JAVA PROGRAMMING

Procedure oriented programming	Object oriented programming
1) It is an "algorithm" type.	1) It is a "data" type.
2) We use functions as sub-procedures.	2) We use objects.
3) It is a top-down approach.	3) It is a bottom-up approach
4) No OOPs concept	4) Abstraction, encapsulation, polymorphism etc concepts are available.
5) Less secured	5) More secured
Ex: C, FOPRAN, PASCAL, BASIC etc	Ex: CPP, JAVA, .NET etc.

Need of Object Oriented Programming:

→ To implement abstraction, polymorphism, hiding, inheritence etc in programming to solve the real world problems

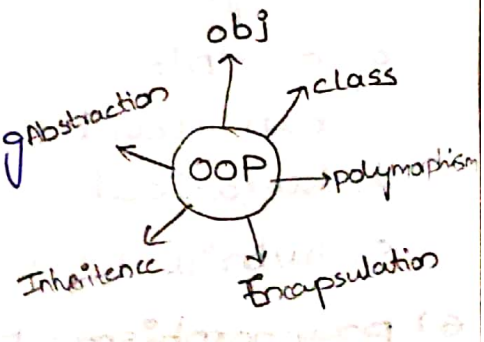
principles:

1) object - It is a real world thing (prop, Task)

Ex: student

Properties - name, R.No, age

Task - read, write, walk.



→ Object is instance of class.

(data, state, behaviour)

2) class - a template/blueprint/prototype from which objects are created (collection of variables & methods) logical entity.

Ex: class - name of class (variables, methods)

3) Abstraction: (Hiding implementation details) showing only essential parts. and implimentation details will be hidden.

Ex: Android application, app's.
APk or execcutable file will be available.

4) Encapsulation: Binding of variables & methods (code & data together) for providing security.

Ex: class, Jave bean

5) Inheritance: child class acquring properties & behaviour from parent class.

Ex: parent (Bc) → child (DC)
(Base class) → code reusability
(derived class) → runtime polymorphism (overriding)

Types of inheritance
a) single
b) multiple
c) multi-level
d) hierarical
e) hybrid

6) polymorphism: performing the task (same) in different ways many ways

Ex: add ( )
add (int x, int y)

Types of polymorphism
a) overloading (compile time)
b) overriding (run time)

OOP languages and applications:

Java, C++, python, small talk, Ruby (RUBY), PL/SQL, .NET, Common LISP, MATLAB.

Applications:

1) Real time symbo systems
2) Object orienteddata base
3) client and server communication
4) Artificial intelligence
5) Neural networks & parallel programming.

History of JAVA: James Gosling

→ 1991 → sun microsystems started "Greentalk" project for solving the issues of set-up boxes, televisions and electronic devices with the association of James Gosling.

→ After few years, he was not satisfied with Greentalk project result and he proposed the OOPL called oak to fix the bugs.

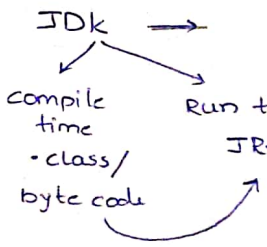
→ After lot of discussions & suggestions given by other researchers oak is renamed as 'Java'. in 1996 the first version JDk 1.0 was released Similarly 1996-1.0, 1997-1.1, 1998-1.2, 2000-1.3 (1.8 - 2014).

→ we have 3 versions of Java called
a) Standard Edition -
b) enteprice Edition -
c) Mobile Edition

→ In 2010, Oracle acquired java from sunmicrosystems

Difference b/w JDk, JRE, JVM :

JDk = JRE (JVM) + Development tools



Java - programing lang & platform

Java features :

Simple, robust, high performance, distributed, platform independent, portable, interpreted etc.
(write once, run anywhere) secured, dynamic, object oriented.

Installation of JDk :

→ Creating a folder ('class path')

No folder in Ddrive D:

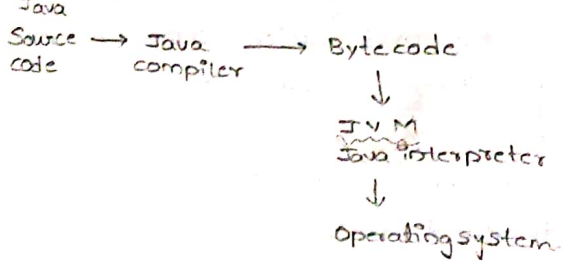
Enter MD RollNo;

CD RollNo;

D: RollNo/ javac filename.java → .class

java filename → output

JRE → JVM → output



Java basic program structure :

1) Documentation section → suggested
2) package statements
3) Import statements
4) Interface statements
5) class definitions
6) Main method class
7) Main method → Execution starts.

optionals

mandatory

Syntax :

```
Class <nameofclass>
{
  // members
  // methods
}
```

/** stmb */ → for documented comments. javadoc tool. (documentation API)

Example :

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

→ For compiling a program >> javac sample.java

execution >> java sample

output : Hello java students

Datatypes:

In java we have two categories of datatypes

1) premitive data types
2) Non-premitive data types

Premitive data types:

(i) int
(ii) float
(iii) double
(iv) char
(v) boolean

byte b; long distance;

Integer (Int): byte, short, long, int.

short s1, s2; int num;

short = 2bytes
int = 4bytes
long = 8 bytes

byte - 1 byte
short - 2bytes
int - 4bytes
long - 8 bytes

float - 4bytes
char - 2 bytes
double - 8bytes.
boolean - 1 byte

float :

float average;

Double: double temperature;

boolean: boolean check;

char: char a;

1) Write a program to print the default values of premitive datatypes. (0)

frame : defaultvalues.java

```
class defaultvalues
{
  static int i;
  static float f;
  static char k;
  static double d;
  static boolean c;
  public static void main(String args[])
  {
    System.out.println(i);
    System.out.println(f);
    system.out.println(k);
    system.out.println(d);
    system.out.println(c);
  }
}
```

Expected output:

```
0
0.0

0.0
false
```

Non-primitive data types:

(i) string - collection of characters.

Syntax: (ways to represent):

```
String S=new string ("Hello");
(or)
String s ="Hello";
(or) (Concatenation)
String S1 = "Hello" + "CSE";
(or) Assigning S1 to T
String T = new string(S1);
```

Examples:

→ class <name of class>

```
class NonPremitive
{
  public static void main(String args[])
  {
    String S = new string("Hello");
    System.out.println(S);
  }
}
```

Output : Hello

2) class new
```
{
    public static void main (string args[])
    {
        string s: "Hello" + "CSE";
        System.out.println (s);
    }
}
```

(ii) Array: collection of elements of same datatype

Ex:

1) int abc = new abc[];

Identifiers in java:

→ Identifiers should be a class name, method/variable name (or) any other label.

Rules:

→ An identifier in java should start with alphabet (A-Z) or (a-z)

→ An identifier is a combination of 0-9 digits, (A-Z)/(a-z), _, $.*

→ Identifier should not start with 0-9.

→ Java identifiers are case sensitive.

→ There's no limit in length of identifier but suggested to use (4-15) letters

→ Reserved words (keywords + literals) can't be used as identifiers.

Ex:

valid identifiers.

Reserved words int while = 20;
(It's an error)

valid identifiers: abc, _abc, ab abc123

Invalid identifier: abc xyz, abc 18, abc@

Example:

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

Identifier:

(i) sample

(ii) main (method name)

(iii) String (name of class)

(iv) args (variable name)

Naming conventions in java:

(i) package name should be in lower case.

(ii) class name should start with uppercase and should be a noun.

(iii) Interface name starts with uppercase letter & it an adjective.

(iv) Method name starts with lowercase letter & should be a verb.

(v) Variable name should start with be a lowercase letter.

(vi) Constant variable - uppercase seperated by '_'

Reserved / key words: 53 */ goto, const are not in use

int	double	continue
float	for	long
byte	while	enum
short	if	do
char	break	register..etc.

Literals: True, false, NULL

Not key words.

Operators in java:

↳ used to perform operations with operands.

→ We have no of operators in java using them in developing application (program) depends on requirement.

→ The list of operators are discussed as follows

(i) Arthematic operators: +, −, ×, %, /

(ii) Ex:
```
class sum
{
int a,b;
public static void main(String args[])
{
int a=3, b=4, c;
c = a+b;
System.out.println(c);
}
}
```
a+b
a-b
a×b
a/b
a%b

Result:
Output: 7
-1
21
0.75
3

(iii) Assignment operators: (=), +=, -=, *=)

Ex:
```
class assign
{
public static void main(String args[])
{
int a,b;
a=a+1;
System.out.println(a);
b=b-1;
System.out.println(b);
}
}
```
Expected output
4
3

(iii) Unary operator: (Single operand)
(++, --, !)

Ex:
```
class unary
{
public static void main(String args[])
{
int a=3, b=4; c;
a=a++;
c = ++a;
d = b--;
System.out.println(c);
System.out.println(d);
}}
```
Expected output:
4
4

(iv) Relational operators: {>, <, <=, >=, !=, =}

Ex:
```
class relational
{
public static void main(String args[])
{
int a=10, b=5;
boolean c, d, e;
c = a≥b;
d = a<b;
e = a!=b;
System.out.println(c);
System.out.println(d);
System.out.println(e);
}
}
```
Output:
True.
False
True.

(v) Bitwise operator: (&, |, ^, >>, <<)

```
class bit
{
  public static void main(string args[])
  {
    int a=00, b=10, c, d;
    c = a & b;
    d = a || b;
    System.out.println(c);
    System.out.println(d);
  }
}
```

output:

00
10

(vi) Conditional operator: (Ternary)

```
c = a>b ? a : b;
print(c)
```

Ex:

```
class cond
{
  public static void main(string args[])
  {
    int a,b,c; a=5, b=2, c;
    c = a>b ? a : b;
    System.out.println(c);
  }
}
```

Output:

a.

I/O streams:

java.lang.* (All classes)
java.lang (package) class
scanner class

1) System.in

Scanner class.

Scanner <nameof obj> = new Scanner (System.in);

import java.util.*
java.util.Scanner

→ To access string:

sc.next();

String kitty = sc.next(); / nextString().

→ Scanner sc = new Scanner(System.in)

int a = sc.nextInt();

float f = sc.nextFloat();

Example:

1) Area of circle.

```
import java.util.*;
class Area
{
  public static void main(string args[])
  {
    double
    int a,b;
S.O.P("Enter radius");  Scanner sc = new Scanner(System.in);
    a = sc.nextInt();
    b = 22/7 * a * a;      perimeter = 2 * 22/7 * a;
    System.out.println(b);
  }
}
```

Output: (expected)

Enter radius

1

3.14

2) Area of a triangle:

```
class Area
{
   public static void main (String args[])
   {
      int a, b;
      double ar;
      Scanner sc = new Scanner (System.in);
      a = sc.nextInt();
      b = sc.nextInt();
      ar = (1*a*b)/2;
      System.out.println(ar);
   }
}
```

Expected output:

```
Enter base
2
Enter height
1
1
```

3) Area of rectangle, perimeter of rectangle:

```
class Rectangle
{
   public static void main (String args[])
   {
      int l, b;
      double a, p;
      Scanner sc = new Scanner (System.in);
      l = sc.nextInt();
      b = sc.nextInt();
      a = l*b;
      p = 2*(l+b);
      System.out.println(a);
      S.O.P(p);
   }
}
```

Expected output

```
1
1
4
```

4. Area of square, perimeter of square.

```
class Square
{
   public static void main (String args[])
   {
      int s;
      double a, p;
      Scanner sc = new Scanner (System.in);
      s = sc.nextInt();
      a = s*s;
      p = 4*s;
      System.out.println(a);
      System.out.println(p);
   }
}
```

Expected output:

```
1
1
4
```

method : (math.sqrt)

5) Find the roots of quadratic equation. $ax^2+bx+c=0$

```
import java.lang.Math.sqrt.*;
class Roots
{
   int a, b, c;
   double s1, s2, d;
   public static void main (String args[])
   {
      Scanner sc = new Scanner(System.in);
      a = sc.nextInt();
      b = sc.nextInt();
      c = sc.nextInt();
      d = b*b - (4*a*c);
      s1 = (-b + Math.sqrt(d)) / (2*a);
      s2 = (-b - Math.sqrt(d)) / (2*a);
      S.O.P("Roots are");
      S.O.P(s1);
      S.O.P(s2);
      if(d > 0)
      {
        S.O.P("Roots are distinct");
      }
      else if (d = 0)
       {
         S.O.P("Roots are equal");
       }
      else
        S.O.P("Roots are complex");
   }
}.
```

Type casting and conversions:

(widening / up casting / automatic casting)

*1) Implicit: (i) Inbuilt conversion by compiler

(ii) Left to right direction

(iii) No loss of data.

**** char → only small to large (small → large)

*[byte → short → int → long → float → double]

2) Explicit: (i) programmer gives instruction to convert the data. [R to L]

(narrow / down casting) a chance of

(ii) There's loss of data.

char

** byte ← short ← int ← long ← float ← double (small ⇄ large)

6) Write a program on type casting

```
class Typecasting
{
  public static void main (String args[])
   {
     int x = 130;
     byte b = x;
     System.out.println(b);
   }
}
```

Output: The compiler doesn't allow the conversion.

Error. Hence, convert it explicitly.

Error: possible loss of precision

byte b = a;

required byte found int.

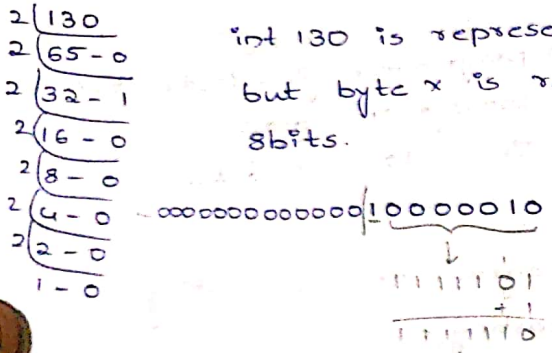
```
class Type
{
   public static void main (String args[])
   {
      int x = 130;
      byte b = x;
      b = (byte) x;
      System.out.println(b);
   }
}
```

output:

-126

The value of x is 130. but after explicit conversion from int to byte it print's '-126'. So, here, we can understand. that there's a loss of data.

Loss of data:



int 130 is represent in 32 bits but byte x is represented in 8bits.

Control statements in java:

1) if statement
2) nested if
3) if-else
4) else-if ladder
5) Switch
6) jumping statements
   (i) break
   (ii) continue.

} selection.

if statement:

```
if (condition or expression)
{
   statements;
}
```

Ex:

```
if (a>35)
{
   system.out.println ("The student cleared the exam");
}
```

2) Nested if statement:
group of if's

```
if (cond)
{  statements;
     if (condition)
     {
        statement;
          if (condition)
          {
            statements;
          }
     }
}
```

Ex:

```
if (a>0)
{  if (a<10)
   {
      if (a>=5)
      {
         S.O.P(a);
      }
   }
}
```

output:
a=
→ 1
exit if statement after 2nd if.

→ a = 6.
6.

→ a = 4
exit after 2nd statement.

3) if-else statement:

```
if (condition)
{
   statements;
}
else
 {
   stmts;
 }
```

Ex:

```
if (a>35)
 {
   S.O.P ("PASSED");
 }
else
 {
   S.O.P ("FAILED");
 }
```

4) else-if ladder:

```
if (condion)
{
   stmts;
}
else if (condition)
{
   stmts;
}
else if (condition)
{ stmts;
}
else
 { stmts;
 }
}
```

Ex: Generation of electricity bill.

```
class Ebill
{
   public static void main
                (String args[])
   {
      double units, billp=0;
      Scanner sc = new
                Scanner
                (System.in);
      S.O.P ("Enter units");
      units = sc.nextDouble();
      if (units < 100)
      {
         billp = units * 1.5;
                SOP (billp)
      } S.O.
```

```
else if (units < 300)
{
   billp = units * 5;
                S.O.P (billp)
}
else
{
   billp = units * 10;
}      "You have to pay"
S.O.P (billp);
}
}
```

Output:

```
Enter units
50
You have to pay 75
```

2) Find the largest of three numbers.

```
class Max
{
   public static void main (String args[])
   {
      int a,b,c;
      Scanner sc = new Scanner (System.in);
      a = sc.nextInt();
      b = sc.nextInt();
      c = sc.nextInt();
      if (a>b && a>c)
      {
         S.O.P ("greatest"+a);
      }
      else if (b>a && b>c)
      {
         S.O.P ("greatest", +b);
      }
      else
      {  S.O.P ("greatest", +c);
   } }
}
```

Output:

1

2

3

greatest is 3.

5) Switch statement:

character

```
Switch (variable)
{
  stmts;   | case 1 :
           | case 2 :
           |  .
           | caseN :
           | default : stmt;
}
```

Ex: Write a program to print the day in a week.

```
class Day
{
  public static void main (String args[])
  {
     char ch;
     Scanner sc = new Scanner (System.in);
     S.O.P ("Enter the number");
     int a = sc.nextInt();
     Switch(a)
     {
       case 1 : S.O.P("Monday");
                break;
       case 2 : S.O.P("Tuesday");
                break;
       case 3 : S.O.P("Wednesday");
                break;
       case 4 : S.O.P ("Thursday");
                break;
       case 5 : S.O.P ("Friday");
                break;
       case 6 : S.O.P ("saturday");
                break;
       case 7 : S.O.P("sunday");
                break;
       default : SOP ("check & re-enter the number");
       return;
     }
  }
}
```

Output:

Enter the number

1

Monday.

Enter the number

8

check & re-enter the number.

2) Write a program to create calculator using switch statement (+,-,x,/)

```
class Calc
{
  public static void main (String args[])
  {
     Scanner sc = new scanner (system.in);
     int a=
     S.O.P ("Select operator");
     char ch = sc.nextChar();   sc.next().charAt(0)
     int a = sc.nextInt();   → S.O.P ("Enter operands")
     int b= sc.nextInt();
     Switch (ch)
     {
        case '+' : S.O.P (a+b);
                   break;
        case '-' : S.O.P (a-b);
                   break;
        case '*' : S.O.P (a*b);
                   break;
```

```
        case '/' : S.O.P(a/b);
                   break;
        default : S.O.P("check Invalid operator");
    }
  }
}
```

Output:

Select operator
+
Enter operands

3

4

7.

Jumping statements:

[Break, continue]

```
for(i=0; i<=10; i++)
{
  if(i=2)
    break;
  S.O.P(i);
}
```

output:

0

1

```
for(i=0; i<=10; i++)
{
  if(i==2)
    continue;
  SO.P(i);
}
```

output:

0
1
3
4
:
10

Looping statements:

1) For loop:

```
for(initialization; condition; incrementation)
```

1) write a program to print 1 to 100

```
class Num
{
  public static void main(String args[])
  {
    for(i=1; i<=100; i++)
    {
      S.O.P(i);
    }
  }
}
```

output:

1
2
:
100

2) write a program to find the factorial of a given number.

```
class Factorial
{
  Public static void main(String args[])
  {
    Scanner sc= new scanner(System.in)
    S.O.P("Enter number");
    Int n = sc.nextInt();
    int fact = 1;
    for(i= 1; i<=n; i++)
    {
      fact= fact*i;
    }
    S.O.P("fact"+fact);
  }
}
```

Output:

Enter number

5

fact 120

3) Write a program to print even numbers from 1 to 100

```
class Even
{
    public static void main (String args[])
    {
        int i;
        for (i=1; i<=100; i++)
        {
            if (i%2==0)
            S.O.P(i);
        }
    }
}
```

Output:

2
4
:
100

4) Write a program to print odd numbers:

```
class Odd
{
    public static void main (String args[])
    {
        int i;
        for (i=1; i<=100, i++)
        {
            if (i%2 != 0)
            S.O.P(i);
    }   }   }
```

Output:

1
3
5
:
99

5) Write a program to check the given number is even or odd.

```
class Check
{
    public static void main (String args[])
    {
        Scanner sc = new Scanner (System.in);
S.O.P("enter num");
        int n = sc.nextInt();
        if (n%2==0)
            S.O.P ("Even");
        else
            S.O.P ("odd");
    }
}
```

output:
Enter num.
5
odd

While():

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

Ex:

1) Write a program to print numbers from 1 to 10

```
class Print
{
    public static void main (String args[])
    {
        int n=1;
        while (n<=10)
        {
            S.O.P(n);
            n++;
    }   }   }
```

Output:

1
2
:
10

```
while (true)
{
    S.O.P("Hello");
}
```

Infinite time run

(or)

Do while(): cod statements are executed atleast once.

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

1) Write a program to print 1 to 10.

```
class Print
{
   Public static void main (String args[])
   {
      int i=1;
      do
       {
         S.O.P(i);
         i++;
       } while (i<=10);
   }
}
```

output:

1
2
:
10

UNIT-II

Class and Objectives.

→ class is a [collection of set of variables and methods] & one of the object oriented principles.

Types of variables:

1) static variable (value remains same in the entire program)
2) Local variable (scope within that method)
3) Instance variable. (within class & outside method)

Types of Methods:

Collection of statements to perform a particular task.

Syntax of class:

```
class  Cname
{
   // variables;
   # methods;
}
```

class : a template that describes behaviour/state that object supports

Obj: block of memory, to (created.) store all instance variables

(new-operator)

Ex:

```
class Student   (Local variable)
{
   void add()
   {
      int a=10;
      S.O.P(a);
   }
Public static void main (String args[])
   {
      Student s = new student();
      s.add();
   }
}
```

characteristics of obj
1) state (data)
2) Behaviour
3) Identity. ↓ JVM to identify each obj uniquely.

output:
10

Instance variable:

Cannot be accessed directly from the main method without creating object.

Ex:

```
class Student
{
    int a = 10;
    void add()
    {
      int b = 20;
      S.O.P(b);
    }
    public static void main(String args[])
    {
      Student s = new Student();
      s.add();
      S.O.P(s.a);
    }
}
```

Output:
20
10

Static variable:

for static variable memory can be assigned only once & cannot be accessed through object.

Ex:

```
class student
{
    static int a = 30;
    public static void main(String args[])
    {
      S.O.P(a);
    }
}
```

output:
30

1) write a program on types of variables.

```
class Var
{
    int a = 10; // instance variable.
    static int b = 30; // static variable
    void public static void main(String args[])
    {
        int c = 40;
        S.O.P(b);
        Var V = new Var();
        S.O.P(V.a);
        S.O.P(c);
    }
}
```

output:
30
10
40

Method:

Syntax:

returntype nameofmethod();

→ It is a collection of statements to perform a task.

Ex:

2) write a program to find addition of two numbers

```
Class Addition
{
    public static Void main(String args[])
    {
    Void add()
    {
      int a=20, b=10, c;
      c = a+b;
      S.O.P("sum"+c);
    }
```

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

Output:

Sum: 30

Accessing multiple methods:

```
class Operation
{
  void add()
  {
    int a=5, b=5, c;
    c = a+b;
    S.O.P(c);
  }
  void sub()
  {
    int e=10, d=5, f;
    f = e-d;
    S.O.P(f);
  }
  public static void main(String args[])
  {
    Operation O = new Operation();
    O.add();
  }  O.sub();
}
```

Output:

10
5

Passing parameters:

```
class Addition
{
  void add(int a, int b)
  {
    int c;
    c = a+b;
    S.O.P(c);
  }
  public static void ~~meth~~ main(String args[])
  {
    Addition A = new Addition();
    A.add(15,5);
  }
}
```

Output:

20

3) Write a program to find area of a circle.

util
lib
Scanner
class.

```
class Circle.
{
  void area(int r) ..
  {
    int A;
    A = (22*r*r)/7;
    S.O.P("Area:" +A);
  }
  Public static void main(String args[])
  {
    Scanner Sc = new Scanner(System.in)
    r = Sc.nextInt();
    Circle C = new Circle();
  }  C.area(r);
}
```

Output:

1
Area: 3

4) Write a program to find largest of three numbers.

```
class Max
{
   void largest(int a, int b, int c)
   {
      if (a>b && a>c)
         S.O.P("largest"+a);
      else if (b>a && b>c)
         S.O.P("largest"+b);
      else
         S.O.P("largest"+c);
   }
   public static void main(String args[])
   {
      int a=5, b=4, c=3;
      Max m = new Max();
      m.largest(a,b,c);
   }
}
```

Output:

largest 5

Constructors in java: (Type of method, doesn't return anything)

Constructor is a special method in java used to initialise an object. Rules to be followed by java:

1) Name of constructor should be ~~as~~ same as name of class.

2) Constructor should not return anything/any value like the method return.

i.e, void, int.

3. A constructor is called when an object is created for a class. / instance of class is created

Syntax:

```
constructor()
{
  //statements,
}
```

for every new() atleast one constructor is called.

→ constructor can't be static, abstract, final, synchronized

Ex:

```
class Constructors
{
  int a;
  String name;
  constructors()
  {
    SO.P("This is constructor"+a+name);
  }
  public static void main(String args[])
  {
    constructors c = new constructors();
  }
}
```

output:

This is constructor 0 null

Types of constructors:

1) default constructor

2) parameterized constructor

Default constructor

```
class A
{
  A()
  {
  }
}
```

(blank) if it has parameters all are of default values.

→ compiler automatically creates default const.

Parametrized constructor

```
class A
{
  A(int x)
  {
    k=x;
  }
}
```

Ex

```
class constructor1
{
  int a;
  string name;
  constructor1()
  {
  }
}
```

```
class constructor1
{
  int a,id;
  string name,s;
  constructor1(int a, String name)
  {
    id = a;
    s = name;
  }
  S.O.P(id);
  S.O.P(s);
  public static void main (String args[])
  {
    constructor1 C = new constructor1(1,"kitty");
    S.O.P(id);
    S.O.P(s);
  }
}
```

output: 1
kitty

Constructor over loading:

The constructor overloading represents the same constructor name with different arguments.

Ex:

1) class constructors
{ int a;
  constructors().
  {
    S.O.P("This is default constructor");
  }
  constructors(int a)
  {
    S.O.P("This is constructor3 with single arg");
    S.O.P(a);
  }
  public static void main (String args[])
  {
    constructors obj1 = new constructors();
    constructors obj2 = new constructors(20);
  }
}

output:
This is default constructor.
This is constructor with single argument
20.

Ex:

2) class const4
{ int k,a; String c;
  const4(int a)
  {
    k = a;
    S.O.P(k);
  }
  const4(int b, String c)
  {
    ch = c+b;
    S.O.P(ch); S.O.P(b+c);
  }
  public static void main (String args[])
  {
    const4 c = new const4(5);
    const4 c = new const4(3,"kitty");
  }
}

output.
5
3 kitty

Ex:

3) class const5
{ int k,a; String name, Name;
  const5(int a)
  {
    k = a;
  }

```
const5(String name)
{
    Name=name;
}
void display()
{
    S.O.P(k);
    S.O.P(Name);
}
public static void main(String args[])
{
    const5 c1 = new const5(3);
    const5 c2 = new const5("Abal");
    c1.display();
    c2.display();
}
```

Output:

3 3
Abal. Abal

Garbage collection in Java: free()

For collecting garbage in C 'free' operator is used. In C++ 'delete' operator will be used but in java, the garbage collection will be done automatically because as JVM already know that, java is a robust language. In java the garbage will be collected automatically i.e, garbage collectors run backend and collect unused objects (garbage) from the program.

→ reclaiming the runtime unused memory automatically.

When object becomes eligible for garbage collection:

1) Assigning NULL
2) Re-assigning an object with another object.
3) Anonymous object

→ The method used to cleanup the unused object in program is called finalize method.
finalize()

→ To invoke the finalize method JVM requires following method
1) System.gc()
2) Runtime.gc()

Ex:

```
class Garbage_1
{
    protected void finalize()
    {
        S.O.P("Garbage collected");
    }
    public static void main(String args[])
    {
        Garbage_1 g1 = new Garbage_1();
        g1 = null;
        System.gc();
    }
}
```

Output:

Garbage collected.

→ class Garbage1
{
  protected void finalize()
  {
    S.O.P("Garbage collected");
  }
  public static void main(String args[])
  {
    Garbage1 g1 = new Garbage1();
    Garbage1 g2 = new Garbage1();
    g1 = g2;
    System.gc();
  }
}

output:
Garbage collected.

→ class Garbage1
{
  protected void finalize()
  {
    S.O.P("Garbage collected");
  }
  public static void main(String args[])
  {
    new Garbage1(); ——— Anonymous
    System.gc();
  }
}

output:
Garbage collected.

Static variable, static block & static method:

→ A static variable can be accessed in a single class without the use of class name but in the case of multiple classes the static variable can be accessed with the help of classname.

Static - keyword
class, Variable, method, block
→ can be accessed without obj

Ex:

single class

1) class A
{
  static int a = 10;
  P.S.V.M(String args[])
  {
    S.O.P(a);
  }
}

output:
10

Multiple class

2) class A
{
  static int a = 10;
}
class main
{
  P.S.V.M(String args[])
  {
    S.O.P(A.a);
  }
}
classname.variable

• save file with Main.

output:
10

Static block and static method:

The static block can be executed by default i.e, without calling either with object or classname and static method can be accessed from the main method without the help of object.

static block - used to initialize static variables.

Ex:

```
1) class A
 {
   static
   {
     S.O.P ("This is static block");
   }
   static void display()
   {
     S.O.P ("static method");
   }
   P.S.V.M (String args[])
   {
     display();
   }
}
```

output:

This is static block.
static method.

```
2) class A
 {
   static
   {
     S.O.P ("This is static block");
   }
   static void display()
   {
     S.O.P ("static method");
   }
   static int a = 10;
   P.S.V.M (String args[])
   {
     S.O.P ("The static variable is ", a);
     display();
   }
}
```

static method can access static variables without obj.

output:

This is static block.
The static variable is 10
static method.

'This' keyword: (only single class)

'This' keyword in java is used in the following ways: [This.variable]

1) 'This' keyword is a reference object to refer the instance variable in the current class.
2) It is used to invoke the current class method.
3) To invoke current class constructor.
4) Used to pass as an argument in method call.
5) Used to pass as an argument in constructor call

Ex: compiler order - Local > Instance, static.

case-1)

```
class A
{
  int A = 25; ⟶ Instance variable.
  void display()
  {
    S.O.P(A);
  }        This.A
  P.S.V.M (String args[])
  {
    A a = new A();
    a.display();
  }
}
```

output:
25

In the above case A is printed as 25 it means the compiler by default understood that S.O.P(A) as S.O.P(This.A)

case : 2 ii

```
class A
{
  int a=25;
  void display()
  {
    int a=30;
    S.O.P(a);
  } S.O.P(this.a);
    P.S.V.M(String args[])
    {
      A s = new A();
      s.display();
    }
}
```

Output:
30
25

Here a=25 is instance variable and a=30 is local variable In this situation the compiler gives preference to the local variable. So, that It prints 30 and then 25.

By observing above two cases we need to understand that 'This' keyword is used to refer an instance variable.

case : 2

```
class A
{
  void display1()
  {
    S.O.P("Method 1");
  }
  void display2()
  {
    S.O.P("Method 2");
  }
  P.S.V.M(String args[])
  {
    A a = new A();
    a.display1();
    a.display2();
  }
}
```

output:
Method 1
Method 2

In the above program we created a object 'a' and called methods display1, display2 to get the o/p. But if we implement the 2nd feature

(ii)
```
class A
{
  void display1()
  {
    S.O.P("Method 1");
  } this.display2();
  void display2()
  {
    S.O.P("Method 2");
  }
  PSVM(String args[])
  {
    A a = new A();
    a.display1();
  }
}
```

output:
Method 1
Method 2

case-3:

```
class A
{   int x;
    A constructor1 ( )
    {
      S.O.P("constructor1");
    }
    A constructor2 (int x)
    {
      S.O.P ("constructor2");
      this ( );
    }
    P.S.V.M (String args[])
    {
      A a = new A( );
      A b = new A(10);
    }
}
```

output:

case 4:

```
class A
{
  int x,y,z;
  A (int x,int y, int z)
  {
  this.x = x;
  this.y = y;
  this.z = z;
  }
  void display( )
  {
    S.O.P(x);
    S.O.P(y);
    S.O.P(z);
  }
  P.S.V.M (String args[]);
  {
  A a = new A(10,11,12);
    a.display( );
  }
}
```

output:

10
11
12

command line arguments in java: (Values during runtime)

1)
```
class Command
{
  public static void main (String args[])
  {
    S.O.P ("Hello"); / S.O.P (args[0]);
  }
}
```

→ javac command.java.

→ java Command Hello

output:

Hello.

** To convert str to int

Integer.parseInt(a[0])

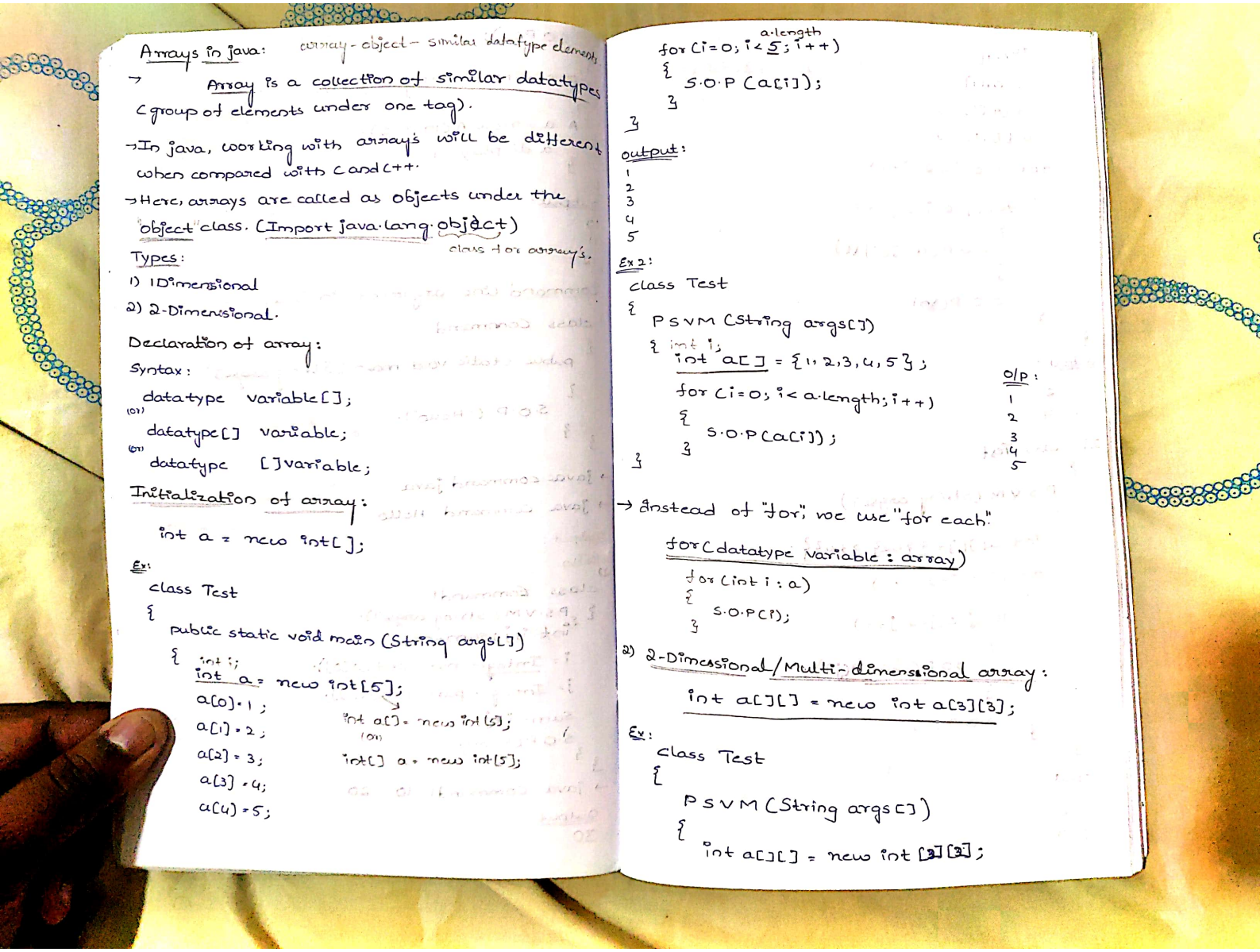
Method name of Int.

2)
```
class Command1
{ P.S.V.M (String args[]).
  {
    int i,j,sum=0;
    i = Integer.parseInt(args[0]);
    j = Integer.parseInt(args[1]);
    sum = i+j;
    S.O.P(sum);
  }
}
```

→ java Command1 10 20

output:

30



Arrays in java: array - object - similar datatype elements

→ Array is a collection of similar datatypes (group of elements under one tag).

→ In java, working with arrays will be different when compared with C and C++.

→ Here, arrays are called as objects under the "object" class. (Import java.lang.object) class for arrays.

Types:

1) 1Dimensional
2) 2-Dimensional.

Declaration of array:

Syntax:

datatype variable[];
(or)
datatype[] variable;
(or)
datatype []variable;

Initialization of array:

int a = new int[];

Ex:

```
class Test
{
  public static void main (String args[])
  { int i;
    int a = new int[5];
    a[0]=1;
    a[1]=2;
    a[2]=3;
    a[3]=4;
    a[4]=5;
```

int a[] = new int[5];
(or)
int[] a = new int[5];

```
    for(i=0; i<a.length; i++)
    {
      S.O.P(a[i]);
    }
}
```

output:

1
2
3
4
5

Ex2:

```
class Test
{
  P S V M (String args[])
  { int i;
    int a[] = {1,2,3,4,5};
    for(i=0; i<a.length; i++)
    {
      S.O.P(a[i]);
    }
}
```

O/P:
1
2
3
4
5

→ Instead of "for", we use "for each"

```
for(datatype variable : array)
  for(int i : a)
  {
    S.O.P(i);
  }
```

2) 2-Dimensional/Multi-dimensional array:

int a[][] = new int a[3][3];

Ex:

```
class Test
{
  P S V M (String args[])
  {
    int a[][] = new int [3][3];
```

```
to
a[0][0] = 1;        int a[][] = {{1,2},{3,4}}
a[0][1] = 2;
a[1][0] = 3;
a[1][1] = 4;
for(i=0;i<2;i++)
{
  for(j=0;j<2;j++)
  {
    S.O.P(a[i][j]\t);
  }
  S.O.P(\n);
}
}
```

output:

```
1  2
3  4
```

2)

```
class Test
{
  P S V M (String args[])
  {
    int a[][] = {{1,2},{3,4}};
    for(i=0;i<2;i++)
    {
      for(j=0;j<2;j++)
      {
        S.O.P(" "+a[i][j]);
      }
      S.O.P( );
    }
  }
```

Output:

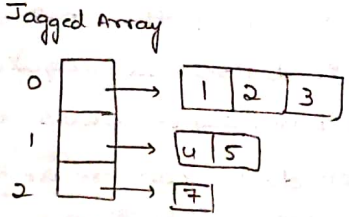
```
1 2
3 4
```

Jagged array:

It is an array of array's with diffacent size of an array.

Ex:

```
1 2 3
4 5
7
```



UNIT-III

Inheritance, Packages and exception

Inheritance: (Adv: code reusability)

It is one of the principle of 'OOP'. Inheritance is defined as acquiring the properties from parent/base/super class to child/sub/derived class. If you do we can reuse the code can be implimented through inheritance.

→ It is 'IS-A' kind of relationship i.e, inheriting from parent to child)

→ Ex:

```
class Employee
{
  float salary = 73000;
}
class Programmer extends Employee → save this as filename
{
  int a = 30;
  PSVM(String args[])
  {
    Programmer P = new Programmer();
    S.O.P(P.salary);
    S.O.P(P.a);
  }
}
```

output:

```
73000
30
```

Types of Inheritance:

1. single level
2. Multi level
3. Hierarchical

} through class

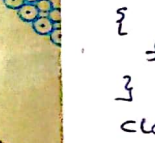
4. Multi-level
5. Hybrid

} through interfaces as java doesn't support.

Single Level Inheritance:

If the base class is A, then the properties will be inherited to class B (derived) by using "extends" keyword.

Syntax:




```
class A
{
  //stmts;
}
class B extends A
{
  //stmts;
}.
```

Ex:

```
class A
{
  S.O.P("Hello");
  a = 10;
}
```

```
class A
{
  void display()
  {
    S.O.P("This is base class");
  }
}
class B extends A
{
  void display1()
  {
    S.O.P("Derived class");
  }
```

```
P.S V M (String args[])
{
    B b = new B();
    S.O.P(b.display);
    S.O.P(b.display1);
}
}
```

Output:

This is base class
Derived class.

Multi-level Inheritence:

class A — parent
↑
class B — derived & base
↑
class C — derived

```
class A
{
  stmts;
}
class B extends A
{
  stmts;
}
class C extends B
{
  stmts;
}
```

Ex:

```
class A
{
  void display()
  {
    S.O.P("Base class");
  }
}
class B extends A
{
  void display1()
  {
    S.O.P("Base & derived class");
  }
}
class C extends B
{
  void display2()
  {
    S.O.P("derived class");
  }
```

```
P S V M (String args[])
{
  C c = new C();
  S.O.P(c.display);
  S.O.P(c.display1);
  S.O.P(c.display2);
}
}
```

output:

Base class
Base & derived class
derived class.

2)

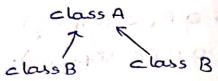
```
class A
{
  void add(int a, int b)
  {
    S.O.P(a+b);
  }
}
class B extends A
{
  void sub(int a, int b)
  {
    S.O.P(a-b);
  }
}
class C extends B
{
  void multiply(int a, int b)
  {
    S.O.P(a*b);
  }
}
P S V M (String args[])
{
  int a, b;
  C c = new C();
  S.O.P(c.add(4,2));
  S.O.P(c.sub(4,2));
  S.O.P(c.multiply(4,2));
}
}
```

output:
6
2
8

Hierarichal Inheritance:

Syntax:

```
class A
{
  stmts;
}
class B extends A
{
  stmts;
}
class C extends A
{
  stmts;
}
```



Ex:

```
1) class A
   {
     void D
     {
       S.O.P("Base");
     }
   }
   class B
   {
     void D1
     {
       S.O.P("child1");
     }
   }
   class C
   {
     Void D2
     {
       S.O.P("child 2");
     }
     PSVM (String args[])
     {
       C c = new C();
       S.O.P(c.D);
       S.O.P(c.D2);
     }
   }
```

O/P:

```
Base
child 2
```

c.D1 → error

```
2) class A
   { void add (int a, int b)
     { S.O.P(a+b);
     }
   }
   class B extends A
   { void sub (int a, int b)
     { S.O.P(a-b);
     }
   }
   class C extends A
   { void mul (int a, int b)
     { S.O.P(a*b);
     }
     PSVM(String args[])
     {
       C c = new C();
       S.O.P(c.add(4,2));
       S.O.P(c.sub(4,2));
     }
   }
```

O/P:

```
6
2
```

Method overloading:

Ex: A method overloading tells us about using the same method with different arguments/parameters in the class. whereas method overriding tells us about parent class method overriding by child class.

Ex:

```
class Overloading
{
    void add(int a, int b)
    {
        S.O.P(a+b);
    }
}
class Overloading1 extends Overloading
{
    void add(int a, int b, int c)
    {
        S.O.P(a+b+c);
    }
    P S V M (String args[])
    {
        int a, b, c;
        Overloading1 o = new Overloading1();
        o.add(1,2);
        o.add(1,2,3);
    }
}
```

O/P:

```
3
6
```

Ex:

```
class Overriding
{
    void add(int a, int b)
    {
        S.O.P(a+b);
        S.O.P(a);
    }
}
```

} overridden method

```
class Overriding1 extends Overriding
{
    void add(int a, int b)
    {
        S.O.P(a+b);
    }
    P S V M (String args[])
    {
        Overriding o = new Overriding();
        o.add(1,2);
    }
}
```

} → overriding method.

O/P:

```
3
```

"Super" keyword:

"super" keyword is used to refer an immediate parent class object and also used to invoke the instance variable, method and constructor.

Ex:

```
class overiding
{
    void add(int a, int b)
    {
        S.O.P(a+b);
        S.O.P(a);
    }
}
class Overiding1 extends overiding.
{
    void add(int a, int b)
    {
        S.O.P(a);
    }
    super.add(a,b);
    P S V M (String args[])
    {
        Overiding1 o = new Overiding();
        o.add(1,2);
    }
}
```

O/P

```
1
3
1
```

"final" keyword:

In java, we have three main advantages by using the keyword "final". They are

1) The variable which is declared as final cannot be reinitialised.
2) The method which is declared as final cannot be over-ridden.
3) The class which is declared as final cannot be Inheritted.

case(i):

```
class Case1
{
    PSVM(String args[])
    {
        int a=10;
        a=a+1;
        b=a+10;
        S.O.P(a);
        S.O.P(b);
    }
}
```

O/P:

11
21

```
→class Case1
{
    PSVM (String args[])
    {
        final int a=10;
        // a=a+1;
        b=a+10;
        S.O.P(a);
        S.O.P(b);
    }
}
```

O/P
10
20

In the above program the value of a is finalized as 10 and can't be changed further (as 'final' variable cannot be reinitialized).

case(ii):

```
class Case2
{
    PSVM(String args[])
    {




    void display()
    {
        S.O.P("Method");
    }
}
class Case2a extends Case2.
{
    void display()
    {
        S.O.P("Overriding Overridden method");
    }
    PS VM(String args[])
    {
        Case2a c = new Case2a();
        c.display();
    }
}
```

O/P

Overriding Overridden method.

```
class Case2
{
    final void display()
    {
        S.O.P("method");
    }
}
class Case2a extends case2
{
    void display()
    {
        S.O.P("Method 1");
    }
    PSVM(String args[])
    {
        Case2a c = new Case2a();
        c.display();
    }
}
```

O/P:

Error. (Overridden method is final)

In the above program the child class method cannot be overridden because the parent class method is defined as 'final'.

Case(iii):

```
final class Case3
{
    void display()
    {
        S.O.P("Method1");
    }
}
class Case3a extends Case3
{
    void display()
    {
        S.O.P("Method2");
    }
    PSVM(String args[])
    {
        Case3a c = new Case3a();
        c.display();
        c.display();
    }
}
```

O/P

Error cannot inherit from final class

In the above program there's compilation error because the class itself is defined as final then the base class data will not be inherited.

Abstract class:

Abstraction is nothing but hiding of data but implimentation cannot be shown.

Ex: ATM

→ An abstract class is defined as "It should consist of only declaration part of the method but not implimentation".

→ In abstract class, the method implimentation will be done in derived classes (any derived classes).

→ An abstract class may or maynot have abstract methods.

→ An abstract class doesn't create an object i.e, Object will be created only the method implimentation is found.

Ex:

```
abstract class A
{
  abstract void m1();
  abstract void m2();
  void m3();
  { s.o.p(1);
  }
}
class B extends A.
{
  PSVM(String args[])
  {
    B b = new B();
    b.m1();
    b.m2();
    b.m3();
  }
  void m1()
  {
    s.o.p(2);
  }
  void m2()
  {
    s.o.p(3);
  }
}
```

O/P

2

3

1

Interfaces:

Interface is a key word or concept useful for the implimentation of complete abstraction and multiple inheritence. Because the abstract class is also allowing the general method implimentation in addition to abstract method.

→ The concept of inheritence acquires the properties of two or more base classes (or) parent classes is not supported. So, the interface concept resolve the problem of acquiring properties of base class. [multiple Inheritence].

Ex:

```
class A
{
  void display()
  {
    s.o.p("Hello");
  }
}
class B
{
  void display()
  {
    s.o.p("Hi");
  }
}
class C extends A,B
{
  P.s.v.m(String args[])
  {
    C c = new C();
    c.display();
    c.display();
  }
}
```

O/P:

Error

In the above program multiple inheritence is not possible.

Ex:

```
interface A
{
  abstract void display();
}
interface B
{
  abstract void display1();
}
class C implments A
{
   void display()
   { S.O.P("Hello");
   }
   PSVM(String args[])
   {
      C c=new C();
      c.display();
   }
}
```

O/P:

Hello

2)

```
interface A
  {
     abstract void add(int a,int b);
     abstract void sub(int a,int b);
     abstract void mul(int a,int b);
  }
  class B implments A
  {
      void add(int a,int b)
      {  S.O.P(a+b);
      }
      void sub(int a,int b)
      {  S.O.P(a-b);
      }
      void mul(int a,int b)
      {   mul(a*b);
      }
       PSVM(String args[])
       {
          B  b= new B();
          b.add(1,2);
          b.sub(4,2);
          b.mul(1,1);
       }
}
```

O/P

3
2
1

* Multiple inheritence.

```
interface A
  {
    abstract void display();
  }
interface B
    {
      abstract void display1();
    }
class C implments A,B
{
    void display()
    {
      S.O.P("Hello");
    }
    void display1()
    {
       S.O.P("people");
    }
```

```
    PSVM(String args[])
    {
        C c = new CC);
        c.display();
        c.display1();
    }
}
```

Output:

Hello
people

Hybrid inheritence:

(Through class)

1) Single + Multi-level
2) Single + Hierarichal → Hybrid
3) Multi-level + Hierarichal
4) Single + Multilevel + hierarichal.

It is a combition of two or more inte inheritences as shown.

→ Hybrid inheritence can also be implemented through interface.

Ex: [M+H]

```
class A
{
    void display()
    {
        S.O.P("A method");
    }
}
class B extends A
{
    void d1()
    {
        S.O.P("B method");
    }
}
class C extends B
{
    void d2()
    {
        S.O.P("C method");
    }
}
class D extends A
{
    void d3()
    {
        S.O.P("D method");
    }
    PSVM(String args[])
    {
        C c = new CC);
        D d = new d();
        d.display();
        c.d1();
        c.d2();
        d.d3();
    }
}
```

O/P:

A method
B method
C method.
D method

Hybrid inheritance through interfaces:

```
class A
{
    void a()
    {
        S.O.P("Method A");
    }
}
interface B
{
    abstract void b();
}
```

```
interface C
{
   abstract void cc();
}

class D extends A implements B,C
{
   void bc()
   { S.O.P("Method B");
   }
   void cc()
   {
     S.O.P ("Method C");
   }
   P S V M (String args[])
   { D d = new DC();
     d.ac();
     d.bc();
     d.cc();
   }
}
```

O/P:
Method A
Method B
Method C.

Interface extension:

```
interface A
{
   abstract void ac();
}
interface B extends A
{
  abstract void bc();
}
class C implements A,B
{
   void bc()
   {
     S.O.P("Hello");
   }
   void ac()
   {
     S.O.P("Bye");
   }
   P S V M (String args[])
   {
     C c = new CC();
     c.bc();
     c.ac();
   }
}
```

O/P:
Hello
Bye

Interface V/s Abstract class.

parameters	Interface	Abstract class
1) Method :	only abstract methods are allowed	abstract and non-abstract methods are allowed.
2) 'final' :	By default the variable or method are declared.	May contain non-final variable or method.
3) Accessibility :	By default public	private, protected are also allowed.
4) Multiple Inheritence :	Supported	Not supported.
5) Variables :	Only static & final.	Non-final, final, static, non-static
6) Implementation (method imp) :	No.	Yes.
7) Keyword :	implements	extends.

Access specifiers:

Major access specifiers in java are

1) Default
2) Public
3) Protected
4) Private

Ex:

1) default.

```
class A
{
  int a=3;
}
class B extends A
{
   void display()
   {
     S.O.P(a);
   }
   PSVM(String args[])
   {
     B b = new B();
     b.display();
   }
}
```

O/P: 3

2) public.

```
class A
{
  public int a=3;
}
class B extends A
{
  void display()
  {
    S.O.P(a);
  }
  PSVM(String args[])
  {
    B b = new B();
    b.display();
} }
```

O/P

3

3) private protect

```
class A
{
protected int a=3;
}
class B extends A
{
  void display();
  { S.O.P(a);
  }
  PSVM(String args[])
  { B b = new B();
    b.display();
} }
```

O/P:

~~Error to~~

3

4) private

```
class A
{
  private int a=3;
}
class B extends A
{
   void display()
   { S.O.P(a);
   }
   PSVM (String args[])
   { B b = new B();
     b.display();
} }
```

O/P: (Compilation error)

Error