Vision Computers

Industrial training :-

JAVA

→ It is object oriented programming Language
→ It is introduced by Sun microsystems in 1996.
→ Java is an improved language from c and c++
→ It is also known as web application language.

Features of web application Language:-

1. Cross platform / OS independency
2. cross device / device independency
3. Cross network / Network independency

History of Java:-

1990: Sun micro system was started
→ 6 programmers started and Led by James Gosling

1991: They started one project called "Green project".
→ That project is to develop programming Language
→ This Language is for sake of electronic device

1992:- Oak Language introduction.

1993:- In this year they are promoting totally Oak application.

1994:- Oak been converted for web application

1995:- HOT Java browser was introduced.

1996:- Java Language was introduced.
JDK → Java development kit

2007:- Oracle corporation

Contents of Java:-

1. Core Java / Java Language
   ↓
   Coding Skills
2. Advanced Java / Web technologies } Website development
   JSP/ JdbC, Servelets (HTML)
   (sql)
   3-tier architecture

3. Java full stack
   ↓
   Core Java
   Adv Java
   MVC (spring boot) frame work
4. Android Applications
   (studio)

Core Java:-

Features:-

→ Java is a case sensitive Language (Lower case)
→ It is high Level language
→ Java is Oop language
→ Java is Compiled & interpreter Language
→ Byte Code Convertion
→ Native code Generation
→ Support of External Languages
   Third party notation
   Java, Html
   Java, XML
   Java, Sql
→ It's a Secured & robust programming language
→ It's a Multithreaded programming Language
→ Types of User interfaces
   1. GUI (Graphical User interface)
   2. CUI (Character User interface)

2/1/2023

Implementation Requirements:-

Hardware req:-
1. P2 processor onwards
2. 4GB RAM Onwards
3. 540 GB Harddisk onwards

Software req:-
1. Any OS
2. JDK 11 (Oracle Corporation)
   After installation of JDK Path is to be set

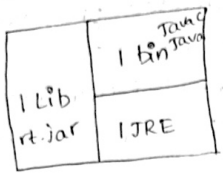
Other S/W's:
1. edit plus } used to develop Core Java
2. Notepad ++
3. Eclipse } used to develop Java, full stack.
4. Net beans

* How to develop in online?
  Search:- GDB Compiler for Java

Program Development:-

1. Write code in
   filename.Java using notepad Editor
2. Compile with
   Javac compiler
3. Run with
   Java interpreter.

JDK architecture:-



Jar -> Java archive

A.Java
↓Javac
A.class
↓
Byte code
↓Java
Java Native code
↓JRE
CPU

Java program Syntax:-

```
Class A
{
public static void main (string args[])
{
System.out.println("Welcome");
}
}
```

Save:- Cntr + S

Compile:- Javac A.Java

Run: Java A.

Contents:-

1. I O Statements
2. Datatypes & Variables
3. Operators
4. Control Statements [if, Loop]
5. Logic Oriented programming
6. String, math, data
7. Exception handling
8. I O Stream
9. OOp
10. Multithreading
11. Collections & Generics.

Java library:-

-> Every Language Library contains developer commands.
-> Java library is in the form of package.

Package:-

It is a Container of classes & Sub-packages interface.

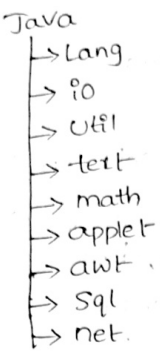
List of packages:-

1. Java
2. Javax
3. Sun

Java package:-

It's a Container of Sub-packages.

Hierarchy of Java package:-



Note:-

Java.Lang is default imported package.

1. IO Statement

O/p Handling:-

The process of writing/printing some text console

2. Method:-

Java.Lang.System.out.print("Hello");

System.out.println();

-> println includes new line

* WAP Hello World

First.Java

```
Class First
{
Public static Void Main(string args[])
{
 System.out.print("Hello World");
}
}
```

> Javac First.Java

> Java First.

* WAP to print World Wide Web.

Second.Java

```
class Second
{
Public Static Void main (string args[])
{
System.out.println("World");
System.out.println("Wide");
System.out.println("Web");
}
}
```

* Write Java token

-> Java tokens are keywords:

abstract	default	if	Package
assert	do	implements	private
boolean	double	import	protected
break	else	instanceof	public
byte	enum	int	return
Case	Extends	interface	Short
catch	final	Long	Static
Char	finally	native	Strictfp
class	float	new	Super
Continue	for	null	Switch
this	Void	true	Synchronized
throw	Voltile	false	
throws	While		
transient	Const		
try	goto		

Escape characters & Escape Machanism :-

Escape character :-

→ White Space characters are known as Escape character

\n → new line
\t → new horizontal tab
\b → back Space

* Write a python to print

I Like Java
Language
So much

Pat.Java

```
class pat
{
public Void static main (string args[])
{
System.out.println ("I \t like\t Java\n\t language
                     \n \t\t so \t much");
}
}
```

Escape machanism:-

→ The process of ignoring meaning of reserved words/characters

Regular	Escape
\b	\\b
\n	\\n
\t	\\t
"	\"
'	\'
\	\\

* WAP to print

"World"
'Java'
\tweet\

test.Java

```
class test
{
 p.s.v.m (String args[])
{
 S.O.P("\"World\"");
 S.O.P ("\'Java\'");
 S.O.P ("\\tweet\\");
}
}
```

Datatypes & Variables :-

Data type
- Primitive
- Non-primitive

Non-primitive :-

→ Non-primitive datatypes are collection of Object type

Example:-
String name = "Vision";

Java.Lang.String

→ It holds multiple values upto 2GB

Primitive :-

→ These types holds one value

Primitive
- numeric
- Non numeric

Numeric:-

→ This can be read +ve, -ve, int, float.

Datatype	Purpose	range	no of bytes
1. byte	it holds very small int value	-128 to +127	1 byte
2. short	Medium int val	±32K	2 byte
3. Int	huge int val	±214K	4 byte
4. Long	very huge val	processor	8 byte
5. float	floating val	7 decimal	4 byte
6. double	floating no's	15 decimal	8 byte

Note:-

→ Java default integer type is Int

→ Java default floating type is double

Non-numeric:-

Char:-

→ It holds 1 char

→ Storage Capacity is 1 byte

boolean:-

→ It holds either true or false

→ Storage Capacity is 1 bit.

Variable:-

→ A username temporary storage location(s) of RAM

Features:-

→ It should be declared first

→ It never be a reserved word

→ It can contain alphabets, numbers, special character (_, $)

→ First character never be a digit.

Variable initialization & declaration:-

```
int x;
int x,y;
String name="vision"
float op2=10.3, op3;
```

Floating Constants:-

```
float op1 = (float) 10.3
float op2 = 10.3f
float op3 = 10.3F
```

Long Constants:-

```
Long x = 100L;
Long x = 100l;
```

Input Handeling:-

The process of reading set of characters from keyboard

1. Java.lang.System.in

→ This object represents input device

2. Java.util.scanner

Scanner:-

→ This class object reads flow of characters from keyboard

Usage:-

```
import Java.util.Scanner;
Create Scanner object
Scanner sc = new Scanner(System.in);
```

New operator:-

→ It allocates the memory dynamically

Methods:-

```
int n = sc.nextInt();
float A = sc.nextFloat();
double B = sc.nextDouble();
Long C = sc.nextLong();
String S = sc.next(); -> one word
String S = sc.nextLine(); -> Multiple words
```

* Write a program to read your name & print with Hello:xxxxx

```
import java.util.Scanner;
class Test
{
 P.S.V.m (String args[])
 {
  Scanner sc=new Scanner (System.in);
  S.O.P ("Enter your name");
  String name=sc.nextline();
  S.O.P ("Hello:"+name);
 }
}
```

1. WAP to read 2 integers and print their Sum
2. WAP to read 3 strings join together and display as Concatination String

1.

```
import java.util.Scanner;
class Test
{
 p s v m (string args[])
 {
  Scanner sc = new Scanner (System.in);
  S.O.p ("Enter Value of A");
  int A = sc.nextInt();
  S.O.p ("Enter Value of B");
  int B = sc.nextInt();
  int C = A+B;
  S.O.p("Sum of A&B is"+C);
 }
}
```

Output:-

```
Enter Value of A
10
Enter Value of B
20
Sum of A&B is 30
```

2Ans

```
import java.util.Scanner;
class Student
{
psvm (string args[])
{
Scanner SC= new Scanner (System.in);
S.O.P(" Enter S1");
String S1= SC.nextLine();
S.O.p ("Enter S2");
String S2= SC.nextLine();
S.O.p ("Enter S3");
String S3= SC.nextLine();
String S4=S1+S2+S3;
S.O.p ('Concatinated strings are:"+S4);
}
}
```

Output:-

```
Enter S1
Kejiya
Enter S2
Dasina
Enter S3
Beautiful
Concatinated strings are: KejiyadasinaBeautiful
```

Operators:-

1. Arithmatic Operator

→ +, −, *, /, %

* WAP to read Length & breadth of rectangle print its area & perimeter

```
import java.util.Scanner;
Class Rectangle
{
psvm (string args[]);
{
Scanner Sc = new Scanner (System.in);
int l,b;
S.O.P(" Enter Length, breadth");
L= SC.nextInt();
b= Sc.nextInt();
int area = L*b;
int perimeter = 2*(l+b);
System.out.print(" perimeter ="+ perimeter);
S.O.p("Area ="+area);
}
}
```

Out put:-

```
Enter Length, breadth
10
20
perimeter= 60
Area = 200
```

* WAP to read side of a square print its area & perimeter.

```
import java.util.Scanner;
Class Square
{
psvm (String args[])
{
Scanner sc= new Scanner(System.in);
int side;
S.O.P("Enter side of a Square");
side = sc.nextInt();
int area = side * side;
int perimeter = 4 * side;
S.O.P("Area:"+area)
S.O.P("Perimeter:"+perimeter);
}
}
```

* WAP to read radius of a circle, print its area & perimeter.
* WAP for PTR, and Compute SI

2. Relational (or) Comparision operator: -

```
<
>
>=
<=
==
!=
```

3. Logical operator: -

1. && → all condition to be satisfied.
2. || → any one condition to be satisfied
3. ! → flips input

truth table: -

C1	C2	&&	||
F	F	F	F
T	F	F	T
F	T	F	T
T	T	T	T

i/p	i/p
1	0
0	1

4. Assignment Operator: -

```
=
+=
-=
*=
/=
```
} to represent Expression form as Operand form

5. Unary operator: -

→ only one operand is required
→ changes by 1

increment (++) (+1)
decrement (--) (-1)

Post Change	pre Change
x++	++x
x--	--x

* program for Circle Dimensions

```
import java.util.Scanner;
class circle
{
Public static void main (string args[])
{
Scanner sc= new Scanner(System.in);
System.out.print("Enter value of radius");
int r= sc.nextInt();
```

```
double Area = 3.14*r*r;
double P = 2*3.14*r;
S.O.p ("Area = "+area);
S.O.P ("perimeter = "+P);
}
}
```

Output :-

Enter Circle of a radius
Area = 113.03999999999
Perimeter = 37.68.

* PTR Program

```
import java.util.Scanner;
Class Simple
{
 P S V m (string args[])
{
Scanner Sc = new Scanner (System.in);
S.O.p ("Enter P,t,r values");
float P = Sc.nextFloat();
float T = Sc.nextFloat();
float R = Sc.nextFloat();
float SI = (P*t*r)/100;
S.O.P ("Simple Interest = "+SI);
}
}
```

Output :-

Enter P,T,R values 2 5 6
Simple Interest = 0.6

* WAP to read an integer increase it's value by 1 and print the same

```
import java.util.Scanner
Class C1
{
P S V m ( string args[])
{
Scanner Sc = new Scanner (System.in);
System.out.println ("Enter number");
int n = Sc.nextInt();
n++;
S.O.P (++n); (post increment)
S.O.p (n); (pre increment)
}
}
```

Comment operator :-

// --> Single line Comment
/* == /* Multiline Comment Operator

Ternery operator :-

This is used to test for a simple condition there two Condition T/f. These also called Conditional Operator.

(?) (:)

Ex:- (Condition);
Statement1 : Statement2
(True) (false)

(condition) ? var1 : var2

True → var1

false → var2

Write a program to read two numbers and print biggest number.

```
int a= SC.nextInt();
int b= SC.nextInt();
int big=(a>b)?a:b
S.o.p(big);
```

Special operator:-

new --> This is allocate memory from an object dynamically

•-->It is membership operator for object. Class pkg

Bitwise operator:-

The operator used to manipulate to bits instead of values.

Application: Low level programming.

List of Bitwise operator:-

& → bit AND

| → bitwise OR

^ → bitwise XOR

<< → Left Shift

>> → Right Shift

~ → ones compliment

```
int a=5 b=4;
S.o.p(a&b);//4
S.o.p(a|b);//5
S.o.p(a^b);//1
```

Control Statement:-

There are two types of Control statements

1) Condition
2) Iterative

Conditional Control Statement:-

1. If
2. Switch Case

If:- It's widly used for Control Statement. If Contain several forms.

1. Simple if
2. multiconditional } Wid
3. Nested if : if....if
4. block if: if

Simple if:-

```
if (condition)
{
.
.
}
else
{
.
.
}
```

Write a program to read two numbers and print biggest number

```
import java.util.Scanner;
Class Number1
{
psvm()
{
Scanner sc=new Scanner(system.in);
S.O.P("Enter the Numbers of A & B");
int a= Sc.nextInt();
int b= Sc.nextInt();
if(a>b)
S.O.P("a is the biggest Number");
else
S.O.P("B is the biggest Number");
}}
```

WAP to read a number and test for Even or odd Number.

```
import Java.util.Scanner;
Class Test
{
psvm( )
{
Scanner Sc=new Scanner(System.in);
S.O.P("Enter Number");
int n=sc.nextInt();
if(n%2==0)
S.O.P("Even number");
else
S.O.P("Odd Number");
}
}
```

Write a program to read year and test leap year or not?

```
import java.util.Scanner;
Class Test
{
PSVM()
{ Scanner sc=new Scanner(system.in);
S.O.P("Enter year");
int year=sc.nextInt();
if(year%4==0)
S.O.P("This is leap year");
else
System.out.println("This not leap year");
}
}
```

Write a program a Number and test it common multiple of 3 & 5 (use &&)

```
System.out.println("Enter number");
int n=sc.nextInt();
if(n%3==0)&&(n%5==0)
S.O.P("It is common multiple");
else
S.O.P("It is not common multiple");
```

MultiConditional and if Control Statement :-

If else if

Syntax :-

```
if (condition)
{
}
if else.
```

Write a program to read three numbers and print biggest Number.

```
import java.util.Scanner;
Class Test
{
P S V M (String args[])
{
Scanner SC=new Scanner(System.in);
S.O.P("Enter three numbers");
int a= SC.nextInt();
int b= SC.nextInt();
int c= SC.nextInt();
if (a>b && a>c)
S.O.P("a is the biggest Number");
else if (b>c)
S.O.P("b is the biggest number");
else
S.O.P("C is the biggest number");
}
}
```

* Student marks and assign/print grade based on marks range
  - 91-100 → A
  - 81-90 → B
  - 71-80 → C
  - Otherwise → P

```
import java.util.Scanner;
class A
{
P S V M (String args[])
{
Scanner SC= new Scanner (System.in);
S.O.P("Enter Student marks");
int marks= SC.nextInt();
if (marks >=91 && marks<=100)
S.O.P("grade is A");
else if (marks>=81 && marks<=90)
S.O.P("grade is B");
else if (marks>=71 && marks<=80)
S.O.P("grade is C");
else
S.O.P("grade is P");
}
}
```

ASCII:-

→ It stand for American Standard code for Information Exchange.

Char	ASCII
A-Z	65-90
a-z	97-122
0-9	48-57

Other than this Special Characters

```
class Test
{
p s vm (String args[])
 char ch='%';
if (ch>=65 && ch<=90)
S.O.P ("Upper case alpha");
else if (ch>=97 && ch<=122)
S.O.P ("Lower case alpha");
else if (ch>=48 && ch<= 57)
S.O.P ("digit");
else
System.out.println(" Special Character");
```

Nested if:-

If with in another if are followed by another if

Syntax:-

```
If (condition) => outer if
{
If (Condition) => inner if
{
}
.
}
else
}
}
```

Write a program to read a number even or add between 1 to 10.

```
import java.util.Scanner;
Class Test
{
p.s.v.m (String args[])
{
Scanner sc=new Scanner (System.in);
S.O.P (" Enter number");
int n= sc.nextInt();
if (n>=1 && n<=10)
{
if(n% 2==0)
S.O.P ("It is Even")
else
S.O.P ("It is odd")
else
S.O.P ("It is not in range");
}
}
```

Block if:-

It is if control statement without Else keyword

```
if(condition)
{
}
```

Switch Case

It is an improvement of if else if to test for multiple conditions.

It is provided with break statement.

Usage :-

```
Switch (varname)
{
case val1:
        break;
case val2:
        break;

default:
}
```

Advantage :-

→Fast Execution with break statement.

Example :-

WAP to read a number and print in words with in brackets 1 to 5.

```
import java.util.Scanner;
class Test
{
P S V m (String args[])
{
Scanner sc = new Scanner (System.in);
System.out.println ("Enter Number");
int n= sc.nextInt();
Switch (n)
{
  case 1: S.O.P ("one");
  break;
  case 2: S.O.p ("two");
  break;
  case 3: S.O.p ("three");
  break;
  case 4: S.O.p ("four");
  break;
  case 5: S.O.p ("five");
  break;
  default: S.O.p ("Invalid number");
 }
}
}
```

Iterative Control Statements :-

These are known as Looping Control Statements used to perform repetative Execution.

Loops:

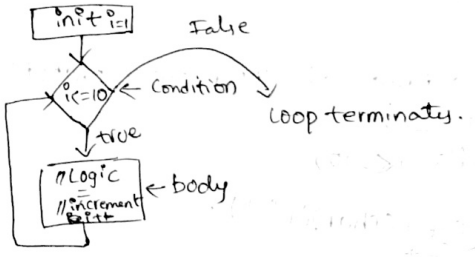
1. While
2. Do-while
3. For loop
   → Java Supports 3 forms of forloop
   1. simple for
   2. Extended for
   3. Labeled for

While Loop:-

→ It Executes as long as condition is true

→ It terminate the loop if the Condition is false.

Flow Chart :-



Syntax:-

```
while (condition)
{
;
;
}
```

WAP to print all the numbers from 1 to 100

```
class Test
{
psvm (String args[])
{
 int i=1;
 while (i<=100);
{
 System.out.println(i);
 i++
}
}
}
```

WAP to print all the numbers from 10 to 1.

```
int i=10
while (i>=1)
{
S.O.P(i); // S.O.P(i+", ")
i--;
}
```

WAP to print all uppercase alphabets from A to Z

```
int i=65;
while (i<=90)
{
S.O.P((char)i+" ");
} i++;
```

WAP to print only even numbers from 1 to 40

```
int n=1;
while (n<=40)
{
 if (n%2==0)
   S.O.P(n);
 n++;
}
```

Do While:-

-> It Executes body first and then condition gets checked.

Syntax:-

```
do
{
;
;
} while (condition)
```

Ex:- Print 1 to 10 numbers

```
int n=1
do
{
S.O.P(n);
n++;
}
while (n<=10);
```

Output:-

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

-> With wrong initialization loop will iterate at once.

For Loop:-

→ It is a user friendly loop (Syntax wise)

Syntax:-

```
for (init, cond, incre)
 {
 }
```

WAP to print first 50 numbers in reverse order:

```
for (int i=50; i>=1; i--)
 {
 S.O.P(i+'\t');
 }
```

WAP to print a multiplication table upto 10 terms by reading the number.

```
import java.util.Scanner;
Class Test
{
 P S V M ( )
 {
  Scanner SC= new Scanner (System.in);
  S.O.P(" Enter Number");
  int N=SC.nextInt( );
  for (int i=1;i<=10;i++)
   {
     int r=N*i;
     S.O.P(N+"X"+i+"="+r);
   }
 }
}
```

Nested Loops

-A Loop inside of another loop one loop is known as outer loop and other loop is known as inner loop.

Usage:-

```
for (i=1;i<=10;i++)   // outer (rows)
 {
 for (j=1;j<=5;j++) // inner (column)
 {

 }
}
```

In this program for 1 iteration of outer loop Entire iterations of inner loop will be taken

→ These types are used in Matrix, table (Pattern)

WAP to print the following Series

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

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

WAP to print

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

```
for (int i=1 ; i<=5 ; i++)
  {
  for(int j=1;j<=i;j++)
     {
      S.o.p(j)
     }
   s.o.p("\n");
  }
```

WAP to print

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

```
for (int i=1; i<=5; i++)
  {
   for (int j=1; j<=i; j++)
    {
     S.o.p ("* ") ;
    }
   s.o.p("\n");
  }
```

Jumping Statements :-

1. break
2. continue
3. System.Exit(0);
   -> This terminates the main program abruptly

Null Keyword :-

-> It's a keyword / reserved word represents nothing (no value).

Syntax :-

```
String Comment = null ;
int    Score = null;
```

Exception handling :-

-> It is known as runtime Error generated by Successfull application.

Error overview :-

by developing application these are two types of Errors get generated

1. Compile time Error
2. Run time Errors.

1. Compile time Error :-

All Syntax Errors

Ex :-

-> ; , " " , ' ' , } Missing

-> Usage of undeclared variables.

-> Usage of mismatched reserved words

-> Usage of invalid program structure etc.,

2. Run time Error :-

It is an error generated by successfull program at client Environment with lack of resources.

-> File Non Existing files.

-> No RAM, printer, Modem, etc.,

-> Invalid links.

Eg:- divide by zero

Note:-
Runtime Error cause abrupt termination

Need of exception handling :-
to avoid abrupt termination Exception to be handled.

Implementation requirements:-
-> try block } to handle Exception
-> catch block }

Syntax of try & catch :-

```
try
{
:
:
}
catch( )
{
:
:
}
```

Features :- (try)
-> Try is an Excutable block
-> It Should be followed by Catch block.
-> try throws out genetated Error.

Catch :-
-> This knowns are Error handler block
-> It Execules on Error occurance only

Exceptions :-
-> Java.xxxx.xxxxException :-
(Inner package, Some primary name)
-> Java uses a class for Every Exception.

	Exception Name	Error Cause
1.	Java.lang.Arithmatic Exception	divided by '0'
2.	Java.lang.Number format Exception	unable to Convert String to Number
3.	Java.util.inputmismatch Exception	Entering String instead of integer.
4.	Java.lang.Arrayinderout of bound Exception	on using unused inder
5.	Java.lang.Stringinderout of bounds Exception	on using unused inder

1. one try with one catch block :-
* write a program to read your marks, print the same, by handling Exception.

```
import java.util.Scanner;
Class Test
{
 P S V M ( )
 {
  Scanner sc=new Scanner(System.in);
 try
  {
    S.O.P("Enter marks");
    int marks=sc.nextInt( )
   S.O.P(Marks);
 }
```

```
Catch(InputMismatchException im)
{
S.O.P("Invalid Number");
}
}
}
```

1 try with Multiple Catch block:-

→ This is prefer when there is a chance of getting more than one exception

Syntax:-

```
try
{
}
catch()
{
}
catch()
{
}
```

* WAP to read 2 integer from keyboard divide one by other than display the result.

```
import java.util.Scanner;
import java.util.InputMismatchException;
class Kejiya
{
PSVM()
{
Scanner SC= new Scanner(System.in);
try
{
S.O.P("Enter 2 numbers");
int a,b;
a= SC.nextInt();
b= SC.nextInt();
int res= a/b;
S.O.P(res);
}
Catch(ArithmaticException ae)
{
S.O.P(" Cannot divide by zero");
}
Catch(InputMismatchException im)
{
S.O.P("Invalid Number");
}
}
}
```

Error handlers:-

→ The statements those defining catch block called Error handlers

They are two types

1. User defined
2. pre defined

1. User defined:-

This doesn't Contain Error debug machanism

Ex:-

```
Catch(FilenotfoundException fe)
{
S.O.P("no such file")
}
```

2. Pre-defined Error handlers)

This is for developer sake that Contains Error debug machanism

Usage:-

```
Catch(Arithmatic Exception Ae)
{
Ae.printStackTrace();
S.O.P(ae.getmessage) → only message
}
```

Ex:- WAP to read your name, position and print positioned character

```
import java.util.Scanner;
import java.util.InputMismatchException;
import java.util.StringIndexOutOfBoundsException;
class Test
{
psvm()
{
Scanner sc=new Scanner(system.in)
try
{
s.o.p("Enter String");
String st=sc.next();
s.o.p("Enter position");
int p=sc.nextInt();
s.o.p(st.charAt(p));
}
Catch(StringIndexOutOfBoundsException ae)
{
ae.printStackTrace();
}
Catch(InputMismatchException im)
{
s.o.p(im.getmessage());
}
}
}
```

Types of Catch blocks:-

->There are two types of Catch block

1. Known
2. default

Known:-

->The Catch block that can handle only one type of Exception

Default:-

->The Catch block that handles all types of Exceptions and it replaces the usage of no of known catch blocks

Syntax:-

```
Catch(Exception ex)
{
:
}
```

java.lang.Exception:-

It acts as a parent class for all known Exception

Multiple try(s) with Multiple Catch(s):-

->This is prefered in a huge application that contains many logics few may raise Exceptions and few may not raise Exceptions.

Usage:-

```
//logic1
try
{
s.o.p(10/0);
}
Catch(Exception ex)
{
s.o.p("Cannot divide by zero");
}
//logic 2
s.o.p("Hello world!");
//logic 3
try
{
:
}
Catch(...)
{
:
}
```

Types of pre defined Exception:-

1. Predefined Exception
2. User defined Exception

Predefined Exception:-

-> Checked Exception
-> Unchecked Exception

* Checked Exception:-

-> This is an Exception provided with built-in handling machanism.

Ex:- java.lang.ArithmaticException

* UN-Checked Exception:-

-> This Exception's Compulsarly to be handled by developer

Handling of UN-checked Exceptions:-

1. try, catch
2. throws (clause)
3. -> It's a clause used to offer handling machanism to ~~user defi~~ Un-checked Exception

Syntax:-

```
PSVm(...) throws Exception
{
.
.
}
```

3. Finally

-> It's a Special block that Executes always with Error & without Error. It should be defined after Catch block.

Usage:-

```
try
{
.
}
Catch
{
.
}
finally
{
}
```

String handling / String logic:-

-> String is a Collection of text identified in the form of words & sentences

Manipulations on the String:-

```
String st1 = "java";
String st2 = "SERVER";
S.O.p(st1.toUpperCase());
S.O.p(st2.toLowerCase());
S.O.p(st1.length());
```

String Manipulation with operators:-

'+' -> Concatination Sake.

$st_3 = st_1 + st_2$;

'=' -> To assign the String

$st_1 = st_2$; // copy.

'==' -> Compare two strings for Equality if the values are Static. It can't Compare.

String Comparision:-

1. st_1.Equals(st_2) <t, f> -> It Considers the case
2. st1.EqualsIgnoreCase(st_2)

Example:-

WAP to read two Strings and Compare them for Equality.

```
import java.util.Scanner;
Class Test
{
 P S V M ( )
 {
  Scanner sc= new Scanner (System.in);
  S.O.P(" Enter two strings");
  String st1, st2;
  st1 = sc.next();
  st2 = sc.next();
  if (st1.Equals(st2)=true)
    S.O.P("Same");
  else
    S.O.P("different");
 }
}
```

3. st1.Compareto(st2)
   - ==0
   - >0
   - <0

st1==st2 → 0
st1 > st2 → >0
st1 < st2 → <0

→ This method compares both the strings and returns as 0, >0, <0.

*WAp to read two strings and print biggest String.

```
import java.util.Scanner
class Test
{
 P S V M ( )
 {
  Scanner Sc= new Scanner (System.in);
  S.O.P("Enter two strings");
  String st1, st2;
  st1 = sc.next();
  st2 = sc.next();
  if(st.compareTo(st2)>0)
   { S.O.P(st1);
  }else
  {
   S.O.P(st2);
  }
 }
}
```

Partial string comparision:-

1. st.endsWith(pat); <t f
2. st.StartsWith(pat); <t f
3. st.indexOf(pat); -P -1

→ It returns its position if found otherwise -1.

WAp to read email id print as valid id if it Ends with .com and it must contain @ character.

```
import java.util.Scanner;
Class Test
{
 P S V M ( )
 {
  Scanner sc= new Scanner (System.in);
 System.out.println("Enter Email");
  String st= sc.next();
  if(st.endsWith(".com"))
  {
  S.O.P ();
  }
```

```
else
{
S.o.p("missing.com");
}
if(st.indexOf('@')>0)
{
s.o.p("valid email");
}
else
{
s.o.p("@ missing");
}
}
}
```

Partial String / Sub String :-

* st.charAt(i);
  -> It returns the character at given index.
* st.substring(p1, p2);
  -> It returns partial String between p1 & p2 by Excluding p2.

Ex:-

```
st="welcome";
st.charAt(0);
st.substring(0,3);
```

WAP to print one by one character of your name

```
Class Test
{
Public Void Static main(String args[])
{
String name="Hafiya";
for(int i=0; i<name.length(); i++)
System.out.println(name.charAt(i));
System.out.println(name.substring(0,3));
}
}
```

String replace :-

```
st.replace(oldstr, newstr);
```

Ex:-

```
String st='jack';
S.o.p(st.replace('j','b'));
S.o.p(st.replace("j","b"));
```

String tokenisation :-

The process of splitting a string into small pieces (tokens) with any character as seperator.

Ex:-

```
String st="Word wide web"
String words[]=st.split(" ");
S.o.p(words.length);
```

String reverse :-

java.lang.StringBuffer/Builder :-

-> This object represents varied String. ~~where~~

Manipulations of StringBuffer :-

1. append()
2. delete()
3. insert()
4. reverse()

WAP to show all varied manipulations of SB

```
Class Test
{
 Psvm()
{
 StringBuffer sb1=new StringBuffer ("Web");
 S.O.p(sb1); //Web
 sb1.append ("server");
 S.O.p(sb1); //Web server
 sb1.insert (3,"logic");
 S.O.p(sb1); //weblogic server
 sb1.delete (0,3);
 S.O.p (sb1); //Logic server
 sb1.reverse();
 S.O.p (sb1); // revrescigOL
}
}
```

Math Logic :-

java.lang.Math :-

→ This class contains several mathematical methods.

1. Math.pi

```
Class Test
{
 Psvm()
 {
 S.O.p(Math.PI)    Output :- 3.14.
 }
```

2. Math.e

```
Class Test
{
 Psvm()
 {
 S.O.p(Math.e)    Output 2.71
 }
}
```

3. Math.Sqrt(value);
4. Math.abs(value);
5. Math.pow(value);

```
Class Test
{
 Psvm()
 {
 S.O.p(Math.e);
 S.O.p(Math.sqrt(36)); //6
 S.O.p(Math.abs(-10)); //10
 S.O.p(Math.pow(2,5)); //32
 }
```

6. Math.sin()
7. Math.Sinh()
8. Math.log()
9. Math.exp()
10. Math.asin()

Date logic :-

java.util.Date :-

→ This class object represents Current date & time

Usage :-

import java.util.Date;

→ Create object.

```
Date dt=new Date(); // Current dt & time
```

WAP to print Current date & time.

```
import java.util.Date;
class Test
{
 P S vm ( )
 {
 Date dt=new Date();
 System.out.println(dt.toString());
 }
}
```

Get attributes of the Date :-

1. dt.getDate() → dd(1-31)
2. dt.getMonth() → mm (0-11)
3. dt.getYear() → YYYY
4. dt.getHours() → 0-23-HH
5. dt.getMinutes() → (0-59)mi
6. dt.getSeconds() → SS (0-59)

Formatting Date Content :-

java.text.SimpleDateFormat ;

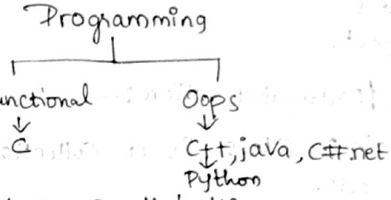
→ This class object converts java's date to any required date format

Ex :-

WAP to print Current Date in Oracles date format

```
import java.util.Date;
import java.util.SimpleDateFormat;
Class Test
{
 Ps vm ( )
 {
  Date dt = new Date();
  Date
  SimpleFormat sdf = new SimpleDateFormat("dd-MMM-yy");
  S.o.p(sdf.format(dt));
 }
}
```

Object oriented programming :-



→ ada, Simula 97, Small taks

Java overcomes the drawbacks in these programming Languages.

→ VC++ < oops / GUI
→ java < oops / web / java
→ C#.Net
→ Python

Overview of oops :-

→ Three features are known as Oops Concepts

1. Encapsulation – Class, Objects
2. Polymorphism – Overloading
3. Inheritance – Overriding

Encapsulation :-

→ The process of hiding and belonging ness of the data to object level.

Implementation requirements :-

1. Class
2. object.

* Class / Data Abstraction :-

→ A class is a collection of data members & methods.

Ex :- Symbolic representation of Class :-

Login

id, pwd	} data member (storage)
isEmpty(); Submit();	} methods (logic)

→ class is an abstract (whose object is to be created)

→ types of classes :-

1. Simple Class
2. Static class
3. Super class
4. Sub class
5. Abstract class
6. Final class.

How to define class :-

```
class xxxx
{
  // data members
  :
  // data methoddefs
  :
}
```

Define a class for Sqr pattern

```
class sqr
{
 int side=90;
 void area()
 {
 S.O.P( side*side);
 }
}
```

Object :-

→ It is an instance of the class

```
class_name obj1, obj2;
```

CREATION OF THE OBJECTS :-

1) 2 step Construction :-

→ declare obj

```
Sqr S1;
```

→ allocate memory

```
S1 = new Sqr(); //dynamic obj instance.
```

2) 1 Step Construction :-

```
Sqr S1= new Sqr();
```

* Access the members of Object.

with . (belonging Operator)

Define a class for circle pattern with necessary attributes and logic.

```
class circle
{
 int Radius=6;
 Void area()
 {
  S.O.P(3.14*Radius*Radius);
 }
}
Class Test
{
 Public Static Void main(String args[])
 {
  Circle c1=new Circle();
    c1.circle()
 }
}
```

Scope modifier / Data-hiding :-

-> Every member of the class should be defined with access Scope modifier.

Types of Scope Modifiers :-

1. private
2. Default
3. public
4. protected

1. Private :-

It is accesible in self class it will not be accesassible at out side of the class.

2. Default :-

-> It's a default Scope modifier
-> No such reserved word.
-> It is accesible to
   -> self class
   -> sub class
   -> other class
   } of same path (or) package

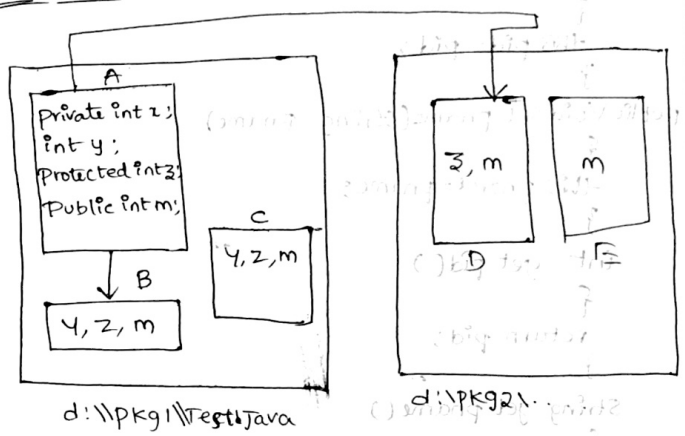
3. protected :-

-> It is accesible to same packaged classes, sub classes of other packages.

4. Public :-

-> It is accesible to any where that is same packaged classes & other packaged classes.

Data-hiding :-



Data Hiding :-

-> The process of hiding data members and accessing these data members with public method.

-> Declare private data member.

Private int id;

-> Manipulate with Setter & Getter methods :-

Setter :-

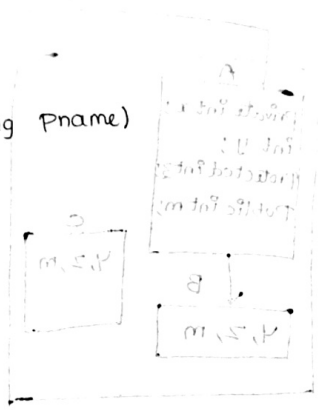
-> This is to assign value to private data member of a class.

Getter:-

→ This is to get the value from private data member of a class.

Ex:-

```
Class product
{
 private int pid;
 Private String pname;
Public Void setpid (int pid)
 {
  this.pid = pid;
 }
Public Void set pname(string pname)
 {
  this.pname = pname;
 }
 int get pid()
 {
  return pid;
 }
 String get pname()
 {
  return pname;
 }}} class Test {
 Public static Void main (string args[])
 {
  product p1 = new product();
  p1.setpid(1212);
  p1.setpname("mobile");
 }
}
```



Logic methods:-

→ These methods include business logic

1. Math/computational
2. Validation
3. Business logic
4. db logic
5. file logic

...etc.,

Define a class for a login task with id, pwd as datamembers and a validation method

id → System, pwd → manager

```
Class login
{
 Private String id, pwd;
 Public Void setId(String id)
 {
  this.id = id;
 }
 Void setpwd(String pwd)
 {
  this.pwd = pwd;
 }
 Void validate()
 {
  if (id.equals("System") && pwd Equals("Manager"))
 {
 S.o.p("Success");
 }else
 {
  S.o.p("unsuccessful");
 }
}
Class Test
{
```

```
psvm()
{
Login u=new Login();
u.setid="admin";
u.setpwd="Manager";
u.validate();
}
}
```

*WAP to define a class for simple interest computation.

Constructer:-

-> It's a member function, that Executes when an Object is created.

Need of Constructer:-

-> This is to perform first time/default/ initialization task.

Definition rules of the Constructor:-

-> Class name & Constructer name should be same
-> It never return value
-> It can be overloaded (more than one Constructer is allowed).
-> It never be called with . notation.
-> It gets Executed at the time of object creation.

Types of constructer:-

1. Default Constructer
2. parametrized Constructer
3. Copy Constructer.

1. Default Constructer:-

-> It's a Constructer member function without arguments.

-> used to initialize an object with default properties.

Exp:-

```
class Rect
{
private int l,b;
Rect()
{
l=10;
b=20;
}
}
```

Call default Constructer.

```
Rect r1=new Rect();
```

2. parameterized Constructor:-

-> It's a Constructer member function defined with arguments/parameters.

-> used to initialize an object with client preferences (parameters.

3. Copy Constructer :-

→ It's a constructer member function defined with Self class object as argument.

→ Used to copy new object with Existing object properties.

* Define a class with 3 forms of Constructers.

```
Class Button
{
private int width, height;
// def constructer
Button ( )
{
 this.width = 100;
 this.height = 20;
}
// paramiterized constructer.
Button (int w, int h)
{
 this.w = width;
 this.h = height;
}
Button (Button Obj)
{
this.w = obj.w;
this.h = obj.h;
}
void Showsize()
{
 S.O.P( w+","+h);
}
}
Class Test
{
 P S V m ( )
{
 Button b1 = new Button ();
 Button b2 = new Button(200,40);
 Button b3 = new Button (b2);
 b1. Show size ( );
 b2. Showsize ();
 b3. Showsize ();
}
}
```

Note :-

→ Java System maintain an implicit default Constructer in a class in which no explicit Constructer definition.

Finalizer :-

It's a member function that executes when an object is destroyed.

Need :-

→ To perform finalyzation task.

Ex :- 1. closing files
2. Resource releasing
3. disconnection.

→ Java doesnot support Explicit finalyzer.

Java.lang.object :-

→ It acts as an invisible Superclass for all Java's classes.

Nested Class:-

→ A class inside of another class.
→ One class is known as outer class and other class is known as inner class.

Usage:-

```
class class-name ← outer
{
    ≡≡≡
    class class-name - Inner.
    {
    }
}
```

Features of Nested class:-

1. Outer class members are refered into inner class.
2. Outer class can refer innerclass members in the form of object only.
3. Client application can create outer class object and it cannot create inner class object.
4. Inner classes are also known as hidden class.

Container Class:-

→ The class that contains object as data members of other classes.

Eg:-

```
Class Book
{
    :
}
Class member
{
}:
Class issues // Container
{
    Book b1 = new Book();
    member m1 = new Member();
    ≡
}
```

Functional Modifier:-

→ This modifier represents belongingness. The members of the class are defined with functional modifier.
There are 2 types of functional modifiers:-

1. Static
2. Non-static.

Non Static:-

→ It's a default modifier. NO such reserved word.
→ It represents object belonging.

Static:-

→ It's a class belonging modifier that doesn't require object.

Eg:-

```
class A
{
 void fun1()  //Non-static
  {
  S.O.P("Obj call");
  }
 static void fun2()  // static
  {
  S.O.P("Class call");
  }
}
class Test
{
 P S V M ()
  A.fun2();
} new A().fun1(); //anonomous object
```

Collections :-

1. Arrays :-

* Array is a collection of Elements.

* Single dimensional array :-

int a=5, b=10, c=20

dt array-name[ ] = {v1, v2 ...} → array
↓ size, ↓ elements

Ex :-

int a[] = {1, 2, 3, 4, 5}
-5 -4 -3 -2 -1
0 1 2 3 4

S.O.p("a[0]);

* Two dimensional array :-

dt array_name [ ][ ] = {{v1, v2}, {v3, v4}}
↓ row ↓ col

Ex :-

int a[2][2] = {{11, 22}, {33, 44}};

S.O.p (a[0][1]);

O/p :- 22.

* Three dimensional array :-

dt array_name [ ][ ][ ] = {{{1, 2, 3, 4}, {v5, v6, v7, v8}}, {{v9, v10, v11, v12}, {v13, v14, v15, v16}}}
↓ table ↓ row ↓ col

Ex :-

int a[2][2][2] = {{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}};

S.O.p (a[0][1][1]);

* Arrays are used to store multiple values in a single variable, instead of declaring seperate variable for each value.

Ex :-

```
Class me
{
 p s v m ( )
 {
  int a[] = {11,22,33,44,55};
  8. for (int i : a) // for-each :-
   S.O.p (i);
 }
}
```

O/p :- 11
22
33
44
55

Length properties :-

```
Class Me
{
 p s v m ( )
 {
  int a[] = {11, 22, 33};
  S.O.p (a.length);
 }
}
```

O/p :- 3.

Ex :-

```
Class me
{
 p s v m ( )
 {
  String n[] = {"K", "S", "V", "J"};
  names a[]
  n[1] = "R";
// S.O.p (n);
  for (String s : n)
   S.O.p (s)
 }
}
```

Polymorphism:-

ability of an object that shows more than one form

Types:-

There are two types

1. Static polymorphism
2. Dynamic polymorphism.

1. Static polymorphism:-

-> An object that shows more than one form at compile time.

Implementation requirements:-

1. Operator overloading
2. method overloading.

* Operator overloading:-

In Operator that performs more than one task known as operator overloading (without change of meaning).

Ex:-

+ < add / concatination

. - belongingness (package/class/obj) ; java.lang System.out.println

Note:-

Java doesnot support userdefined opertor overloading.

* Method overloading:-

A method with more than one definition with common name to perform a similar task in different ways.

Definition rules:-

- Must contain more than one definition
- Every definition should be with same name
- Few methods cannot be overloaded

  example: Finalyzer.
- Every definition should differ in type ~~of~~ args ~~or~~ no ~~of args~~

Define a Class with an overloaded method that returns ~~max~~ sum ~~of~~ two nubers, three numbers.

```
Class A
{
 Static int sum(int a, int b)
 {
  return (a+b);
 }
 Static int sum(int a,int b,int c)
 {
  return (a+b+c);
 }
}
 Class Test
{
  P S V M ( )
{
 S.O.P( A.sum(10,20));
 S.O.P( A.sum(10,20,30));
}
}
```

Inheritance :-

Ability of an object that shares the properties of existing object

Example :-

* Need :-
  → reusablity
* Implementation req's :-
  → Super class
  → Sub class

Super class :-
→ It is an independent class whose members are reffered by subclasses.

Sub class :-
→ It is a dependent class that shares the members of Super class

Types :-
1. Single inheritance
2. hirarchical inheritance
3. Multilevel inheritance
4. Multiple inheritance
5. ~~hirarchi~~ Hybrid inheritance.

Single inheritance :-

→ A Super class with sub class

* Define Super class

```
class A
{
.
.
.
}
```

* Define Sub class

```
class B extends A
{
.
.
.
}
```

Extends :-
This clause is used to share all the members of Super class Except private members.

* Sub class object

```
B b1=new B();
b1.xxxx();
```

Define following class hirarchy in single inheritance.


```
class C1
{
 void fun1()
 {
```

```
System.out.println("It is super class method");
}
}
class C2 extends C1
{
  void fun2()
  {
System.out.println("It is sub class method");
  }
}
class Test
{
 P S V m()
 {
  C2 obj1=new C2();
  obj1.fun1();
  obj1.fun2();
 }
}
```

Considerations in inheritated classes:-

1. Method overriding
2. Constructer overriding

Method overriding:-

-> The process of redefining superclass methods in sub classes

* **Need:-**

-> This is to impose custom logic by overriding default logic

* Define following class hirarchy with overridden method


```
class A
{
 void init()
 {
 S.O.P("Company logic");
 }
}
Class B extends A
{
 void init()
 {super.init();
 S.O.P("Custom logic");
 }
}
Class Test
{
 P S V m()
 {super.
  B B1=new B();
  B1.init();
 }
}
```

Super keyword:-

It's a reserved word that represents super class reference. and it's should be used in sub class only.

Final modifier:-

-> It's a modifier for method, class, datamembe

Final method:-

The method that cannot be overridden. It contains only one definition

```
Class A
{
 final void fun1()
    {
    :
    }
}
class B
{
  void fun1() // It Shows Error.
   {
   :
   }
}
```

Final class:-

->The class that cannot be inherited. that is it doesnot contain Subclasses.

```
final class A
 {
 :
 }
class B Extends A // It gives Error.
 {
 :
 }
```

Final Datamember:-

It acts as Constant whose value cannot be changed. It should be declared with final modifier.

-> It should be initialized.

Ex:-

```
class Server
{
 final int userCount=100;
 :
 :
}
```

Constructer Overriding:-

The process of invoking Super class Constructer from Sub class Constructer

Need:-

-> on instantiating Sub class object SUb class Constructer only get Executed and Super class Constructer doesnot Executed this results in partial initialization.

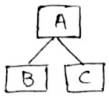
Ex:-

```
class A
{
 int x;
 A()
 {
 x=100;
 }
 A(int x)
 {
 this.x=x;
 }
}
Class B Extends A
{
 int y;
 B()
 {
  Super()
 y=200;
}
```

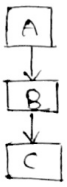
```
B(int x)
{
 super(x);
 this.y=y;
}
int sum()
{
S.O.println(x+y);
}
}
Class Test
{
P S VM ()
{
B b1= new B();
B b2= new B(40,70);
b1.sum();
b2.sum();
}
}
```

Hirarchical inheritance:-

A Super class with more than one Subclass



Multilevel inheritance:-



Single inheritance at different levels.

Interfaces:-

Interface is like a class but that cannot be instantiated

Need of interfaces:-

1. to build dynamic polymorphism.
2. to build multiple inheritance.

Implementation:-

1. define interface
2. define subclass
3. invoke methods of interface

&

Define interface:-

-> It's a collection of datamembers, methods.
-> Interface datamembers are final constants.
-> Interface methods are abstract/pure method.

Usage:-

```
Interface I1
{
 final int x=90; //final is optional.
 void draw(); //abstract method.
}
```

Interface method:-

-> They are in the form of abstract methods whose definition to be done subclass.

Note:-

Interface members are defined with a default scope that is public.

*Define Subclass:-

Subclass is essential for interface it only:
offers behaviour to interface.
-> It must override all methods of interface.

Usage:-

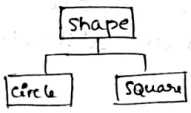
```
class class_name implements interface_name
{
:
}
```

*Invoke methods of interface:-

Sub class object invokes the methods of interface.

Example:-

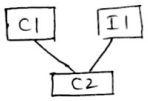
Define following class thirarchy.




```
interface Shape
{
 void draw();
}
class circle implements Shape
{
 public void draw()
 {
  S.O.p ("circle pattern");
 }
}
Class Square implements Shape
{
 public void draw()
 {
  S.O.p ("Square pattern");
 }
}
Class Test
{
 p s v m ()
 {
  Shape S; //ref
  S= new Square ();
  S.draw();
  S= new circle ();
  S.draw();
 }
}
```

Multiple inheritance:-

->

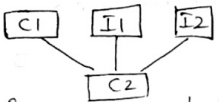


A subclass from one Super class and one interface.

Syntax:-

```
class C2 extends C1 implements I1
{
 : //over ride interface methods
 :
}
```

->



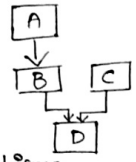
Super class is only one but many interfaces.

Ex: class C2 extends C1 implements I1, I2

```
{
}
```

Hybrid inheritance:-

Combination of any two types of inheritance



Dynamic polymorphism:-

Ability of an Object that shows more than form at run time.

Example:-
-> tool box
-> Exception.

Implementation requirements:-

1. interfaces
2. abstract class

Abstract class:-

It acts as a Superclass but that cannot be instantiated.

-> It is the combination of Superclass and interface

Superclass	Interface	Abstract class
1. Concrete method (definition/method body)	1. Contains only abstract methods.	1. Combination of abstract methods & Concrete methods
2. Whose Object can be created	2. Whose object cannot be created	2. Whose object cannot be created
3. Defined with		3. Defined with abstract modifier.

```
abstract class C1
{
Void fun1()  // Concrete method
 {
 }
abstract Void fun2(); //abstract method.
}
```

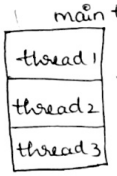
-> sub class is required for abstract class.

```
class C2 Extends C1
{
  //override abstract methods
}
```

Multithreading:-

The process of subtasking a major task into no. of subtasks that is known as Multithreading

Example:-



Applications:-

-> Concurrent services

Thread libraries:-

1. Java.lang.Thread:

-> It acts as a Super Class for all the threads. it contains an overridable logic method run():

Write a program to implement two thread objects.

```
Class A extends Thread
{
 Public void run()
  {
  for(int i=1; i<=100; i++)
   {
    S.O.P("I="+i);
   }
  }
}
Class B extends thread
 {
 public void run()
  {
  for(int j=1; j<=100; j++)
   {
   S.O.P("J="+j);
   }
  }
}
Class Test
{
 P S V M( )
 {
   A t1 = new A();
   B t2 = new B();
   t1.start();
   t2.start();
  }
}
```