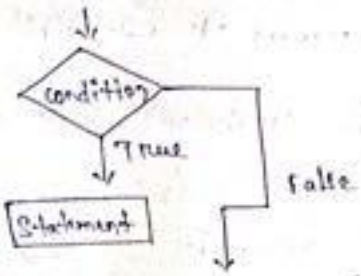
Ans:

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

Infinite loop using while

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

Flowchart



b) do-while Loop

→ It is used to iterate a part of the program several times.

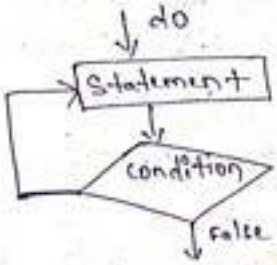
→ If the no. of iteration is not fixed & you have to execute the loop at least once, it is recommended to use do-while loop.

→ Java do-while loop is executed atleast once before cond^n is checked after loop body.

Syntax :-

```
do
{
  code to be executed;
  increament / Decrement
} while (condition)
```

Flowchart :-



Q WAP to print 1 to 10 using do-while

```
public class A
{
  Public static void main (String args[])
  {
    int i;
```

```
do
{
  System.out.println(i);
  i++;
}while(i<=10)
}
}
```

Java infinitive loop

```
do
{
  System.out.println("infinite loop");
}while(true)
```

c) For loop

→ The java for loop is used to iterate a part of the program several times.

→ If the no. of iteration is fixed, it is recommended to use for loop.

→ It consist of 3 parts

i) Initialization :

→ It is the initial cond^n, which is executed once when the loop starts

→ Here, we can used an already initialized variable or we can initialize a variable.

ii) condition :

→ It continues execution until the condition is false.

→ It must returned boolean value i.e either true / false.

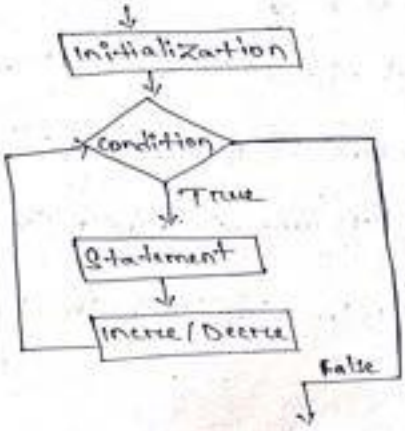
iii) Increament / Decrement :

→ It increments or decrements the value.

Syntax

```
for(initialization; condition; incre/decre)
{
  statement to be executed;
}
```

Flowchart



Q. WAP to print 1 to 10 using

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

Pyramid

```
*
**
***
****
*****
```

```
for(int i=1;i<=5;i++) ← ROW
{
  for(int j=1; j<=i; j++) ← column
  {
    System.out.println("*"); —
  }
  System.out.println(); ← newline
}
```

```
*****
****
***
**
*
```

```
for(int i=1; i<=5; i++)
{ for(j=5; j>=i; j--)
  {
    System.out.println("*");
  }
  System.out.println();
}
```

Java infinite loop

```
for(;;)
{
 System.out.println("infinite loop");
}
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
for(i=1;i<=5;i++)
{ for(j=1; j>=i; j++
  {
    S.O.P(j);
  }
  S.O.P();
}
```

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
for(i=1; i<=5; i++)
{
  for(i=1; j<=i; j++)
  {
    S.o.p(i);
  }
  S.o.p();
}
```

Anatomy / Syntax of class

```
<access specifiers> class class-name
    (can be public/default)
{
    datatype variablename;
    returntype methodname(P1,P2
}
```

Arrays

Class

→ class is a Rule or Structure using that structure we give real life existance/implementation.

→ class is a template or blueprint from which objects are created.

→ Everything we wish to represent in a Java program must be encapsulated in class

→ class defines the state & behaviour of the basic programming components known as objects.

→ classes creates objects & object use methods to communicate with them.

	Class	Properties / State	Behaviour
	Human	Height weight colour	eat talk sleep

Arrays in Java

An array is a collection of similare type of elements which has contiguous memory location.

→ Array in Java is index-based. The first element of the array is stored at the 0th index, and element is stored in 1st index and so on.

0	1	2	3	4	5
25	56	79	53	45	10

int [] A

Advantages

i) code optimization :
it makes the code optimized, we can retrieve or sored the data efficiently.

ii) Random Access :
We can get any data located at an index position

Disadvantage

Size limit :- we can store only the fixed size of elements in the array.

→ It doesnot grow its size at runtime.

Syntax to write array

1) datatype[] arrayname;

eg int[] a;

2) arrayname = new datatype[size];

eg regdno = new int[5];

3) datatype[] arrayname = new datatype[size];

eg int[] regd = new int[5];

Type of Array

Arrays are classified into two types,

1) Single Dimensional (1D) []
2) Multidimensional [][]

1) Single Dimensional :-

eg
```
class A
{
  public static void main(String args[])
  {
    int a[] = new int[5];
    a[0] = 10;
    a[1] = 20;
    a[2] = 30;
    a[3] = 40;
    a[4] = 50;
    for (i = 0; i < a.length; i++)
    {
      System.out.println(a[i]);
    }
  }
}
```

Direct initialization

```
public class A
{
  public static void main(String args[])
  {
    int a[] = {33, 3, 4, 5, 7};
    for(int i = 0; i < a.length; i++)
    {
      S.o.p(a[i]);
    }
  }
}
```

For-each loop for Java Array

We can also print the Java array using **for-each loop**.

↳ Print array elements one by one

Syntax

```
for(data-type : array)
{
  body;
}
```

89

```
class Arrayexp
{
  public static void main(String args[])
  {
    int arr[] = {33,11,22};
    for(int i : arr)
      System.out.println(i);
  }
}
```

User input :→

```
import java.util.Scanner;
class Test_array
{
public static void main(String args[])
 {
  int []ah=new int[5];
  S.o.p("Enter value to array");
  Scanner sc=new Scanner(System.in);
  for(int i=0;i<5;i++)
   {
    ah[i]=sc.nextInt();
   }
  System.out.println("values of array are");
  for(int i=0;i<5;i++)
   {
    S.o.p(ah[i]);
   }
 }
```

Demonstration of character Array

```
class char_array
{
public static void main(String args[])
 {
  char []ah = {'m','e','g','h','a'};
  System.out.println("values of array are");
  for(int i=0;i<5;i++)
  {
    System.out.println(ah[i]);
  }
 }
}
```

Finding Maximum Element in an Array

```
import java.util.Scanner;
class Max_array
{
  public static void main(String args[])
  {
    int[]g = new int[5];
    System.out.println("Enter value to array");
    Scanner sc= new Scanner(System.in);
    for(int i=0; i<5; i++)
    {
      g[i]=sc.nextInt();
    }
    int max=g[0];
    for(int i=0; i<5; i++)
    {
      if(max<g[i])
```

```
        {
            max = g[i];
        }
        S.o.p("The max element of the
        array is" +max);
    }
}
```

Finding Sum of the element of an Array

```
import java.util.Scanner;
class Sum_array
{
public static void main(String args[])
  {
    int[]g = new int[5];
    System.out.println("enter values");
    Scanner sc = new Scanner(System.in);
     int i;
    for(i=0; i<5 ; i++)
    {
      g[i]=sc.nextInt();
    }
int Sum = 0
  for(i=0;i<5 ; i++)
  {
    Sum = Sum+g[i];
  }
S.o.p("The Sum is" +Sum);
  }
}
```

Multidimensional Array

In Java multidimensional array; are actually array which is achieved by adding more square bracket.

→ A 2D array can be like a table o matrix of rows and columns.

Syntax

a) Declaration Syntax

datatype[][]arrayname = new datatype[s][s

eg:-

int[][]a = new int[2][2];

b) Initialization

arrayname[row][column] = value;

eg arr[0][0] = 1;

c) Direct Method

int[][] arrayname = {{},{}};

int [][] a= {{2,3},{4,5}};

```
import java.util.Scanner;
class TwoD_array
{
public static void main(String args[])
  {
    int[][]a = new int[3][2];
```

```
Scanner sc = new Scanner(System.in);
S.o.p("Enter the element of matrix");
For(int r=0; r<3; r++)
{
 for(int c=0; c<2; c++)
  {
   a[r][c] = sc.nextInt();
  }
 }
S.o.p("The elements are");
for(int r=0; r<3; r++)
{
 for(int c=0; c<2; c++)
  {
   S.o.p(a[r][c] + " ");
  }
  S.o.p();
 }
}
}
```

a[0][0]	[0][1]
2	1
[1][0]	[1][1]
4	3
[2][0]	[2][1]
5	4

```
class Testarray
{
 public static void main(String arg[])
 {
  int arr[][] = {{1,2,3},{2,4,5},{3,4,6}};
  For(int r=0; r<3; r++)
  {
   For(int c=0; c<3; c++)
   {
    S.o.p(arr[r][c] + " ");
   }
   S.o.p();
  }
 }
}
```

Addition of 2 Matrices in Java

```
class Test_array2
{
public static void main(String arg[])
 {
  int a[][] = {{1,3,4},{3,4,5}};
  int b[][] = {{1,3,4},{3,4,5}};
  int c[][] = new int[2][3];
 For(int r=0; r<2; r++)
 {
  For(int c=0; c<3; c++)
  {
   c[i][c] = a[r][i] + b[r][c];
   S.o.p(c[r][c] + "r");
  } S.o.p();
```

```
    }
  }
}
```

$$\begin{bmatrix} 1 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}_{2\times3} + \begin{bmatrix} 1 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}_{2\times3}$$

$$= \begin{bmatrix} 2 & 6 & 8 \\ 6 & 8 & 10 \end{bmatrix}_{2\times3}$$

Multiplication of Matrices

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$$

$$= \begin{bmatrix} 1*1+1*2+1*3 & 1*1+1*2+1*3 & 1*1+1*2+1*3 \\ 2*1+2*2+2*3 & 2*1+2*2+2*3 & 2*1+2*2+2*3 \\ 3*1+3*2+3*3 & 3*1+3*2+3*3 & 3*1+3*2+3*3 \end{bmatrix}$$

$$= \begin{bmatrix} 1+2+3 & 6 & 6 \\ 2+4+6 & 12 & 12 \\ 18 & 18 & 18 \end{bmatrix}$$

$$= \begin{bmatrix} 6 & 6 & 6 \\ 12 & 12 & 12 \\ 18 & 18 & 18 \end{bmatrix}$$

```
public class matrixmul
{
 public static void main(String args[])
 {
   int [][]a={{1,1,1},{2,2,2},{3,3,3}};
   int [][]b= {{1,1,1},{2,2,2},{3,3,3}};
   int c[][]= new int [3][3];
 For(i=0; i<3 ;i++)
 {
   For(j=0; j<3 ;j++)
   {
     c[i][j] = 0;
    For(int K=0; K<3 ;K++)
    {
      c[i][j]+ = a[i][K] * b[K][j]
    }
    S.o.p(c[i][j] + " ");
   }
   S.o.p( );
 }
}
}
```

Object

→ Instance of a class is called object.

→ An object in Java is essentially a block of memory that contains space to store variable (class does not reserved memory).

→ objects in Java are created using new operator
↳ used to assign memory

Syntax for object creation

① classname objectname;
objectname = new classname();

② classname object name = new classname();

eg:-
```
class A
{
  static int n, m;
  public static void main(String args[])
  {
    A ob = new A();
    ob.n = 10;
    ob.m = 20;
    System.out.println(+n);
    System.out.println(m);
  }
}
```

Methods in Java

→ A method is a block of code which only runs when it is called.

→ you can pass data, known as parameters/arguments.

→ Methods are used to perform certain actions & they are also know as functions.

create a Method

→ A method must be declared within a class.

→ It is defined with the name of the method, followed by parenthesis ().

Syntax

```
access specifier   return type   method name ( parameters ...)   ← may/may not
```

eg
```
public int add(int a, int b)
```

Addition of 2 no. using Method

```
Public class method_add
{
  public static void add()
  {
    int a = 20, b = 30;
    int c = a+b;
```

```
    System.out.println("The Sum is :" +c);
    }
    public static void main(String args[])
    {
      add();
    }
}
```

Types of Method

There are 2 types of method

a) Predefined Method
b) Userdefined Method

a) Predefined / Built-in Method :-

The method i.e already defined in Java class libraries.

→ We can directly use that method just by calling them in the program at any point.

eg → length(), equals(), compareTo(), Sqrt(), max() etc

↳ concat() : It concatenate 2 strings
S1.concat(S2);

↳ charAt() : Returns a characters by index position
S1.charAt(1);

↳ toUppercase() :- convert value to uppercase
S1.toUpperCase();

↳ toLowercase() :- convert value to lowercase
S1.toLowerCase();

↳ trim() : Remove space from both side of a string. eg S1.trim();

↳ abs() : Return absolute value
eg Math.abs(value);

↳ round() : It round the value to nearest integere
eg Math.round(a); a value

↳ min() / max() :

Access Modifier in Java :-

→ Access modifier in java helps to restrict the scope of a class, constructor, variable, method or data member.

→ It specifies the accessibility

→ There are 4 types of the access modifier used in Java

1) private
2) default
3) Public
4) Protected

1- Private :→

The access level of a private modifier is only with in the class. It can't be accessed from outside the class.

→ A Subclass/childclass does not inherit the private members of its parent class.

B.java

```
class A
{
  private int d=40;
  private void msg()
  {
    System.out.println("Hello");
  }
}
class B
{
  public static void main(String args[])
  {
    A ob = new A();
    System.out.println(ob.d); // compile Time Error
    ob.msg(); // compile time error
  }
}
```

Note :→

If you make any class constructor private, you can not create the instance of that class from outside of the class.

constructor :-

It is block of codes similar to the method. It is called when instance of a class is created.

B.java → It is a special type of method which is used to initialize the object.

```
class A
{
  private A()
  {
    void msg()
    {
      System.out.println("Hello");
    }
  }
public class B
{
  public static void main(String args[])
  {
    A ob = new A(); // compile Time error
  }
}
```

Note

A class can not be private or protected except Nested class.

2) Default :-

If we don't use any modifier, it is treated as default by default.

→ The default modifier is accessible only within the program.

→ In the below example, we have created two packages pack & mypack. We are

accessing the class A from outside its package, since A class is not public. So it cannot be accessed from outside the package

A.java

```
Package pack;
class A
{
  void msg()
  {
    System.out.println("Hello");
  }
}
```

B.java

```
package mypack;
import pack.*;
class B
{
  psvm(...)
  {
    A ob = new A();
    ob.msg();
```

3) Protected

→ The protected access modifier is accessible through inheritance only.

→ The protected modifier can be applied on the data member, method & constructor.

* It can't be applied on the class.

A.java

```
class A
{
  Protected void msg()
  {
    System.out.println("Hello");
  }
}
```

B.java

```
class B extends A
{
  public static void main(String args[])
  {
    B ob = new B();
    ob.msg();
  }
}
```

4) public

The public access modifier is accessible everywhere.

→ It has the widest scope among all other modifiers.

A.java

```
class A
{
  public void msg()
  {
    System.out.println("Hello");
  }
}
```

B.java

```
class B extends A
{
  public static void main(String args[])
  {
    B ob = new B();
    ob.msg(); } }
```

word

-All Java -Access Modifiers

Access modifier	within class	within Package	Outside pckg by subclass only (inheritance)	Outside Package
Private	Yes	No	No	No
Default	Yes	Yes	No	No
protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

~~String Buffer : StrBuffer~~ :→

String

→ In Java, String is a sequence of characters.

→ String is basically an object that represents sequence of char values.

→ The java.lang.String class is used to create String object.

```
String s1 = "Java";
String s2;
s2 = s1;
System.out.println(s2);
```

```
class Test_String
{
 public static void main(String args[])
 {
  char[] ch = {'H','e','l','l','o'};
  for(int i=0; i<ch.length; i++)
  {
   System.out.println(ch[i]);
  }
 }
}

class Test_String2
{
 public static void main(String args[])
 {
  String s1 = "Hello";
  System.out.println(s1);
 }
}
```

How to create String object ?

There are 2 ways to create String object :

1. By String Literal
2. By new Keyword

1. String Literal :

```
String s = "welcome";
```

→ String objects are stored in a special memory area known as String constant pool.

2. New Keyword :

```
String s = new String("Hello");
```

String_ex.java

```
public class String_ex
{
public static void main(String args[])
 {
  String S1 = "Hello";
  char ch[] = {'s','t','u','d','e','n','t','s'};
  String S2 = new String(ch);
                  ⇒ converting char to string
  String S3 = new String("Good");
  System.out.println(S1);
  System.out.println(S2);
  System.out.println(S3);
 }
}
```

Java StringBuffer class : ⇒

Java StringBuffer class is used to created mutable (modifiable) String.

→ The StringBuffer class in Java is same as String class except it is mutable i.e it can be changed.

constructors of StringBuffer class

1. StringBuffer() : Creates an empty String buffer with the initial capacity of 16

2. StringBuffer(String str) : Creates a String buffer with the specified String

3. StringBuffer(int capacity) : Create an empty String buffer with specified capacity as length.

Mutable String : ⇒

→ A String that can be modified or changed is known as mutable String.

1) append() method

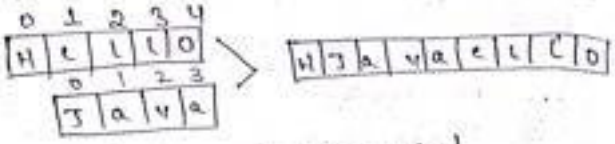
The append() method concatenates the given argument with the String.

```
class A
{
public static void main(String args[])
 {
  StringBuffer sb = new StringBuffer("Hello");
  sb.append("Good Morning");
  System.out.println(sb);
 }
}
```

2) StringBuffer insert() method :

The insert() method inserts the given String with this String at the given position.

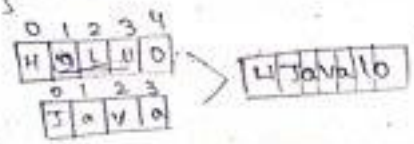
```
class Str_insert
{
public static void main(String args[])
{
StringBuffer sb=new StringBuffer
                ("Hello");

sb.insert(1,"Java");
System.out.println(sb);
```



3) StringBuffer replace() method

The replace() method replace the given String from the specified beginIndex & endIndex

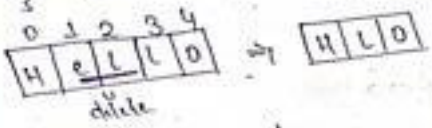
```
Str_replace.java
class Str_replace
{
public static void main(String args[])
{
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);
}
}
```



4) delete() method:

The delete() method of StringBuffer class deletes the String from the specified beginIndex to endIndex

```
Str_delete.java
class Str_delete
{
public static void main(String args[])
{
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);
}
}
```



5) reverse() method

The reverse() method of StringBuffer class reverse the current String.

```
Str_reverse.java
class Str_reverse
{
public static void main(String args[])
{
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);
}
}
```

H e l l o → o l l e H

Difference b/t StringBuffer & Builder

StringBuffer	StringBuilder
→ StringBuffer is Synchronized i.e thread Safe. It mean two threads can't call the methods of StringBuffer Simultaneously.	→ StringBuilder is non-Synchronized
→ Less efficient	→ More efficient

Parameter passing :

There are different ways in which parameter data can be passed in-to & out of method.

Type of parameter

a) Formal parameter

A variable & it's type as they appear in the prototype of the fun^n or method.

Syntax

fun^n name(data-type variable-name)

b) Actual Parameter :

The variable or expression corresponding to a formal parameter that appears in the fun^n in the calling environment.

Syntax

fun^n-name(variable(s));

Are values that are passed to a fun^n when it invoked

Eg: void add(int x, int y)
↳ Formal parameter
{
int z = x+y;
System.out.println(z);
}
add(2,3);
add(5,4); ← Actual Parameter

Important Method bore Parameter Passing

1. Pass By value / call by value :-

→ call by value means calling a method with a parameter as value.

→ The argument value is passed to the parameter

Eg: Public static void main(String args[])
{
Sum(1,2);
}
Public static int Sum(int x, int y)
{
return x+y;
}

Constructor :-

→ A constructor is a special member function / method / procedure / operation / behaviour / action of a class i.e used to create object of that class.

→ It has same name as the class itself.

→ It has no return type & is invoked or called using new operator.

Syntax :-

```
modifiere classname(datatype variable...)
{
  Statement 1;
  :
  Statement n;
}
```

A.java

```
public class A
{
  public static void main(String args[])
  {
    A ob = new A();
  }
  A()
  {
    System.out.println("constructor is called");
  }
}
```

A1.java

```
public class A1
{
  A1()
  {
    System.out.println("Hello");
  }
  public void show
  {
    System.out.println("good morning");
  }
  public static void main(String args[])
  {
    A1 ob = new A1();
    ob.show();
  }
}
```

Type of Java constructor :-

There are two type of constructore in Java.

1. Default constructor
2. Parameterized constructore

1. Default constructor :-

→ A constructor is called default constructor when it does not have any parameter.

Syntax

```
classname()
{
  ==
}
```

add.java

```
class add
{
  add()
  {
    int c = 10+20;
    System.out.println(c);
  }
  public static void main(String args[])
  {
    add ob = new add();
  }
}
```

purpose

it is used to provide the default value to the object like 0, null etc

```
class Student
{
  int Regd;
  String name;
  void display()
  {
    S.o.pt Regd + " " + name);
  }
  public static void main(String args[])
  {
    Student S1 = new Student();
    Student S2 = new Student();
    S1.display();
    S2.display();
  }
}
```

Note

if there is no constructors in a class, compiler automatically creates a default constructor.

3. **Parameterized constructor:-**

A constructor which has a specific number of parameter is called a parameterized constructor.

Add.java

```
public class Add
{
  Add(int m, int n)
  {
    System.out.println(m+n);
  }
  public static void main(String args[])
  {
    Add ob = new Add(10,20);
  }
}
```

Add1.java

```
public class Add1
{
  Add1(int m, int n)
  {
    System.out.println(m+n);
  }
  void add(int a, int b)
  {
    System.out.println(a+b);
  }
}
```

```
public static ...
{
    Add ob = new Add(10,20);
    ob.add(20,30);
}
}
```

Constructor overloading

Constructore overloading in java is a technique of having more than one constructor with different parameter lists.

eg
```
class Student
{
int id;
String name;
int age;
Student(int i, String s)
{ id = i; name = s;
System.out.println(id + " " + name);
}
Student(int i, String s, int ag);
{
  id = i;
  s = name;
  ag = age;
System.out.println(id+" "+name+" "+
                   age);
}
```

```
public static void main(String args[])
{
Student s1 = new Student(1, "Megha");
Student s2 = new Student(2, "Manali", 20)
}
}
```

Comparing & Identifying Object

→ In Java, the == operator compares that two references are identical or not, whereas the equals() method compares two objects.

→ objects are equal when they have the same state. objects are identical when they share the class identity.

eg
```
obj1 == obj2  (test identity)
obj1.equals(obj2)  (compares equality)
```

Inheritance in Java :-

→ Inheritance in Java is a ~~method~~ mechanism in which one object acquires all the properties & behaviours of parent class.

→ The idea behind inheritance in java is that you can create new classes that are built upon existing classes.

→ When we inherit from an existing class, we can reuse method & fields of the parent class. We can add new method

field in our currecnt class.

→ Inheritance represents the IS-A relationship which is also known as Parent-child relationship

Important terminology

i) Super class :- The class whose features are inherited is known as Superclass (or base class / parent class).

ii) Sub class :- The class that inherit the other class is known as Subclass (or derived class / extended class / child class)

Syntax

The keyword used for inheritance is "extends"

↳ indicates that you are making a new class that derived from an existing class.

```
class childclass extends Parentclass
{
  codes
}
```

```
class parentclass
{
  ==
}
class childclass extends Parentclass
{
  ==
}
```

B.java

eg

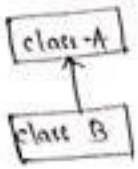
```
class A
{
  int a=10, b=20;
}
class B extends A
{
  public static void main(String args[])
  {
    int c=a+b;
    System.out.println("The sum is:"+c);
  }
}
```

Types of Inheritance :

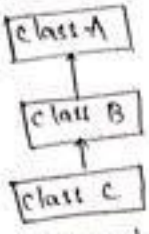
→ on the basis of class, there can be 3 types of inheritance in Java

→ * Multiple & Hybrid inheritance supported through interface only)

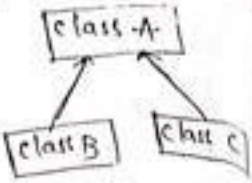
1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance



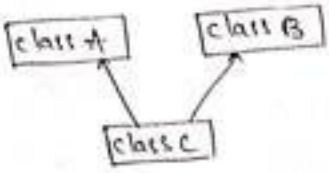
1) Single Inheritance



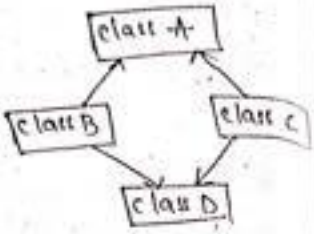
2. Multilevel



3. Hierarchical



4. Multiple



5. Hybrid

1) Single Inheritance:

when a only 1 class) class inherits another class, is known as Single Inheritance

Syntax

```
Public class A
{
  ==
}
Public class B extends A
{
   ==
}
```

```
B.java
Public class A
{
 public void msg()
 {
  System.out.println("Hello");
 }
}
public class B extends A
{
 public static void main(String args[])
 {
  public void display()
  {
  System.out.println("Good Morning");
  System.out.println("How are you?");
  }
  B ob = new B();
  ob.msg();
  ob.display();
 }
}
```

2. Multilevel inheritance

When there is a chain, it is called multilevel inheritance

Syntax

```
public class A
{ ==
}
public class B extends A
{ ==
}
public class C extends A B
{ - }
```

Z.java

```
class X
{
  public void ad mx()
   { System.out.println(" X method");
     }
class Y extends X
{ public void my()
  { S.o.p(" Y method");
   }
}
class Z extends Y
{ public void mx()
  { System.out.println(" Z method");
   }
  public static void main(String args[])
  { Z ob = new Z();
    ob.my();
    ob.my();
    ob.mx();
  }
}
```

3. Hierarchical class :

If one class is inherited by many subclasses i.e known as hierarchical class.

```
public class A { ... }
Public class B extends A { ... }
Public class C extends A { ... }
```

Z.java

```
class X
{
  public void MX()
  { S.o.p(" X method");
   }
}
public class Y extends X
{ public void MY()
  { S.o.p(" Y method");
   }
}
public class Z extends X
{ public void MX()
  { S.o.p(" Z method");
   }
  public static void main(String args[])
  { Y ob = new Y();
    Z ob1 = new Z();
    ob.MX();
    ob.MY();
    ob1.MX();
    ob1.MY();
  }
}
```

* Java does not support Multiple Inheritance. This means that class does not can not extends more than one class.

The Super keyword

⇒ The Super keyword is used to differentiate the members of Superclass / Base class from the members of Subclass / childclass, if they have same name.

Differentiating the members

If a class is inheriting the properties of another class. And if the members of the Superclass have the names same as the Sub class, to differentiate these variables we use Super keyword

Super.variable();
Super.method();

child.java

```
class Parent
{
int n = 10;
public void display()
{
System.out.println("This is Super class");
}
}
class child extends Parent
{
int n = 20;
public void display()
{
System.out.println("This is Sub class");
}
public void my_method()
public static void main(String args[])
{
child ob = new child();
ob.display();
S.super.display();
S.o.p(ob.num);
S.o.p(Super.num);
}
public static void main(String args[])
{
child obj = new child();
obj.my_method();
}
}
```

Note

1) Super can be used in method which is non-static.
2) In the above example Super is used in a non-static method named as my_method

Polymorphism

It is an ability of an object to take on many forms.

```
int x = 10, y = 64;
System.out.println("Hi");  // 1 arg as String
System.out.println("x = " + x);
S.o.p("x = " + x + "y = " + y);
```

Two type of polymorphism

a) Compile Time polymorphism (static or Early Binding)

b) Run Time (Dynamic or Late Binding) or Dynamic Dispatching

a) Compile Time :

The object bind / call / invoke / associated with the appropriate method is already determined or decided prior to the compilation.

b) Run Time

The object bind / call / invoke / associate with the appropriate method is determined or decided at run time only.

```
class calculation
{
 void add(int a, int b)
  {
   System.out.println(a+b);
  }
 void add(int a, int b, int c)
  {
   System.out.println(a+b+c);
  }
 public static void main(String args[])
 {
  calculation ob = new Calculation();
   ob.add(10,20,30);
   ob.add(20,30);
  }
}
```

→ There are 3 ways to achieve polymorphism

1. Overloading
2. Overriding
3. Interfacing

1. Method overloading

Multiple methods by same name but different parameters, is known as Method overloading.

→ Compiler knows which method / function to call based on the parameter it is passed.

Why ?

Suppose we have to perform addition of the given number but there can be any no. of arguments, if we write the method such as add2(int,int) bore two parameters and add3(int,int,int) bore 3 arguments then it may difficult

bore programmers -to understand behaviour so we perform method overloading.

Two ways -to overload method

1. By changing number of arguments
2. By changing the datatype

Note: Method overloading is not possible by changing the return type of the method.

By changing no. of arguments

```
class calculation
{
 void add(int a,int b)
 {
  System.out.println(a+b);
 }
 void add(int a,int b, int c)
 {
  System.out.println(a+b+c);
 }
 public static void main(String args[])
 {
  calculation ob=new calculation();
  ob.add(10,20);
  ob.add(10,20,30);
 }
}
```

By changing datatype

```
class A
{
 void add(int a,int b)
 {
  System.out.println(a+b);
 }
 void add(double a,double b)
 {
  System.out.print(a+b);
 }
 public static void main(String args[])
 {
  A ob=new A();
  ob.add(10,20);
  ob.add(10.5,20.5);
 }
}
```

2. Method overriding → Same name, Same parameters

If Subclass has the same method name as the parent class it known method overriding

Rules

1. Method must have same name as in the parent class.
2. Method must have same parameters as in the parent class.

eg

```
class V
{
  void run()
  {
    S.o.pt("inside parent class");
  }
}
class B extends V
{
  void run()
  {
    S.o.pt("inside child class");
  }
  public static void main(String args[])
  {
    B obj = new B();
    obj.run();   // B obj2 = new B();
                 obj2.run();
  }
}
```

class

Pass by value & Pass By Reference

→ Pass by value is also called call by value

→ Pass by Reference is also called call by Reference.

eg

```
public class B
{
  public void twofold(int n)
  {
    n = 2*n;
    S.o.pt("value inside twofold method="
           + n);   // 20
  }
  public void demo()
  {
    int x = 10;
    S.o.pt("value before twofold method call="+x);
    twofold(x);   // 10
    S.o.pt("value after twofold method = "+x);
                  // 10
  }
}
```

Output

value before twofold method call = 10
value inside twofold method = 20
value after twofold method = 10

Call By Reference

-Arguments are passed by reference rather than passing by value.

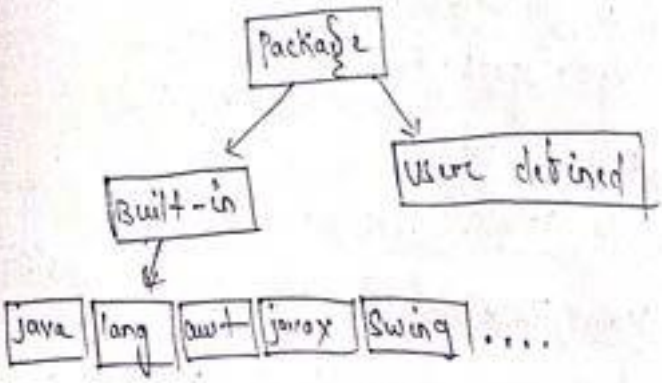
```
class callByRef
{
  static int x=1, y=2, s;
  public void Sum()
  {
    S.o.p("original value" +x+ " "+y);
  }
  s = x+y;
  S.o.p("original Sum = " +s);
  callByRef ob = new callByRef();
  Test(ob);
  S.o.p("value after call" +x+ " "+y
}
public void Test(callByRef ob)
{
  callByRef.x = 100;
  callByRef.y = 200;
  int p = x*y;
  S.o.p("x*y = " + p);
}
```

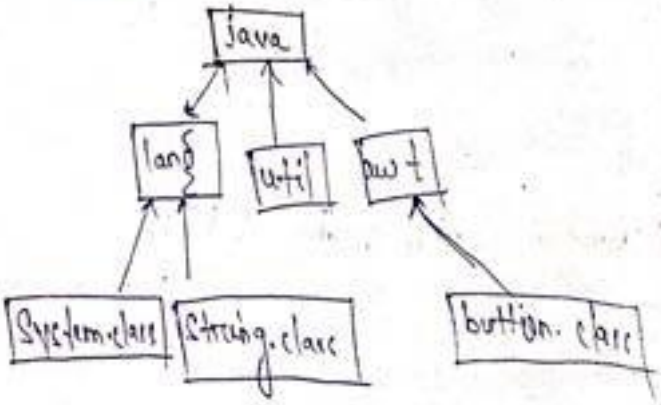
JAVA PACKAGE

→ Package are used in java in order to prevent naming conflict, to control access, to make searching/locating & usage of classes, interfaces, enum etc easier.

→ A package can be defined as a grouping of related types (classes, interfaces etc). providing access protection & namespace management.

→ Package helps us to categorize our code in different folders.





→ The "package" keyword is used to create a package in java.

How to compile Java Package

javac -d directory javafilename

(-d switch to specifies the destination where to put the generated class.

⇒ javac -d . Simple.java

⇒ . (.) package within the same directory

javac -d F:\Java\pck Simple.java

How to Run java package

java packagename.classname

java mypack.Simple

Simple.java

```
package mypack;
public class Simple
{
  public static void main(String args[])
  {
    System.out.println("welcome");
  }
}
```

→ changing drive from C to D

:\>D:

How to Access package from another package ?

There are 3 ways to access package from outside the package.

1. import package.*;
2. import package.classname;
3. Fully Qualified name

1) Using package Name :

→ If we use package.*, then all the classes & interfaces of this package will be accessible but not sub package.

→ The import keyword is used to make classes & interface of another package accessible to the current package.

A.java

```
package testpack;
public class A
{
  public void msg()
  {
    System.out.println("Hello");
  }
}
```

B.java

```
package mypack;
import testpack.*;
public class B
{
```

```
public static void main(String args[])
{
A ob = new A();
ob.msg();
}
}
```

compile

F:\Java\PCK> javac -d F:\Java\PCK A.java
F:\Java\PCK> javac -d F:\Java\PCK B.java
Run F:\Java\PCK> java mypack.B

2) Using packagename.classname

If we import packagename.classname only declared class of this package will be accessible.

Ex:-

A.java

```
package testpack;
public class A
{
  public void msg()
  {
    System.out.println("Hello");
  }
}
```

B.java

```
package mypack;
import testpack.A;
public class B
{
public static void main(String args[])
{
A ob = new A();
ob.msg();
}
}
```

3) Using Fully Qualified Name :

If we use fully qualified name then only declared class of this package will be accessible. Now there is no need to import.

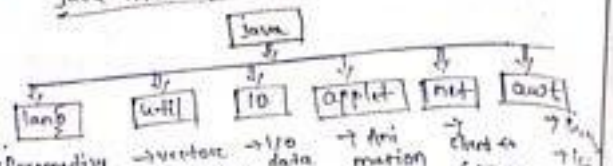
A.java

```
package testpack;
public class A
{
  public void msg()
  {
    System.out.println("Hello");
  }
}
```

B.java

```
package mypack;
public class B
{
  public static void main(String args[])
  {
    testpack.A ob = new A();
    ob.msg();
  }
}
```

Java API Package



JVM (Java Virtual Machine) :

→ JVM is a specification that provides runtime environment in which java bytecode can be executed.

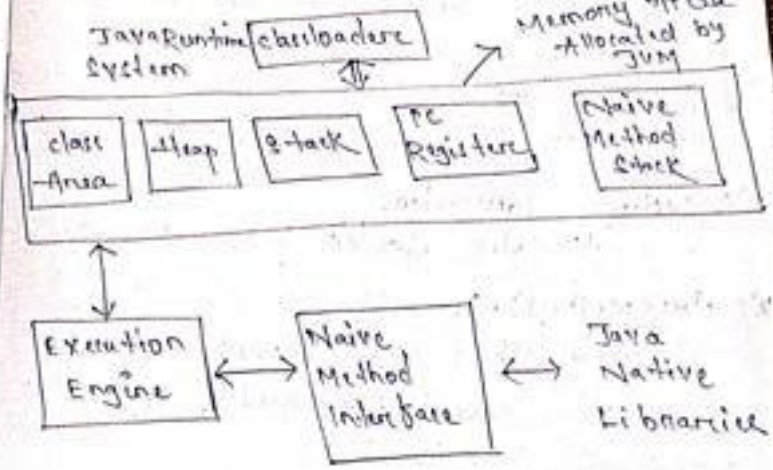
What it does ?

The JVM performs following operation

→ Loads code
→ verifies code
→ Executes code
→ provides runtime environment

JVM - Architecture

→ It contains classloader, memory area, execution engine etc



1) Class Loader:

→ It is a subsystem of JVM which is used to load class files.

→ Whenever we run the java program, it is loaded first by the classloader.

→ There are 3 built-in classloader in Java

a) Bootstrap ClassLoader:

Super class of extension classloader.

→ It loads rt.jar file which contains all class files of java standard like like java.lang, java.net, java.io, java.util etc

b) Extension ClassLoader:

This is the child classloader of Bootstrap & parent classloader of System classloader.

→ It loads the jars files located inside $Java_HOME/jre/lib/ext directory

c) System ClassLoader:

It loads classfile from classpath.

2) class(Method) Area:

Stores per-class structures such as runtime const. pool, field & method data, code for method.

3) Heap:

Run-time data area in which objects are allocated.

4) Stack:

It holds local variables & partial results, and plays a part in method invocation & return.

5) PC Register:

contains address of the JVM instruction currently being executed.

6) Native Method Stack:

It contains all the native methods used in the appl^n.

7) Execution Engine:

It contains:

a) A virtual processors

b) Interpreter

c) JIT (Just-in-Time): JIT compiles parts of the byte code that have similare fun^n at the same time

8) Java Native Interface:

It is a framework which provides an interface to communicate with another appl^n written in another language like C, C++ etc.

Exception Handling in JAVA

Errors:
Errors are the wrong that make a program go wrong.

Compile-Time Errors:
Syntax errors at compile time because of

→ Missing Semicolons

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

→ Missing(or mismatch of) brackets in class & method body.
→ Misspelling of identifiers & keywords
→ Missing double quotes in String.
→ Use of undeclared variable
→ Incompatible types in assignment
→ Bad reference to object ... etc

Reading Data from Keyboard

There are many ways to read data from keyboard. For example

a) InputStreamReader
b) Scanner
c) console / command Line Arguments
d) DataInputStream.

a) InputStreamReader class:
It can be used to read data from keyboard.

→ It performs 2 tasks.
  i) connect to input stream of keyboard
  ii) converts the byte oriented stream into character-oriented stream

BufferReader class:
It can also be used to read data line by line by readLine() method.

```
import java.io.*;
class A
{
public static void main(String args[])
  {
  InputStreamReader r = new InputStreamReader
                        (System.in);
BufferReader b = new BufferReader(r);
System.out.println("Enter your Name");
  String name = b.readLine();
} S.O.P(name);
```

b) Java Scanner class:

There are various ways to read input from the keyboard, the java.util.Scanner class is one of them.

→ The Java Scanner class breaks the input into tokens using a delimeter that is whitespace by default.

commonly used Method of Scanner class:

i) public String next(): —
It returns the next token from the Scanner.

ii) public String nextLine(): —
It moves the Scanner position to the next line & returns the value as a String.

iii) public byte nextByte(): —
It scans the next token as Byte

iv) public int nextInt(): —
It scans the next token as Integer

v) public long nextLong(): —
It scans the next token as Long.

vi) public float nextFloat(): —
It scans the next token as Float.

vii) public double nextDouble(): —
It scans the next token as Double.

1) import java.util.Scanner;
class S
{
public(...)
{
Scanner sc = new Scanner(System.in);
S.O.P("Enter your Name");
String name = sc.next();
S.O.P("Name is:" + name);
}
}

Java Scanner example with Delimiter(white space):

import java.util.*;
class S1
{
public(...)
{
String input = "10 tea 20 coffee";
Scanner sc = new Scanner(input);
sc.useDelimiter(" ");
S.O.P(sc.nextInt());
S.O.P(sc.next());
S.O.P(sc.nextInt());
S.O.P(sc.next());
S.close();
}
}

Addition of 2 no:

```
import java.util.Scanner;
class input
{
 psvm(...)
 {
  Scanner Sc = new Scanner(System...
   S.o.p("Enter 1st No.");
  int a = Sc.nextInt();
  S.o.p("Enter 2nd No.");
  int b = Sc.nextInt();
  int c = a+b;
  S.o.p(c);
 } }
```

c) Command Line - Argument as input

Java main method consist array of Strings as main(String args[]). When we run the program, the array will be filled with the value of any arguments through command lines.

→ Each element in an array can be referenced or accessed by an array name & index (a[0])

eg c:> javac -A.java
c:> java -A 1 2 5

Here command line contains 3 arguments

a[0] is 1
a[1] is 2
a[2] is 5

```
class -A
{
 public static void main(String args[])
 {
  System.out.println("-Hello");
  System.out.println(a[0]);
 }
}
```

compile :- c:...> javac -A.java
Run :- c:...> java -A Monali

O/P -Hello Monali

```
class -A1
{
 public static void main(String args[])
 {
  int i=0;
  for(i=0; i<a.length; i++)
  {
   System.out.println(a[i]);
  }
 }
```

Java IO :-

Java IO(input & output) is used to process the input & produce the output.

→ The java.io package contains all the classes required for input & output operations.

Stream

A stream is a sequence of data.

→ In java, a stream is composed of bytes. It's called a stream because it is like stream of water that continues to flow.

→ In java 3 streams are created for us automatically.

1) System.out : Standard output Stream
2) System.in : Standard input Stream
3) System.error : Standard error Stream

```
System.out.println("Simple Message!");
System.error.println("error Message!");
int i = System.in.read();
System.out.println((char)i);
```

Output Stream :

Java appl? use an output Stream to write data to a destination.

→ It may be a file, an array, peripheral device.

Input Stream :

Java appl? uses an input Stream to read data from a source ; it may be a file.

OutputStream class :

It is the superclass of all classes representing an output stream of byte.

→ An output stream accepts output bytes & sends them to some sink.

Useful Methods of OutputStream

Method	Description
1) public void write(int) throws IOException	is used to write a byte to the current output stream
2) public void write(byte[]) throws IOException	is used to write an array of byte to the current output stream
3) public void flush() throws IOException	flushes the current output stream
4) public void close() throws IOException	is used to close the current output stream

InputStream class :

→ It is the superclass of all classes representing an input stream of bytes

Useful Methods of InputStream

Method	Description
1) public abstract int read() throws IOException	→ Reads the next byte of data from the input stream. It returns -1 at the end of the file.
2) public int available() throws IOException	→ returns an estimate of the no. of bytes that can be read from the current input stream
3) public void close() throws IOException	→ is used to close the current input stream

Java FileOutputStream class :—

Java FileOutputStream is an output stream used for writing data to a file. (write byte)

1) import java.io.FileOutputStream;
public class Fileoutputexample
{
public static void main(String args[])
{
try
{
FileOutputStream fout = new FileOutputStream("D:\\t1.txt");
fout.write(65);
fout.close();
System.out.println("Success!");
}
catch(Exception e)
{
System.out.println(e);
}
}
}

t1.txt
A

2) write String

import java.io.FileOutputStream;
public class FileExample
{
public static void main(String args[])
{
try
{
FileOutputStream fout = new FileOutputStream("D:\\t1.txt");
String s = "Welcome to my file";
byte b[] = s.getBytes();
fout.write(b);
fout.close();
System.out.println("Success");
}

```
catch Exception e)
{
  System.out.println(e);
}
}
}
```

Java FileInputStream class

Java FileInput Stream class obtains input bytes from a file.

Java FileInputStream class Methods

Method	Description
int available()	It is used to return the estimated no. of bytes that can be read from the input stream
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to b.length bytes of data from the input stream.
void close()	It is used close the stream

Read Single character

```
import java.io.FileInputStream;
public class DataStreamEx
{
  public static void main (String args[])
  {
    try
    {
      FileInputStream fin = new FileInputStream
            ("D:\\u.txt");
      int i = fin.read();
      System.out.println((char) i);
      fin.close();
    }
    catch (Exception e)
    {
      System.out.println(e);
    }
  }
}
```

Read all chnacters

```
import java.io.FileInputStream;
public class A
{
  public static void main (String args[])
  {
    try
    {
      FileInputStream fin = new FileInputStream ("D:\\u.
                                                  txt");
      int i=0;
```

```
    catch(Exception e)
    {
      System.out.println(e);
    }
  }
}
```

Java FileInputStream class

Java FileInput Stream class obtains input bytes from a file.

Java FileInputStream class Methods

Method	Description
int available()	It is used to return the estimated no. of bytes that can be read from the input stream
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to b.length bytes of data from the input stream.
void close()	It is used close the Stream

1) Read Single characters

```
import java.io.FileInputStream;
public class DataStreamEx
{
  public static void main(String args[]
  {
    try
    {
      FileInputStream fin = new FileInputS
              ("D:\\dl.txt");
      int i = fin.read();
      System.out.println((char)i);
      fin.close();
    }
    catch(Exception e)
    {
      System.out.println(e);
    }
  }
}
```

2) Read all chnacters

```
import java.io.FileInputStream;
public class A
{
  public static void main(String args[])
  {
    try
    {
      FileInputStream fin = new FileInputStream("D:\\dl.
                                                txt");
      int i=0;
```

```
while((i = fin.read()) != -1)
{
  System.out.println((char)i);
}
fin.close();
}
catch(Exception e)
{
  System.out.println(e);
}
}
}
```

File Handling in Java

⇒ In Java, File Handling is done with the help of the File class of the java.io package.

File Handling operation

1. Creating a new file
2. Writing in a file
3. Reading an existing file
4. Deleting a file

→ To perform any of the above operations on a file in java, we need to create an object of the File classes

```
import java.io.File;
File obj = new File("Filename.txt");
```
↳ Specify the name of the file

1. Creating a New file :

```
import java.io.File;
import java.io.FileWriter;
public class CNF
{
  public static void main(String args[])
  {
    File ob = new File("abc.txt");
    try
    {
      ob.createNewFile();
      S.O.P("Success full");
    }
```

```
catch (IOException e)
{
  S.O.P(e);
}
}
}
```

↳ if the file can't be created bcoz some issue

→ To create a file in java, you can use the createNewFile() method.

→ This method returns a boolean value i.e T/F. (The method is enclosed in a try... catch)

2. Write to a File

→ We use FileWriter() It is used to write character-oriented data to a file) class (Reading/writing text file) together with its write() method to write some text to the file we created.

→ You should close it with the close() method.

```
import java.io.FileWriter;
import java.io.IOException;
Public class WriteToFile
{
  Public static void main (String args[])
  {
    try
    {
      FileWriter Fo = new FileWriter("a1.txt");
      Fo.write("Tricky but fun enough");
      Fo.close();
      S.O.P("Success full");
    catch (IOException e)
    {
      S.O.P("Error");
      e.printStackTrace();
    }
  }
}
```

3. Read a file ;—

We use the Scanner class to read the contents of the file

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
Public class A
{
  psvm(String args[])
  {
    try
    {
      File obj = new File("a1.txt");
      Scanner myReader = new Scanner(obj);
      while(myReader.hasNextLine())
      {
        String data = myReader.nextLine();
        S.O.P(data);
      }
```

```
        myReader.close();
    } catch(FileNotFoundException e)
        {
          S.o.pc("Error");
        } e.printStackTrace();
        }
}}
```

4. Deleting file :-

To delete a file, use the delete() method

```
import java.io.File;
public class DeleteFile
{
  public static void main(String args[])
  {
    File obj = new File("a1.txt");
    if(obj.delete())
    {
      S.o.pc("Deleted file : "+obj.getName());
    }
    else
    {
      S.o.pc("Failed to delete the file");
    }
  }
}
```