Java

19-07-24

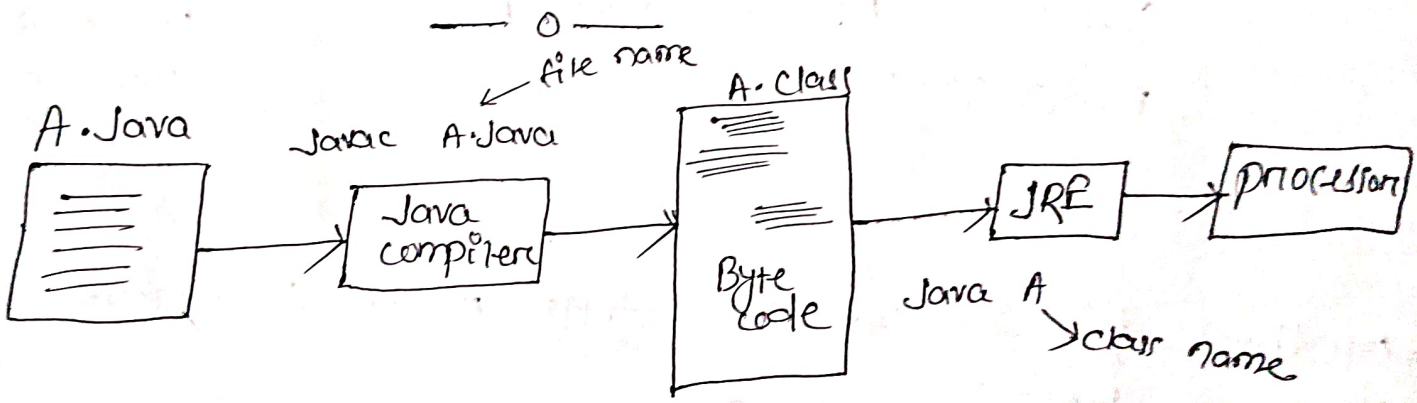
→ Basic prog.
- variable, operator, if-else strut, looping strut, method, programs.

→ oops
class/object, constructor, overloading, Inheritance, override, casting, abstract method & class, Interface, polymorphism, abstraction, Encapsulation, Aggregation, composition.

→ String prog, Array prog, Data structure, collection framework, file operation, exception handling, multi-threading, Java 8 features, JDBC, servlet, JSP, hibernate, spring.

→ SQL, html, CSS, Javascript.

Java full stack



→ editplus download - 6.0 2st one

```
class D {
    public static void main (String args []) {
        S.O.P ("Main starts");
        D.M1();
        D.M1();
        S.O.P("Main ends");
    }
    public static void M1() {
        S.O.P ("M1 starts");
        S.O.P ("M1 ends");
    }
}
```

```
Class E {
 psvm (String [] args) {
   S.o.p ("Main method of class E");
   D.m1();
 }
}
```

—o—

```
Class F {
  psvm2 (St... ) {
    S.o.p ("Main of F");
  }
}
```

In this we don't have main method so can be compile but can't give o/p on run or can't give o/p.

```
class G {
    psvm (St... ) {
        S.o.p (" M2 in class F");
        F.M2();
    }
}
```

Javac G.java
Java G.

- Method is a set of stmts written to execute a business task.
- where a Java class is executed, JRE calls main method. So execution always starts from main method.
- Other methods are executed when they are called.
- Main method is not compulsary to compile a class, but main method compulsary to execute a class

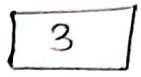
```
class H {
  psvm (Stri... ) {
    S.o.p ("Main of class H started");
    H obj1 = new H();
    obj1.m3();
  }
```

```
// non-static or
// instance method
public void m3()
{
S.o.p ("non-static method");
}
}
```

Variable :-

It is a small memory which is used to store one value at a time

```
a = 1+2 ;
↓
variable
```



→ Before using variables, variable must be declared.

→ datatype varName ;

```
class A {
  psvm (S--- ){
    int a ;
    a = 10;
    S.o.p (a);
    a = 20;
    S.o.p (a);
  }
}
```

cd..

cd variables

D:\CJDM17\variables .

Datatypes

primitive type

integer numbers

* byte (1byte memory allocation create)
* short (2byte)
* int (4byte)
* long (8byte)

4byte

* float } decimal no.
* double } 8byte
* char
* boolean.

Non-primitive type

Class name can be named as datatype

```
class B {
    psvm (S.....){
    // float f1 = 1.5 ; → compile time error
    float f2 = 1.5f ;
    S.o.p (f2);
    double d1 = 1.5;
    S.o.p (d1);
  }
}
```

```
class C{
   psvm ( ) {
    char c1 = 'a' ;
    S.o.p (c1);
    boolean b1 = true;
    boolean b2 = false;
    S.o.p (b1);
    S.o.p (b2);
  }
}
```

→ In Java class names can be used as a datatype while declaring a variable.
→ In Java class is a non-primetive datatype

```
class D {
    psvm (--) {
    A a1 = new A();
    B b1 = new B();
// D d1 = new A(); //Compile time error.
    S.o.p ("print");
}
}
```

Types of variables :-

→ Local var
→ Static var.
→ Non-Static var.

Operators :-

→ It is a predefined symbols which is used to perform a specific task.
→ Based on no. of operands operators are 3 types

① Unary operators
② Binary operators
③ Ternary operators

① Unary operators :-

→ which operators are accepting one operand as an argument. those are called as unary operators.

② Binary operators :-

→ which operators are accepting two operands as arguments those are called as binary operators.

③ Ternary operators :-

→ which operators are accepting three operands as arguments. those are called as ternary operators.

Types of operators depend on functionality —

→ Arithmatic operators (+, -, *, %, /)

→ Assignment operators (+=, -=, %=, *=, /=)

→ Relational or comparision operators (==, <, >, <=, >=, !=)

→ Logical operators (&&, ||, !)

→ Increment & decrement operators

→ conditional operators

→ Bitwise operators.

Arithmatic (binary operators)

```
class Demo {
    psvm (.....) {
    int a = 20;
    int b = 10;
    s.o.p(a+b); // 30
    s.o.p (a-b); // 10
    s.o.p (a*b); // 200
    s.o.p (a%b); // 0
    s.o.p (a/b); // 2
    }
}
```

Assignment operator —

int a = 50;

operator	Ex-1	Ex-2 full form
+=	a+=50	a = a+50
-=	a-=20	a = a-20
*=	a*=2	a = a*2
%=	a%=10	a = a%10
/=	a/=10	a = a/10

int a=20;
s.o.p("A=" + a); // concatenation (add no. with " " character)

```
Class assign {
  psvm ( ... ) {
    int a = 50;
    a += 5;  // a = a+5 = 50+5 = 55
    a -= 3;  // a = a-3 = 55-3 = 52
    a *= 1;  // a = a*1 = 52*1 = 52
    a %= 10; // a = a%10 = 52%10 = 2
    a /= 10; // a = a/10 = 2/10 = 0.0
    S.o.p (a);
  }
}
```

binary operators

by a += 5 : a = a+5

2 operands are there

```
int a=30;
int b=20;
a += b; // 30 = 30+20 = 50
int c = a;
S.o.p (a); //50
S.o.p (c); //50
```

Relational or comparision operator :-

```
int a=20;
int b=10;
S.o.p ("20+10", +(a+b));
```

or

```
S.o.p (a+" "+b+"="+a+b);
```

```
int a=10;
int b=20;
int c=0;
c = b;
b = a;
a = c;
S.o.p(a);
S.o.p(b);
```

Swapping using 3rd variable

```
class concat {
  psvm ( ... ) {
    int a=20;
    int b=10;
    int c = a+b;
    S.o.p (a+" "+b+" = "+c);
  }
}
```

```
class Demo {
  psvm ( ... ) {
    int age = 25;
    long mobno = 8339855633;
    S.o.p("My name is Newton" + " " + "My age is " + age + " and my mobile no is " + mobno);
  }
}
```

S.o.p("My age

~~S.O.P ("My age is " + age)~~

(1) Co. a prog to swap 2 nos — in Java — using 3rd variable —

```
class swap {
  psvm (...) {
    int a = 20;
    int b = 10;
    int c = 0;
    S.O.P ("Before swapping");
    S.O.P ("A = " + a);
    S.O.P ("B = " + b);
    c = b;
    b = a;
    a = c;
    S.O.P ("After swapping");
    S.O.P ("A = " + a);
    S.O.P ("B = " + b);
  }
}
```

23.07.24

a = 97, A = 65

'a' + 1 → 98 (char + number, add)

'a' + '1' → a1 (char + char, concat, string → string)

++ (increment)

++5 ; X

++a ; (pre increment)

a++ ; (post increment)

```
class A {
  psvm (---) {
    int a = 0;
    int b = a++ + 2;
```

pre the a value is 0, after that it will changes to 1 in the memory

```
    S.O.P (a);
    S.O.P (b);
  }
}
```

o/p: 1, 2

S.O.P ('1' + 2 + 3) ; → 123 (12, 123)

S.O.P ('1' + (2+3)) ; → 15

S.O.P (1 + 2 + '3') ; → 33 (3, 33)

-- (decrement)

--a (predecrement)

a-- (post decrement)

```
int b = ++a + 2
```

o/p: 1, 3

```
class A {
  psvm (---) {
    int a =0;
    int b = ++a + a++;
    s.o.p (a);
    s.o.p (b);
  }
}
```

o/p
2
2

a [0 1 2]

++a + a++
1 1 = 2

```
int b = ++a + a+;
```
o/p
2
1

```
class A {
  psvm ( ) {
    int a = 0;
    s.o.p (++a); // 1
    s.o.p (++a); // 2
    s.o.p (a++); // 2
    s.o.p (a++); // 3
  }
}
```

Relational operator :-

precedance

→ It is predefined & we can change the precedance by using bracket () or parenthesis.

order of evaluation

unary operator ++ -- ! ~
* / %
+ -
relational
bitwise
logical
assignment

Logical operator

&& and
|| or
! not

Boolean

a > 3 && b == 0 ✓
a+3 && b-4 ✗

and (&&)

t t → t
t f → f
f t → f
f f → f

or (||)

t t → t
t f → t
f t → t
f f → f

not (!)

t → f
f → t

```
class B {
    psvm (——) {
     int a=0;
     S.o.p (a==0);          //true
     S.o.p (!(a==0));       //false
     S.o.p (!a==0);         // compile time error.
```

24-07-24 Exm

```
class A {
    int K=30;
    psvm (——) {
     B b = new B();
     S.o.p (b.K);           // 30
    }
}
```

(b: class name, object/variable; new: keyword; B(): class name; object creation.)

Exm

```
class A {
    static int i=10;
           int j=20;
    psvm (——) {
     S.o.p (A.i);
     A obj = new A();
     S.o.p (obj.j);
    }
}
```

Non-static variable :-

→ Declaring a variable inside class block outside method block without static keyword is known as non-static variable.

→ we can't access non-static variables using class name.

→ we can access non-static variables using object reference

→ we can create an obj using "new" keyword

Local variables :-

→ inside the class & method.

→ creating a variable inside the class block & inside the method block is called as local variable.

→ local variables, we can't use outside the method block.

→ we can call local variable directly inside a method.

Exm:

```
class A {
    psvm (String [] args) {
        int i=10; // local variable
        S.o.p (i);
    }
}
```

```
class A {
    static int i=10;
    int j=20;
    psvm (---) {
        int K=30;
        A b=new A();
        S.o.p (A.i);
        S.o.p (b.j);
        S.o.p (K);
    }
}
```

```
class A {
    static int i;
    double j;
    psvm (---) {
        S.o.p (A.i); // 0
        A a=new A();
        S.o.p (a.j); // 0.0
    }
}
```

#Note:-

→ we cannot use local variable without initializing a value.

→ If we try to use local varible without initializing we will get compile time error.

→ Local variable does not contain any default values

```
class A {
  psvm (---) {
      int K;
      s.o.p (K); // error
  }
}
```

Static variables

→ If a variable declared inside class block outside method block with Static keyword it known as static variable.

→ we can call static variable using class name.

→ we cann't declare static variable inside method block.

Ex:-

```
class Demo {
    Static int i = 10;
    psvm (---) {
      s.o.p (Demo.i);
    }
}
```

24.07.24

```
class C {
  psvm (...){
  int a=5;
  int b=6;
  S.o.p (a++>5 && ++b>6); //false
  S.o.p (a); //6
  S.o.p (b); //6
```

```
S.o.p(a++>=5 && b++>6)
S.o.p(a); //6
S.o.p(b); //7
```

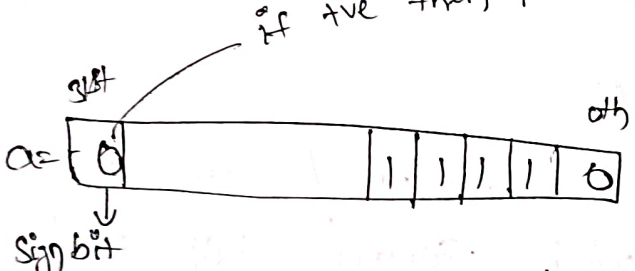
Bitwise operators:-

- → & (and) , >> (signed right shift) when we shift sign bit will add last
- → | (or) , >>> (unsigned right shift) when we shift the 0 will add last.
- → ^ (XOR) , << (left shift)

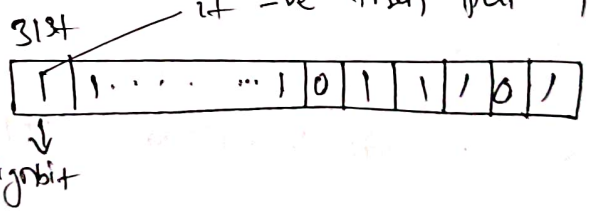
0	1	1
1	0	1
1	0	0

- → ~ (INOT) ,

int a=30;



int b=-35



35 = 100011

32 bit = 0 ---- 00100011 → 35

1's comp = 1 ---- 11011100 → 1's

2's com = +1

1 ... 11011101 → 2's

this

2's comp of

1's = 0 00100010

+1

2's 0 ..... 00100011 → 35

```
class D {
  psvm (···) {
    int a = 30;
    int b = -35;
    int c = a&b;
    s.o.p (c); //28
    int d = a|b;
    s.o.p (d); // -33
    int e = a^b;
    s.o.p(e); // -61
  }
}
```

```
class E {
  psvm (···) {
    int a = 30;
    int b = ~a;
    s.o.p (b); //-31
    int b = a>>2;
    s.o.p (b); //7
    int a = -30;
    int b = a>>>2;
    s.o.p(b); // 107....
```

```
a = cond ? exp1 : exp2;
```

```
class f {
  psvm() {
    int a = 10;
    int b = 20;
    int n = a>b ? a : b;
    s.o.p (n); //20
  }
}
```

25.07.24

Scanner :→ It is a readymade class.

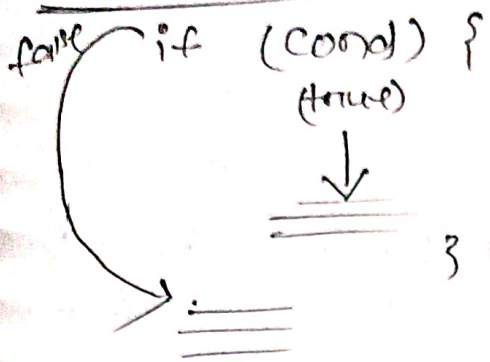
```
import java.util.Scanner;
class a{
  psvm(..){
Scanner sc=new Scanner(System.in);
 S.o.p("Enter a line");
String s=sc.nextLine();
 S.o.p(s);
}
}
```

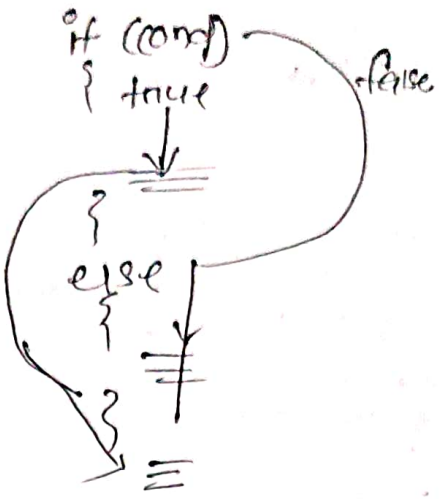
```
package java.util;
class Scanner{
public String next(){
}
}
```

```
import java.util.Scanner;
class Add{
    psvm(--){
    Scanner sc=new Scanner(System.in);
    S.o.p("Enter 2 numbers");
    int a=sc.nextInt();
    int b=sc.nextInt();
    S.o.p(a+"+"+b+"="+(a+b));
}
}
```

nextInt() ⟶ returns int
nextByte() ⟶ " byte
nextShort() ⟶ " Short
nextLong() ⟶ " long
nextfloat() ⟶ " float
nextDouble() ⟶ " double
nextBoolean() ⟶ " boolean
next() ⟶ " Single character.
nextLine ⟶ " String.

if clause:-






```
if (cond1) {
    ===
}
else if (cond2)
{
    ===
}
else if (cond3)
{
    ===
}
else {
    ===
}
```

not compulsory

EXM

```
import java.util.Scanner;
class Evenorodd{
  psvm (---){
Scanner sc = new Scanner(System.in);
S.o.p ("enter a num");
int n = sc.nextInt();
boolean res = n%2==0;
If (res) {
  s.op ("even");
} else {
    s.op ("odd");
  }
}
```

or

```
if (n%2==0){
    s.op ("even");
} else {
    s.o.p ("odd");
  }
}
```

```
import java.util.Scanner;
class BiggestNum {
  psvm (---){
Scanner sc = new Scanner (System.in);
S.o.p ("Enter 2 nos");
int a = sc.nextInt();
int b = sc.nextInt();
if (a>b) {
    s.o.p ("a is biggest");
}else if (b>a){
    s.op ("b is biggest"); }
else{s.o.p ("Both are equal");}}
```

```
Class A {
  psvm( ——— ) {
    int a=5;
    if (a%2==0) → it is false
    ++a;  ——→ so this will not executed bcz if we not use { } then inside if there will one line is in.
    ++a; } These two
    ++a; } are executed
    s.o.p (a); →7
  }
}
```

```
import java.util.Scanner;
class Marriage {
  psvm ( ) {
    Scanner sc = new Scanner (System.in);
    s.o.p ("enter age");
    int a = sc.nextInt ();
    if (a>=26) {
      if (a<=30)
        { s.op ("congrats");
        }
      else
        { s.o.p ("go to temple");
        }
    }
  else { s.op ("concentrate on reading"); }
```

```
import java.util.Scanner;
class currencycal {
  psvm (——) {
  Scanner sc = new Scanner (System.in);
  s.o.p ("Enter amount");
  int a = sc.nextInt();
  if (a>=2000) {
    s.o.p ("2000 x" + (a/2000));
    a = a%2000;
  }
  if (a>=500) {
    s.o.p ("500 x" + (a/500));
    a = a%500;
  }
  if (a>=200) {
    s.o.p ("200 x" + (a/200));
    a = a%200;
  }
  if (a>=100) {
    s.op("100 x" + (a/100));
    a = a/100;
  }
  }
}
```

26/07/24

Switch :-

> Default is not mandatory.
> No specific order for cases
> case value should not be repeated.

Switch (expression) → byte, short, int, char, ~~boolean~~ String

```
{
    case value1 :
        ========
        break;
    case value2 :
        ========
        break;
    case value3 :
        ========
        break;
    .
    .
    .
    Default :
    ========
}
```

Exm

```
import java.util.Scanner;
class weekdayName {
    psvm (---) {
Scanner sc = new Scanner(System.in);
S.o.p ("Enter the weekday name");
int d = sc.nextInt();
switch(d) {
    case 1 : S.o.p ("Sunday");
             break;
    case 2 : S.o.p ("Monday");
             break;
    case 3 : S.o.p ("Tuesday");
             break;
    Case 4 : S.o.p ("Wednesday");
             break
```

→ all case 1 → S.o.p ("sunday");

```
    case 5 : sop("Thursday");
             break;
    case 6 : S.o.p("Friday");
             break;
    case 7 : S.o.p("Saturday");
             break;
    default : S.o.p("Invalid");
}
}
}
```

```
import java.util.Scanner;
class leapyear{
  psvm (---){
  Scanner sc=new Scanner(System.in);
  S.o.p ("Enter year");
  int y=sc.nextInt();
  if ((y%400==0) || (y%4==0 && y%100!=0)){
      S.o.p ("Leap year");
    }
  else
  {
    S.o.p ("Not a leap year");
  }
};
```

```
import java.util.Scanner;
  class Numofdays{
    psvm(---){
     Scanner sc=new Scanner(System.in);
     S.o.p ("Enter month");
     int m=sc.nextInt();
     S.o.p ("Enter year");
     int y=sc.nextInt();
Switch (m){
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
  case 10:
  case 12: S.o.p ("31 days");
           break;
  case 4:
  case 6:
  case 9:
  case 11: S.o.p("30 days");
           break;
  case 2:
    if ((y%400==0)||(y%4==0
        S.o.p("29 days");
    else { && y%100!=0)){
        S.o.p ("28 days");
        break;}
  default: S.o.p("invalid month");
}}
```

Loop :-

for, while, do-while, for-each.

Syntax (for)

for (Statement 1 ; cond ; statement 2)
{
}

① executes only once
② true
③
④ executes at the end of each iteration.

Ex: class A {
psvm (—) {
for (int i=1 ; i<=3 ; i++) {
s.o.p ("Hello world") ;
} } }

for (int i=1 ; i<=3; i++, i++)
i=i+2)
i+=2)

for (int i=1 ; i<=10;
i=i+2) {
s.o.p (i);
} }

---

class A {
psvm (—) {
int i=0;
for (i=1 ; i<=3 ; i++) {
s.o.p (i);
}
s.o.p ("----");
s.o.p (i);
} }

O/P
1
2
3

4

---

class A {
psvm (—) {
int i=0;
for (i=1 ; i<=3; i++)
s.o.p (i); ← one line is in the body of for loop
s.o.p (i); } This two are out of
s.o.p (i); } for loop
} }

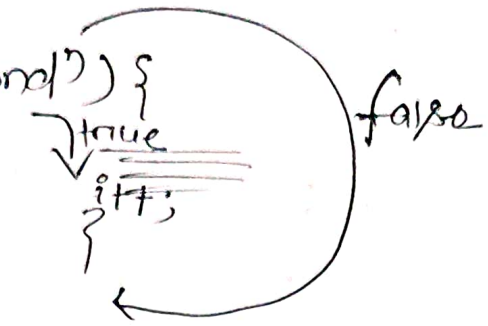
O/P
1
2
3
4
4

for (s.o.p("hello"); i<=3; s.o.p ("world")) {
s.o.p (i);
i++;
}

27.07.24

while loop :-

Syntax :- while (cond?) {
    ++;
}



Exm:

```
class A {
    int a;
    psvm ( ) {
        while (a > 0) {
            a = a;
```

a	a
0	0
1	0+1
3	1+2
6	3+3

Exm

```
class A {
    psvm ( ) {
        int a = 0;
        int i = 1;
        while (i <= 10) {
            a = a + i;
            a++;
        }
    } s.o.p(a);
}
```

(or)

```
for (i=1 ; i<=10 ; i++) {
    a = a + i;
    s.o.p(a);
}
```

(or)

```
import java.util.Scanner;
class A {
    psvm ( ) {
        Scanner sc = new Scanner (System.in);
        s.o.p("Enter a no");
        int n = sc.nextInt();
        int sum = 0;
        for (int i=1 ; i<=n ; i++) {
            sum = i;
        }
        s.o.p (sum);
    }
}
```

```
import java.util.Scanner;
class Join{
    psvm(String s=" ");
    Scanner sc=new Scanner(System.in);
    S.O.P("Enter a no");
    int a=Sc.nextInt();
    for(int i=1; i<=a; i++){
        s=s+i
        if(i<a){s=s+"+";}
    }
    S.O.P(S)
}
```

o/p → a=4
1+2+3+4

```
class B{
  psvm(—){
  for(int i=1; i<=10; i++){
  S.O.P(i)
  if(i%3==0) break;
  }}
```

o/p: 1 2 3

```
import java.util.Scanner;
class Join{
    psvm(—){
    String s=" ";
    Scanner sc=new Scanner(System.in);
    S.O.P("Enter a no");
    int a=Sc.nextInt();
    int sum=0
    for(int i=1; i<=a; i++){
        sum=sum+i;
        s=s+i
        if(i<a) s=s+"+";
    }
    S.O.P(S)
    S.O.P(sum);
}}
```

```
class B{
  psvm(—){
  for(int i=1; i<=10; i++){
  if(i%3==0)
  break;
  S.O.P(i)
  }}
```

o/p: 1 2

```
s=s+"="+sum;
s.op(s);
```

① write a prog. to print multiplication table for a given no.

→ import java.util.Scanner;
class —
String

② w.a.p to print product of 1 to n. natural nos.
factorial

1X2=2
2X3=

continue :

```
class B{
  psvm {
   for (int i=1; i<=10; i++){
      if (i%3 == 0)
        continue
      s.o.p(i);
   }}
```

```
int n = 67324
for(i=1; n>0; n/=10)
{
s.o.p(n%10)
}
```

```
while (n>0)
{
s.o.p(n%10);
n /= 10;
}
```

o/p

4
2
3
1
6

```
int n = 123
int sum=0;
while (n>0){
   sum += (n%10);
   n /=10;
}
s.o.p (sum);
```

o/p

→ 123
6

```
int sum=0;
for(int i=153; i>0; i/=10).
{
 sumt = i%10;
}
```

③ add only even digits in a number

```
import java.util.Scanner;
class factor{
  psvm (——) {
    Scanner sc = new Scanner (System.in)
    s.o.p ("Enter a no");
    int n = sc.nextInt ( );
    for (int i=1; i<=n; i++){
       n%i == 0
       s.o.p (i);
  }}
```

o/p = 6
→ 1,2,3,6

```
o/p if (n%i==0)
      s.o.p(n);
```

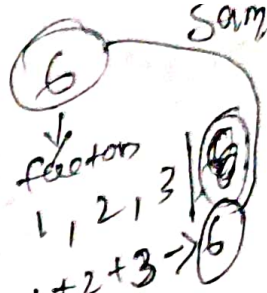
```
int n=6
for (int i=1; i<=n/2; i++)
{ if (n%i==0)
    s.o.p (i);
}
```

o/p

→ 1,2,3

```
import java.util.Scanner;
class perfect{
  psvm (---){
  Scanner sc = new Scanner (System.in);
  S.O.p ("Enter a no");
  n = sc.nextInt();
int sum = 0;
for (int i=1; i<n; i++){
if (n%i==0)
        sum = sum+i;
if (sum==n)
c.opprint ("perfect");
else
  s.Op ("not perfect");
}
```

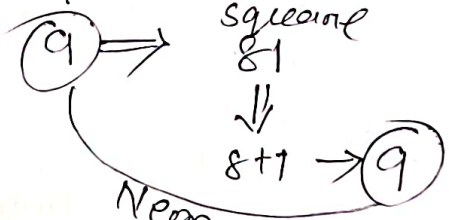
same



Enter a no
6
perfect no

---

e. a Java query to print neon number.

neon → 

```
int n = 9;
int sq = n*n;
int sum = 0;
for (i=0; i<n; i++){
  n = n*n;
  if (sq%i==0)
     sum = sum+i;
  }
if (sum==sq)
S.O.p ("neon")
else
  S.O.p ("not neon");
}
```

```
int n=9;
int sq=n*n;
int sum=0;
// while (sq>0)
{
int n = sq%10;
sum+=n;
sq/=10;
}
if (sum==n)
  S.O.p ("neon")
else S.O.p ("not neon");
```

Q:- wo a q. spy number or not.

Spy number → 123 → 1+2+3 = 6
→ 1×2×3 = 6

```
int mul=1;
int n=123;
int sum=0; int m=n;
while (n>0){
int r=n%10;
sum+=r;
mul*=r;
n/=10;
}
if (sum==mul)
{
s.o.p("spy no")
}
else s.o.p("not spy no"+m);
```

① Spy no & neon no. using for loop.
② Revers a no.

30-07-24

do-while loop:-

need this.{}

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

Ex:-

```
int i=1;
do{
s.o.p(i);
i++;
}
while(i<=3);
```

op 1 2 3

(or)

```
int i=1;
while(i<=3)
{
s.o.p(i);
i++
}
```

op 1 2 3

```
int n= 1246
Sum=0
do while (n>0)
{
Sum+= n%10
n/=10
S.o.p(sum)
while (sum<=9);
```

sum 6 6 6 16/10

1+2+4+6

13

1+3

4

6==0 6>9

n=6

sum=0

```
int n=3826;
int sum=0;
while (n>0)
{
sum+ =n%10;
n/=10;
if (n==0 && sum>9)
{ S.o.p(sum);
n=sum;
sum=0;
}
}
S.o.p(sum);
```

6+2+8+3

n=3826 0==0 19>9

Sum=0+6+2+8+3=19

update n=19 0

Sum=0+9+1=10

0==0 && 10>9

update n=10

Sum=0+0+1=1

n==0 && Sum>9

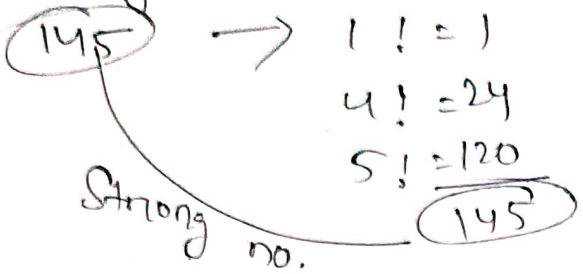
0==0 && 1>9 false

out of the loop.

OR

```
int n=3826;
int sum=0;
do{
while (n>0)
{ sum+=n%10;
n/=10
}
n=sum;
sum=0;
S.o.p(n);
}while (n>9);
```

Strong number — sum of the factorial of each digits if matching to same no.




```
int n=145;
Sum = 0;
while (n>0)
    n= n%10;
```

```
import java.util.Scanner;
class Strongno{
  psvm(——){
  Scanner sc = new Scanner(System.in);
  S.O.P("Enter a no");
  int n = sc.nextInt();
int int temp=n;
int sum=0;
while (n>0) {{
        int d=n%10;

      int prod=1;
      for (int i=1; i<=d;i++){
          prod *=i;
      }
      Sum + =prod;
      n/=10;
  }
  if (temp == sum)
  S.O.P ("Strong no");
  else
    S.O.P ("not strong no");
}}
```

O/P
→ 145
Strong no.

Prime no. :-

```
int n = 2 ; if (n<=1) { S.o.p ("special no");
int a = 3; else
boolean a = true;
for (int i=2; i<=n/2 ; i++)
  {
    if (n%i==0) {
        a = false;
        S.o.p ("not prime")
        break;
    }
  }
if (a)
  S.o.p ("prime");
}
```

or//

```
if (n%i==0){
  a = false;
  break;
}
if (a)
 S.o.p ("prime");
else
 S.o.p ("not prime");
}
```

Armstrong no :-

```
import java.util.Scanner;
class anm {
  psvm (→ {
Scanner sc = new Scanner (System.in);
S.o.p ("Enter a no");
int n = sc.nextInt();
    int temp = n;
    int count = 0;
    while(n>0) {
        count++;
        n/=10;
    } S.o.p ("count = " +count);
    n= temp;
    int sum = 0;
    while (n>0) {
        int d = n%10;
        int prod = 1;
```

```
for (int i=1; i<=count;
        i++) {
    prod *= d;
}
sum += prod;
n/=10;
}
S.o.p (sum)
if (temp==sum) {
S.o.p ("Arm");
else
 S.o.p ("not anm");
}
```

① check no is automorphic no. or not.

5 → $5^2 = 25$

Same = automorphic

25 → 625

② number is palindrome or not (1221 → reverse 1221)

③ convert decimal no to binary.

④ convert binary to decimal no.

⑤

—•—

palindrome

```
int n = 121;
int temp = n;
int res = 0;
while (n > 0) {
  int rem = n % 10;
  res = (res * 10) + rem;
  n /= 10;
}
if (res == temp)
  s.o.p ("palindrome");
else
  s.o.p ("not a palindrome");
}
```

—o—

automorphic no

```
int n = 25;
int sq = n * n;
boolean flag = true;
while (n > 0) {
  if (n % 10 != sq % 10) {
    flag = false;
    break; }
  n /= 10;
  sq /= 10;
}
if (flag)
  S.o.p ("automorphic")
else
  S.o.p ("not automorphic");
```

Decimal to binary

```
int n = 20;
String res = " ";
while (n > 0) {
    res += (n % 2);
    n /= 2;
}
for (int i = res.length() - 1; i >= 0; i--) {
    s.o.p (res.charAt (i));
}
```

Binary to decimal —

```
int n = 10100;
int count = 0;
sum = 0;
while (n > 0) {
    int rem = n % 10;
    if (count == 0 && rem == 1) {
        sum += 1;
    } else if (rem == 1) {
        int sq = 1;
        for (int i = 0; i < count; i++) {
            sq *= 2;
        }
        sum += sq;
    }
    count++;
    n /= 10;
}
s.o.p (sum);
```

Decimal to binary :-

```
String s = " ";
int n = 58;
while (n>0) {
    int bit = n%2;
    // s.o.p (bit);
    s = bit + s;
    n/=2;
}
s.o.p (s);
```

0 1 0 1 1 1

⟶ (111010)

n = 58 29 14 7 3 1 0

0 1 0 1 1 1

bin 111011

i = 100000

n = 59 29 14 7 3 1 0

```
int n = 58;
int bin = 0;
int i = 1;
while (n>0)
{ int bit = n%2;
  bin = bit * i + bin;
  i *= 10;
  n/=2;
}
s.o.p (bin);
```

—o—

bin to decci

```
int bin = 1101101;
int dec = 0;
int i = 1;
while (bin>0)
{ dec += (bin%10) * i;
  i *= 2;
  bin /= 10;
}
s.o.p (dec);
```

Nested loop:-

```
class A {
    psvm (——) {
        for (i=1 ; i<=5 ; i++)
        { for (j=1 j<=5 ; j++) {
            s.o.p (i + " " +j) ;
            if (j%3==0) break;
            s.o.p (i + " " +j) ;
            // if (i%3==0) break ;
            // if (i==j) break;
        }
    }
}
```

O/P

```
11
12
13
.
.
.
55
```

```
11
12
13
```

```
- for(i=1; i<=5 ; i++)
  { if (i%2==0) continue;
      for (int j=1; j<=5; j++) {
          s.o.p (i+" "+j);
      }
  }
```

```
11
.
.
15
31
.
.
55
```

Prime no in a given range——

```
import java.util.Scanner;
class prime{
    psvm (—) {
    Scanner sc = new Scanner (System.in);
    s.o.p("Enter a range");
    int a = sc.nextInt();
    int b = sc.nextInt();
// boolean flag = true
    for (int i=a; i<=b; i++){
        if (i==1) continue;
        boolean flag = true;
        for (int j=2; j<=i/2; j++){
            if (i%j==0){
                flag = false;
                break;
            }
        }
        if (flag)
        s.o.p(i);
    }
}
```

```
s.o.p("prime nums are");
s.o.p("-------");
for (int i=b; i>=a; i--)
```

Largest prime no. in a given range

```
int a = 20
int b = 30
=
=
same
if (flag){
    s.o.p(i);
    break;
}
}}
```

① w.a. prog to print 2nd largest no in a given range

② w.a prog to " alternative prime no " " " "

2, ~~3~~, 5, . — —
dont print

③ To display sum of all prime no. in a " "

④ w. a prog. to display ~~any~~ strong no. in a given range.

⑤

Fibonacci series

```
int n = 15;
int a=0;
int b=1;
int c;
for(int i=1; i<=n; i++){
    s.o.p print(a+" "); // SOP(a);
    c = a+b;
    a = b;
    b = c;
}
```

— o —

9-08-24

```
class a int [12b] a = SC.nextInt();
      int b = " " " ";
for (int i=a; i<=b; i++){
    int n=i;
```

```
int count = 0;
while (n>0) {
    count++;
    n/=10;
}
n=i;
int sum=0;
while (n>0){
    int digit = n%10;
    int prod=1;
    for (int j=1; j<=count; j++){
        prod *= digit;
    }
    sum += prod;
    n/=10;
}
if (i==sum)
S.O.P ("armstrong", (i);
}}
```

2nd largest no —o—

```
int a = sc.nextInt();
int b = sc.nextInt();
int count=0;
for (int i=b; i>=a; i--){
    count++;
    S.O.P (i);
if (count==2)
S.O.P (i);
}}
```

= 1

Star pattern

```
import java.util.Scanner;
class SquareMat{
  PSVM (——){
    Scanner sc = new Scanner (System.in);
    S.O.P ("Enter Site");
    int site = sc.nextInt();
  for (int i=1; i<=site; i++){
      for (int J = 1; J <= Site; J++){
          S.O.P ("*");
        }
        S.O.P( );
    }
}
```

o/p (site = 3)

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

—o—

o/p

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

```
Import Java
class pattern {
  PSVM (—) {
  for (int i=1; i<=5; i++){
    for (int J=1; J<=i; J++){
        S.O.P ("*");
    }
    S.O.P( );
  }
}
```

or

```
for (int i=5; i>=1; i--)
  for (int J=5; J>=i; J--)
```

```
* * * * *
```

```
for (int i=1; i<=5; i++)
for ( int J=5; J>=i; J—)
```

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

```
int letters=5;
for (int i=1; i<=5; i++)
    for (int J=1; J<=letters; J++)
        S.O.P("*");
    S.O.P( );
    letters--;
```

try it

```
for (int i=5; i>=1; i--)
for (int j=1; j<=size; j++)
    S.O.P("*");
  }
  S.O.P( );
}
```

```
for (int i=1; i<=5; i++)
  for (int j=5; j>=i; j--)
    if (j==i)
    S.o.p("0");
    else
     S.o.p("x");
    }
    S.o.p();
```

o/p

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

——o——

```
int n=5;
for (int i=1; i<=n; i++)
{for (int j=n; j>i; j--)
 {System.out.print("x");
 }
 S.o.p(i);
 S.o.p();
}
```

① 1 1 1 1 1
2 2 2 2 0
3 3 3
2 2
1 0

② 1
2 3
4 5 6
7 8 9 10

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

④ Spiral no

1 2 3 4 5
10 9 8 7 6
11 12 13 14 15
20 19 18 12 16
21 22 33 24 25

⑤ 1
3 2
4 5 6
10 9 8 7
11 12 13 14 15

Ans ⑤

```
int n=5;
for (int i=1; i<=n; i++)
if (i%2!==0){
 int x=i-1;
   x=(x*(x+1))/2;
```

```
for (int j=1; j<=i; j++){
 x++; S.o.p(x+"\t");
 }
else {int y=(i*(i+1))/2;
for (int j=1; j<=i; j++){
   S.o.p(y+"\t");
   y--;
 }
 S.o.p();
}
```

① int rows=5;
for (int i=1; i<=rows; i++){
for (int j=1; j<=rows-i+1; j++){
S.o.p
i%2==1 ? 1 : 2
S.o.p(i + " ");
}
S.o.p();
}

or// S.o.p((i%2==1 ? 1 : 2)+" ");

② int no=1;
int rows=4;
for (int i=1; i<=rows; i++){
for (int j=1; j<=i; j++){
S.o.p(no + " ");
no++;
}
S.o.p();
}

③ int n=5;
for (int i=1; i<=n; i++){
S.o.p(i + " ");
}
S.o.p();
for (int i=n; i>=1; i--){
S.o.p(i + " ");
}
S.o.p();
}

④ int n=5;
for (int i=1; i<=n; i++){
S.o.p(i + " ");
}
S.o.p();
for (int i=10; i>=6; i--){
S.o.p(i + " ");
}
S.o.p();
}

or//
int n=5;
for (int i=1; i<=n; i++){
if (i%2!=0){
int x=((i-1)*n)+1;
for (int j=1; j<=n; j++){
S.o.print(x + "\t");
x++;
}
}else{
int x=i*n;
for (int j=1; j<=n; j++){
S.o.print(x + "\t");
x--;
}
S.o.p();

⑤ int x,y;
x=1;
y=x+(n-1);
x=y+1;
SOPLN

HW

① 5 * 4 * 3 * 2 * 1
5 * 4 * 3 * 2
5 * 4 * 3
5 * 4
5

② x x x x x
x - - - x
x - - - x
x - - - x
x x x x x

③ x
x x
x - x
x - - x
x - - - x
x x x x x x

②
```
int n=5;
for(int i=1; i<=n; i++){
for(int j=1; j<=n; j++){
if(j==1||j==n||i==1||i==n)
s.o.print("x");
else
s.o.p("-");
}
s.o.p();
}
```

③
```
int n=5;
for(int i=1; i<=n; i++){
for(int j=1; j<=i; j++){
if(j==1||j==i||i==n)
s.o.p("x");
else
s.o.p("-");
}
s.o.p();
}
```

```
int n=5;
for(int i=1; i<=n; i++){
for(int j=1; j<=n-(i+1); j++){
```

①
```
int n=5;
for(int i=1; i<=n; i++)
{ for(int j=5; j>=i; j--)
{ String s=" ";
s=s+j;
if(i!=j){
s=s+"*";}
s.o.p(s);
}
s.o.p();
}
```

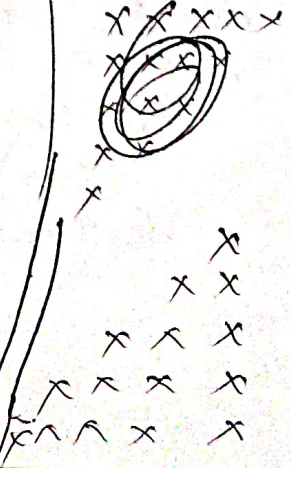
```
int n=5; int spaces=n-1;
for(int i=1; i<=n; i++)
{ for(int j=1; j<=spaces; j++)
{ s.o.print(" ");
for(int j=1; j<=i; j++)
{ s.o.p("*");
}
s.o.p();
spaces--;
}
```

→ none-eq

// for(int j=1; j<=n-i; j++)

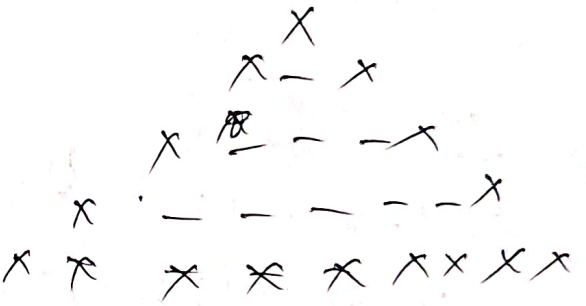
→ no need

bll

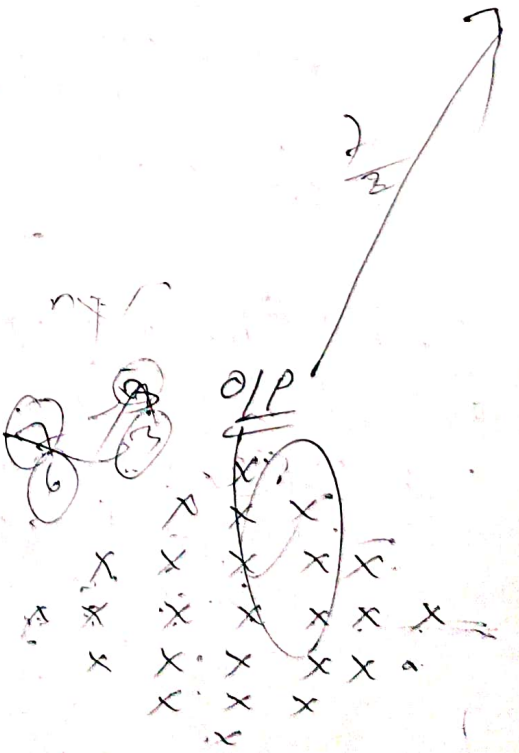


```
int n=5; int spaces=n-1; int stars=1;
for (int i=1; i<=n; i++)
{
  for (int j=1; j<=spaces; j++){
    S.O.P(" ");
  }
  for (int j=1; j<=stars; j++){
    S.O.P("*");
  }
  S.O.P();
  spaces--;
  stars+=2;
}
```

```
int n=5;
int s=n-1
int st=1;
for (int i=1; i<=n; i++){
  for (int j=1; j<=s; j++){
    S.O.P(" ");
  }
  for (int j=1; j<=st; j++){
    S.O.P("*");
  }
  if (j
```



```
int n=5; int spaces=n-1; int stars=1;
for (int i=1; i<=n; i++){
  for (int j=1; j<=spaces; j++){
    S.O.P(" ");
  for (int j=1;
```



```
int n = 7; int stars = 1;
int spaces = n/2;
for (int i = 1; i <= n; i++){
  for (int j = 1; j <= spaces; j++){
    S.O.P (" ");
  }
  for (int j = 1; j <= stars; j++){
    S.O.P ("*");
  }
  S.O.P ( );
  if (i <= n/2) {
    spaces --
    stars +=2;
  }
  else
    spaces ++;
    stars -=2;
}
```

O/P

	spaces	stars
i=1	3	1 → 2
i=2	2	3 → 2
i=3	1	5 → 2
i=4	0	7 → 2
i=5	1	5 → -2
i=6	2	3 → -2
i=7	3	1 → -2

i = 1,2,3 → spaces--;

i = 4,5,6 → spaces++;

if (i <= n/2) → stars += 2;

else stars -= 2.

```
int n=7; int stars=1;
int spaces = n/2
  ~~~
  same
for (int j=1; j<=stars; j++){
  if (j==1 || j==stars)
    S.O.P ("*");
  S.O.P (" ");
}
```

if (j==1 || j==stars || j==(n/2)+1)

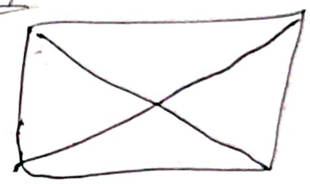
if (j==1 || j==stars || j = n/2 +1 || stars = n/2 +1).

```
int n= input;
int spaces= ;
int stars=1;
for(int i=1; i<=n; i++){
  for(int j=1; j<=n; j++){
  if(i==1 || i==n || j==1 || j==n || i==j || i+j==n+1){
  { s.o.p("X");
  }else{
     s.o.p(" ");
  }
  }
  s.op();
}}
```

if(i==j || i+j==n+1)

or i==n-i+1

O/P



— o —

```
int n= - input;
int spaces= n-1;
for(int i=1; i<=n; i++){
  for(int j=1; j<=spaces; j++){
    s.o.p(" ");
  }
  for(int j=1; j<=i; j++){
    s.o.p(j);
  }
  for(int j=i-1; j>=1; j--){
    s.o.p(j);
  }
  s.o.p();
  spaces--;
}}}
```

O/P n=4

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

Raman sir

05-06-24

```
row → for(int i=1; i<=n; i++)
column → for(int j=1; j<=n; j++)
  if(i==1||j==n||(i<=n/2 ...)||(i>=n/2 ...) i+j==6)
    s.o.print("X");
  else
    s.op(" ");
  }
  s.op()
```

(1,1)X
(2,1)X
(3,1)X
(4,1)X
(5,1)X

if(j==1 || j==n || j==i+1){

Array :-

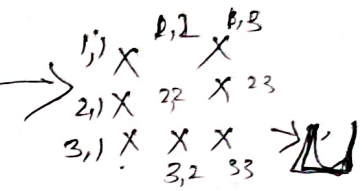
```
int [] a = {10, 20, 30, 40};
S.o.p(a[2]);
S.o.p(a.length);
for (int i=0; i<a.length; i++)
  { S.o.p(a[i]);
  }
```

o/p

```
for (int n : a){
    S.o.p(n);
}
```

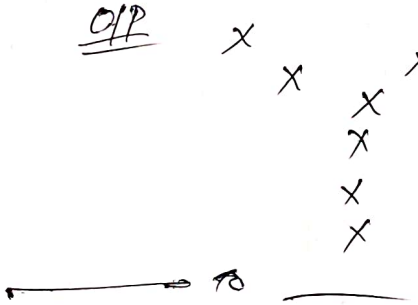
05-08-24

```
if (i==1 || j==1 || i==n)
```

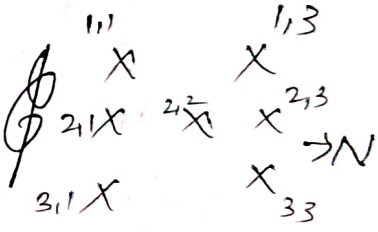



```
if (i<=n/2) && (i==j || i+j==6) || (i>=n/2 && j==3)
```

o/p



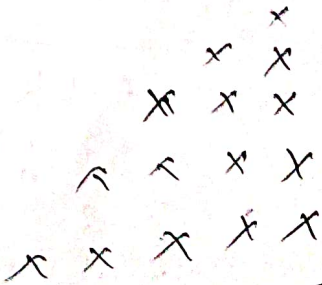
```
if (j==1 || j==0 || j==i)
```



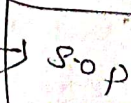

```
for (int i=0; i<6; i++)
  for (int j=0; j<7; j++)
    if (i==0 && j%3==0) || i==1 && j%3==0) || (i+j==8) || (i-j==2))
      print("*");
    else
      s.o.p(" ");
  s.o.p
```

o/p

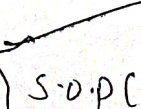
```
int n=5; int sp=n-1; int st=1;
for (int i=1; i<=n; i++)
  for (int j=1; j<=sp; j++)
    s.o.p(" ");
  for (int k=1; k<=st; k++)
    s.o.p("* ");
  sp--;
  st++;
```



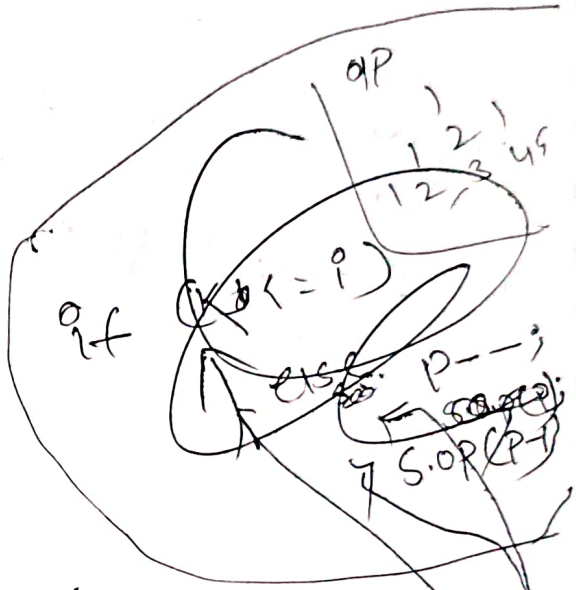
```
s.o.p(i);
```



```
s.o.p(k);
```



```
int n=5; sp=n-1
         st=1;

for(int i=1; i<=5; i++)
 for(int j=1; j<=sp; j++) S.o.p(" ");
  for(int k=1; k<=st; k++) s.op('x');
   S.op();
   sp--;
   st+=2;
```



OR//

```
class S
{ psvm( ) }
  int n=5;
  int sp=n-1;
  int st=1;
 for(int i=1; i<=n; i++)
 { for(int j=1; j<=sp; j++){ S.o.p(" "); }
     int p=1;
   for(int k=1; k<=st; k++)
    {
      S.op(p);
      p++;
    }
  S.op();
  sp--;
  st+=2;
```

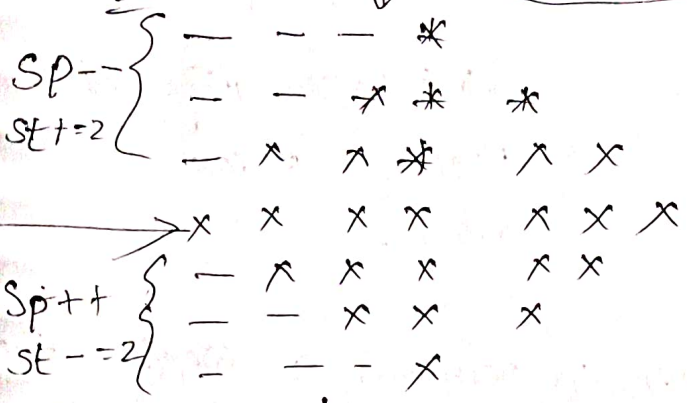
O/P →
1
1 2
1 2 3
1 2 3 4

```
for(int i=1; i<=n; i++)
 for(int j=1; j<=sp; j++) S.o.p(" ");
     int p=1;
  for(int k=1; k<=st; k++)
     if(k<=i) S.op(p); p++;
     else p--; S.o.p(p-1);
   S.op();
   sp++; st-=2;
```

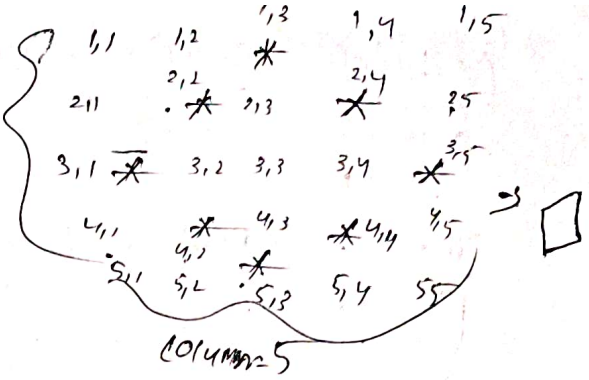
O/P

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

o/p



See how many spaces & stars in 1st row
SP = 3 & n/2 star = 1 row = 5



for even number

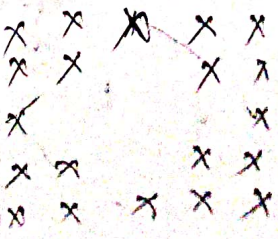
```
int n=7;
int sp = n/2
int st = 1
if (n%2==0) n+1;
for ( int i=1 ; i<=n ; i++) S.O.print("  ");
for (int j=1 ; j<=sp ; j++)
for (int k=1; k<=st; k++){S.O.p("*");
}
S.O.p ( );
if (i<=n/2) { sp--;
              st+=2;
            }
else { sp++;
       st-=2;
     }
```

change

same

```
for (int i=1 ; i<(n-1); i++){ S.O.p("*");
for (int j=1 ; j<=sp ; j++) S.O.print ("*");
 for (int k=1 ; k<st; k++) { S.O.print (" ");
}
for (int j=1 ; j<=sp ; j++) S.O.p ("*"); → for right side star
 if (i<n/2){ sp--; st+=2; }
  else { sp++; st-=2; }
```

o/p



public static void main (String args[])

public → access modifier
static → non-access modifier
void → keyword of returntype
main → method name
(String args[]) → method arguments.

{
stmt
}
→ method body

blue color: Keyword
black
red =

*Method:-

→ A method is a set of stmts to perform a specific task.
→ It will get executed when it is called.

Syntax:-

access-modifier modifier returntype method name (method arguments)
(optional) (optional)

{
stmt
}

premitive datatypes
non " " = no. of

Exm:-

```
class Demo
{
  public static void m1()
  {
    S.o.p("M1");
  }
  public void m2(){
    S.o.p("M2");
  }
  psvm(---){
    S.o.p("Main start$")
    Demo.M1(); // method calling start.(static)
    Demo d = new Demo();
    d.m2(); // method calling stmt (non static)
    S.o.p("Main ends");
  }
}
```

→ while creating a method access modifier and modifier are optional, return type & method name are mandatory

predefined func^n in Java

→ nextInt(), nextLine(), nextDouble() - - -

→ we can call a method "n" no. of times.
→ whenever we call a static method inside a class donot need class name (Demo.add)

```
import java.util.Scanner;
class Demo{
    static void add(){
        Scanner s=new Scanner(System.in);
```

→ we can call a method "n" no. of times
→ whenever we call a static method within the same class, class name not required. (Demo.add())

Exm

```
import java.util.Scanner;
class Demo{
    static void add(){
        Scanner s=new Scanner(System.in);
        s.o.p("Enter two nos");
        int a=s.nextInt();
        int b=s.nextInt();
        s.o.p(a+b);
    }
    psvm(){
        s.o.p("Main method");
        add();
    }
    add();
}
```

Types of methods →

2 types -

* pre-defined method , * user-defined method.

* pre-defined method :-

→ This is the method which is already developed

ex:- nextInt(), println(), etc.

* user-defined method :-

which is developed by user or programmer.

Types of method based on modifier →

2 types —

* static or * non-static.

* static method :-

→ which methods are having "static" keyword as a modifier that methods are called as "static method."

→ we can call this by using class name.

* non-static method :-

→ Which methods are not having "static" keyword as a modifier that methods are called as "non-static method"

→ we can call this by using object reference

07-08-24

Types of method based on parameters —

2 types

* No-argument . * parameterized

PSVM1() → No arg. method — PSVM2(int a)

{ } { }

*No-arg. method:-

→ which method does not have any formal arg. then method known as no-arg. method.

Exm
```
psvm1()
{
}
```

*parameterized method:-

→ which method having a formal arg. that method is called as parameterized method.

→ we can call parameterized method by passing matching parameters. (same datatype)

→ TO call this method no. of formal arg & no. of actual arg. should be same.

```
psvm1(int i)        ← Formal arg.
{
  Stmt
}
m1(17);             ← Actual arg.
```

#Note

→ In Java we should not create a more than one methods with same name & same formal arg.

Exm

```
method declaration
psvm1(int i)        ← method signature
{                   ← method defination
  Stmt
}
```

blue colon = Keyword

```
Class A{
  public void m1(){
    S.o.p("Non-static");
  }
  psv m2(int i, int j){
    S.o.p("static" +i+ " " +j);
  }
  psvm(....){
    A t = new A();
    t.m1();
    // A.m2(10); //CTE
    // A.m2(10,2.5) //CTE
    A.m2(20,10);
    A.m2(20-10,5);
  }
}
```

—o—

```
Class B{
  public static void m1(){ S.o.p("Static-M1()"); }
  public static void m2(int i){
    S.o.p("Static_M2(int i)" +i);
    M1(); // method call
  }
  public static void m3(int i, int j){
    S.o.p("static M3(int i, int j)" +i+ " " +j);
    M2(i+j); // method call
  }
  psvm(String args[]){
    S.o.p("Static M3(int i, int j
    M3(10,20);
  }
}
```

→ If returntype is void then it will not returning anything.

08-06-24

Method return type:-

Exm ①
```
psv int m1(){
    S.o.p("M1() is executed");
    return 5;
}
psv main(←)
{
    int n = m1();
    S.o.p(n);
}
```
O/P
M1() is executed
5

```
S.o.p(m1());
```
→ O/P 5 but the main method is printing this value.

②
```
Class A {
    public static void M1(){
        S.o.p("M1 is executed");
    }
    public static int m2(){
        S.o.p("M2 is executed");
        return 5;
    }
psvM(→){
    m1();
    int n2 = m2();
    S.o.p(n2);//5
}
}
```
O/P
M1 is executed
M2 is executed
5

→ we need for storing the return type value. neither it can't be print.

③
```
public static double m3(double d){
    S.o.p("M3 is executed");
    return d;
}
psvM(→){
    S.o.p(m3(7.3));//M3 is executed
}
```
O/P
7.3

This 7.3 will store in d)

④
```
public static int add(int i, int j){
    return i+j;}
psvM(→){
    add(5,5);//NO O/P
    S.o.p(add(20,10));//30
}
```
```
int res=add(20,30);
S.o.p(res);//50
}
```
O/P
30
50

```
public static int square (int i){
    return i*i;
}
psvm (——){
    square(5);
    s.o.p (square (5)); [or] int n = square(5);
                             s.o.p (n);
    }
}
```

O/P
25

—o—

```
public static boolean check (int i){
    return i ;
}
psvm (—) {
    int n = check(5)
    if (n % 2 == 0)
        s.o.p ("even");
    else
        s.op ("odd");
}
```

```
public static boolean
iseven (int n){
if (n%2==0)
    return true;
else return false;
}
psvm (—) {
    s.o.p (iseven (8));
}
}
```

—o—

~~public static int~~

Control transfer stmt are 3types —

① break ② continue ③ return.

```
psvm (—)
{
for (int i=1; i<=10; i++)
{ if (i==5)
  {
    break;
  }
  s.o.p(i);
}
}
```

O/P
1
2
3
4

```
psvm (—){
for (int i=1; i<=10; i++)
{ if (i==5)}
    continue;
}
s.o.p (i);
```

O/P
1
2
3
4
6
7
8
9
10

```
psvm1(—){
s.o.p("M1 is executed");
return ;
(s.o.p("M1 is end");)
}
psvm(—){
M1 ();
}
}
```

error bcz we cannot write anything after return keyword/ unreachable stmt.

Method return type :-

→ It will indicate which type of values the method will return after the execution ..to its caller method.

→ If the return type is void the method will not return any values to the its calter method.

→ If ~~the~~ method return type is other than void it is mandatory to return the value using "return" keyword.

→ return is a keyword & control transfer statement.

→ After the return stmt we should not declare any other stmts.

```
class A{
  ps vm(--){
   s.o.p("starts");
   for(int i=1;i<=10;i++)
     {if(i==5)}
        return; → this keyword control the JVM so next prog will not be executed
   }
   s.o.p("ends"); → It will not execute
}}
```

O/P
starts
1
2
3
4

But if we use break keyword then the o/p is →
starts
1
2
3
4
It will executed → ends

09-08-24

~~class primenum{~~
~~public static boolean isprime(int n){~~

→ If a method return type is otherthan void in method if we are using any conditional stmt it is mandatory to return the values.

→ Advantage of method = "recusability".

```
import java.util.Scanner;
class primenum{
 public static boolean isprime(int n){
   if(n<=1) return false;
   for (int i=2; i<=n/2; i++) //OR// for(int i=2; i<n; i++)
   {
     if (n%i==0)
      { return false
      }
     return true
 }
 public static void main (String [] args){
     Scanner sc = new Scanner (System.in);
     System.out.println("Enter a no");
     int n = sc.nextInt();
     if (isprime(n)) {
        S.O.P(n+" is a prime no");
     }
     else{
       S.o.p (n+" not a prime no");
     }
 } }
```

```
int st = 10
int end = 100
for (int i=st;
     i<=end; i++)
{ if (isprime(i))
  { s.op (i);
  }
}
```

prime no with in range —

range 1 to 12

off
1, 2, 3, 5, 7, 11

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

Same

```
    return true;
 }
 public static void range(int st, int end)
 { for (int i=st ; i<=end ; i++)
    { if (isprime (i)){
        S.o.p (i); }}
 psvm(——)
```

Same

```
s.op("enter range");
int st=sc.nextInt;
int end=sc.nextInt;
range (st, end);
```

Highest no. of prime no in given range.

Same ≡

```
public static void range (int st, int end){
    for (int i=end; i>=st; i--)
    { if (isprime (i)) {
            s.o.p (i);
            break;
    } }
psvm( ——— )
```

≡ Same

range: 1 to 12

O/P: 11

— o —

2nd highest no. in given range of prime no. —

```
psvrange (int st, int end){
    int count=0;
    for (int i=end; i>=st; i--)
    { if (isprime (i))
      {
        count++;
        if (count==2) {
            s.o.p (i);
            breaks
    } } }
psvm (—) {
```

O/P: 7

Same

— o —

alternate prime no. in a given range.

```
public static void range (int st, int end){
    int count=0;
    for (int i=st; i<=end; i++){
    { if (isprime (i)){
        count++;
        if (count%2!=0)
        {s.o.p (i);
    } } }
```

O/P: 2, 5, 11

perfect no

```
int n=6;
int sum=0;
for(int i=1; i<=n/2; i++)
 { if (n%i==0)
   sum=sum+i;
 }
if (n==sum)
 {
  S.o.p("perfect");
 }
else
 {
  S.o.p("not perfect");
 }
```

```
class isperfect {
public static boolean isperfect(int n)
 {
  int sum=0;
for(int i=1; i<=n/2; i++){
  if (n%i==0)
  {sum+=i;}
  if(n==sum) return true;
  else return false;
 }
PSVM (—){
 Scanner s=new Scanner(System.in);
 S.o.p("Enter a no");
 int n = s.nextInt();
S.o.p(isperfect(n));
}
```

range —

```
public static void range(int st, int end)
{for(int i=st; i<=end; i++)
   {
    if(isperfect(i)) {
     S.op(i);
    }
   }
}
PSVM (—) {
 Scanner sc = new Scanner(System.in);
 S.o.p("Enter a range");
 int st = sc.nextInt();
 int end = sc.nextInt();
  range(st, end);
}
}
```

Hw

prime
perfect
Strong
Armstrong
pall
automorphic
Neon
spy
Fibonacci
Happy -

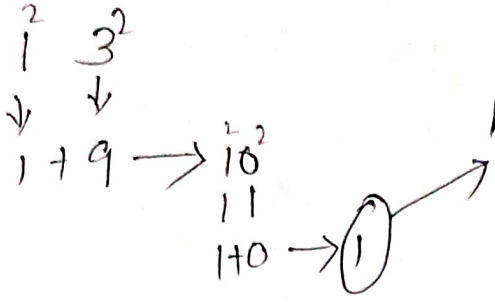
highest no -

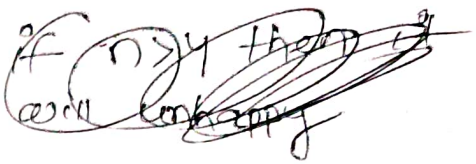
```
psv range(int st, int end){
for(int i=end; i>=st; i--){
 if(isperfect(i)){
   S.op(i);
   break;
}}}
```

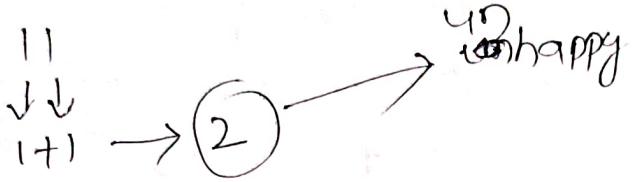
alternate

```
psv range(int st, int end){
   int count=0;
for(int i=st; i<=end; i++){
   if(isperfect(i)){
      count++;
    if(count%2!=0){
      S.o.p(i);
}}}
```

Happy no — 13








```
class Happyno{
    public static int happy(int n){
    {
        int sum=0;
        while(n>0)
        { int r=n%10;
          sum+=r*r;
          n/=10;
        }
        return sum;
    }
    ps boolean ishappy(int n)
    {
        int m=n
        while(m!=1 && m!=4)
        {
            m=happy(m);
        }
        if(m==1) return true;
        else return false;
    }
    psvm(--){
    if(ishappy(i)){
    s.o.p(n+" is a happy no");}
    else{
    s.o.p(n+" is not a happy no");}}}
```

range

```
public static void range(int st, int end){
for(int i=st;i<=end;i++)
  {if(ishappy(i))
    {
      s.o.p(i);
    }
  }
psvm(--){

range(st,end);
}
}
```

alternate

```
psv range(int st, int end){
  int count=0;
for(int i=st;i<=end;i++)
{if(ishappy(i)){
    count++;
  if(count%2!=0){
  s.o.p(i);
  }
}}
```

10-08-24

```
class dectobin
{
psvm(---){
int n=12;
int i=1;
int bin=0;
while(n!=0)
{
int r=n%2;
bin+=(r*i);
i*=10;
n/=2;
}
s.o.p(bin);
}
}
```

```
class bintodec
{
psvm(---){
int bin=1100;
int dec=0;
int i=1;
while(bin!=0)
{
int r=bin%10;
dec+=(r*i);
bin/=10;
}
s.o.p(dec);
}
}
```

Hw: ① print to convert dec to octal.

② print to convert octal to dec.

Method overloading:-

Exm

① p.s.int m1()
{
}
p.s.v m1()
{
}

compile time → Error

psvm1(int a)
{
}
psvm1(int i)
{
}

compile time → Error

③ p.int m1()
{
}
psvm1(int i)
{
}

compile time → Sucess

Exm
```
class Demo
{
static void m1(int i){
 S.o.p("M1(int i)");
}
static void m1(double d){
 S.o.p("M1(double d)");
}
static void m1(int i, int j){
  S.o.p("M1(int i, int j)");
}
psvm(——){
//m1(); //CTE
 m1(10);
 m1(10,5);
 m1(10,20);
}}
```

#Note:-

Q what is method overloading?

Ans: In a class creating multiple methods with same name & different formal arguments is known as method overloading.

→ Java matches a method call to the method implementation based on method name & arguments not based on method return type

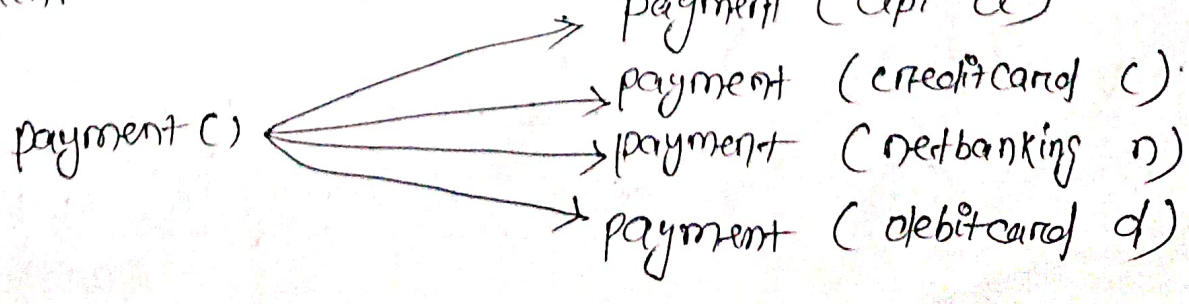
→ We can overload both static & non-static method.

→ While method overloading, a method formal arguments should be different on formal arg. length should be different.

Exm: println() → It is a overloaded method in Java.

#When we go for method overloading? Ans: When we are providing different implementations for the same behaviour when arg. are different.

Exm



Advantages:-

→ By method overloading, we can achieve code-reusability.

→ By method overloading, we can achieve compile time polymorphism.

Can we overload main method?

→ Yes, we can overload main method by passing different formal arg.

→ But JVM calls always which main method having a signature "public static void main" (String arg[])

Exm
```
psvm()  //CTSuccess
{
}
psvm(int i)  // CTS
{
}
psvm(int j)  //CTS
{
}
psvm(String arg[]) // is called by JVM
{
}
```

Is main method is predefined or user defined method?

→ Main method is a user-defined method but we can not change any arg. in main method. If we change any arg. the JVM will not call the main method.

```
class Demo
{
  psvm(--){
    s.o.p(1100); //1100 → it can not convert directly bin to dec.
    s.o.p(010); //8 → bcz it will convert directly octal no.
  }
}
```

psvm1(int...a) → don't give any space

→ variable argument

(we can store an array in a variable)

12-08-24

Variable arguments (var args) :-

```
class Test {
    public static void m1(int a) {
        S.o.p("M1(int) is called");
    }
    public static void m1(int...a) {
        S.o.p(a); ---> it will print address
        S.o.p("M1(int...a) is called");
    }
    psvm(---) {
        M1(10); // 1st m1 method is called.
        M1(10,20); // 2nd M1 method is called.
        M1(50,60,70);
    }
}
```

O/P

M1(int) is called
M1(int...a) is called
M1(int...a) is called

for each loop :-

syntax :- for(datatype variable1 : referencevariable)
{
S.o.p(variable1);
}

print

Variable arguments :-

→ we can pass variable arg. as a method formal argument
→ If a method having var args as an argument, we can call that method by passing "0" no of actual arguments
→ we can write variable args. like this —
syntax → (datatype...referencevariable)
→ when we are writting vari. arg. interanally it will create array with given type
→ we can print array by using for each loop.

For each loop :-

→ It is a type of loop we used to iterate the array or collection.
→ By using this we can print elements only in forward direction.

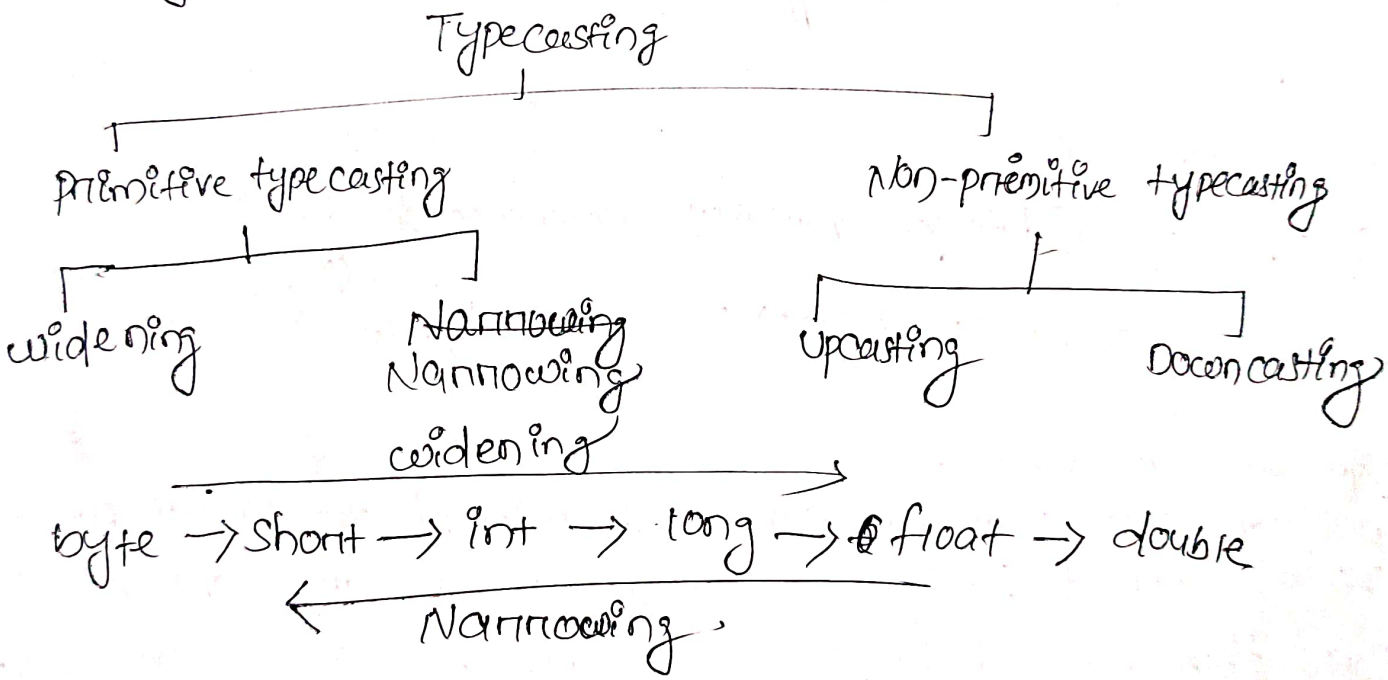
EXM

```
class test{
    public static void m1(int...a){
        for(int n:a){
            sop(n);
        }
    }
    psvm(---){
        m1(20,30,40,50);
    }
}
```

O/P

20
30
40
50

Type casting :-
converting one variable type to another variable type.




```
byte b=10; //1byte=8bits
int a=b; // 4bytes
long c=a; //8bytes
```
widening

```
long b=10;
int a=b;
long c=a;
```
Narrowing

→ The process of converting one datatype into another type is known as typecasting
→ It is a process of converting one variable datatype into another datatype.
→ when we are assigning a values one variable to another variable sometime the two types should be a compatible type.

→ If the two var. type are a compatiable type Java automatically convert to one type to another type

→ when two types are not compatiable type we have to convert one type to another type explicitly.

Primitive typecasting :-

→ The process of converting one primitive type data into another primitive type is known as primitive typecasting.

→ converting var.datatype to another datatype

Types :-

* widening * Narrowing.

* widening :-

The conversion of lower range primitive datatype to higher range primitive datatype. is called widening.

→ compiler allows to convert implicitly lower to higher range type data type

• byte → short → int → long → float → double.

→ It is also called as "implicit typecasting"

The process of con → In this process there is no data loss.

* Narrowing :-

→ The conversion of higher range primitive datatype to lower range primitive type is called "narrowing".

double → float → long → int → short → byte.

→ It is also known as "explicit typecasting"

→ In this process there is a dataloss.

Ex: Class syntax :- (datatype) var.name ;

→ we can achieve narrowing by using typecast operator.

→ when we are converting higher to lower range of data typecast operator is mandatory

Uses :- By typecasting we can achieve memory management.

Note :- we can't convert boolean to other datatype.

end chart

13-08-24

```
class Demo{
    psvm(double d)
    {
    s.o.p(d);
    }
    psvm(---){
     byte b=10; // 1 byte
     int a=b; // 4 bytes (widening)
     s.o.p(a+" "+b);
     s.o.p("=======");
     m1(10);
     m1(9.5);
     m1(8.5f);
     s.o.p(m1('A'));
     m1('5'+6.5);
     s.o.p("====");
    }
    ps int m1(char ch){
        s.o.p(ch);
        return ch; // widening.
    }
}
```

o/p

10 10

10

9.5

8.5

A

65

11.5

```
class Test{
 psvm1(int i, double d){
   s.o.p(i+" "+d);
 }
 psvm1(double d, int i){
   s.o.p(d+" "+i);
 }
 psvm(---){
   m1(10, 10.5);
   m1(10.5, 10);
   m1(10, 10); // (CTE) ambiguous error
 }
}
```

o/p

10 10.5

10.5 10

Ex 1

int a=10; //4 bytes can't convert into 1 byte

byte b=a; (narrowing) //1 byte

↑

it will through an error..

Ex 2

int a=10; //4bytes

byte b=(byte) a; (perfect narrowing //1 byte)

s.o.p(a+" "+b);

It is correct program

— o —

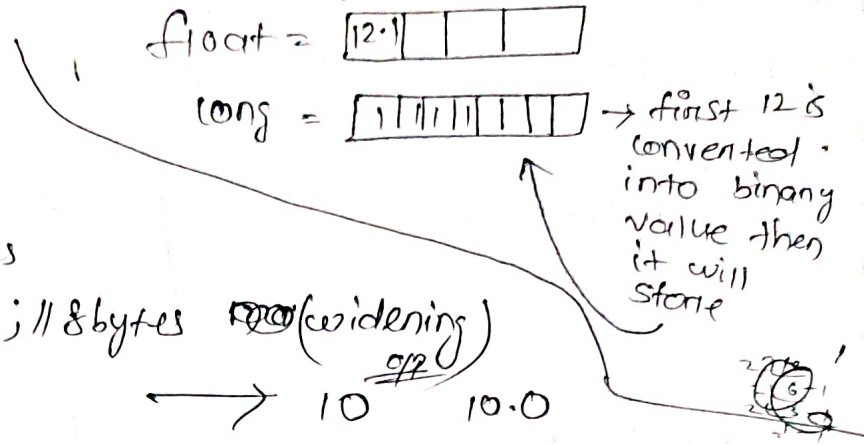
```
class test1{
  psvm(—){
    double d=10.5; // 8 bytes
    int n=(int) d; // 4 bytes (narrowing)
    s.o.p(d+" "+n);      →  10.5   10
                                    data loss (.5)

    int a=68; // 4bytes
    byte ch=(char) a; // 2 bytes (narrowing)
    s.o.p(a+" "+ch);     →  68   D

    int b=130; // 4 bytes
    byte c=(byte) b; // 1 byte (narrowing)
    s.o.p(b+" "+c);      →  130   -126
  }
}
```

byte will store between -128 ... 127 | 128 → -128 | 129 → -127

— o —

```
class test_2{
  ps int m1(double d){
    return (int) d;
  }
  psvm(—){
    s.o.p(m1(9.5)); // 9 (narrowing)
  }
}
```

→ float is bigger than long bcz float is an exponential value. (float will store → 12.1 but long → 12)




```
class test3{
    psvm(--){
        int n=10; // 4bytes
        double d=(double)n; // 8bytes (widening)
        S.o.p(n+" "+d); ⟶ 10 10.0

        float f=10.5f; // 4 bytes
        long l=(long)f; // 8 bytes (narrowing)
        S.o.p(f+" "+l) ⟶ 10.5 10
    }
}
```

— o —

```
class testy{
    psvm(⟶){
        int n=(int) Math.pow(5,3); ⟶ 125
        S.o.p(n);
    }
}
```

o/p

14-08-24

Method reccursion :-

```
class Demo{
    public static void m1(int i){
        if(i==11) return;
        System.out.println(i);
        m1(i+1);
    }
    public static void main(String[] args)
    {
        m1(1);
    }
}
```

```
public static void m1(int i){
// for(int i=10; i>=1; i--)
  {
    if (i==0) return;
      S.O.p (i);
      m1(i+1);
  }
}
psvm(—) {
    m1(1);
}
```

```
if(i==0)
```

m1(10)

—o—

```
psvm1(int i)
{
  if(i==3) return;
  S.O.p(i);
  m1(i+1);
  S.O.p(i);
}
psvm(—){
  m1(1);
}
```

m1(2)

```
psvm1(int i)
{
  if(i==3) return;
  S.O.p(i);
  m1(i+1);
  S.O.p(i);
}
```

m1(3)

i==3

```
psvm1(int i){
  if(i==3) return;
  S.O.p(i);
  m1(i+1);
  S.O.p(i);
```

print print

O/P 1 2 2 1

—o—

```
psvm1(int i)
  if(i%2==0) return
  S.O.p(i%2);
  m1(1)
```

1 2 2 2 1

m1(3)

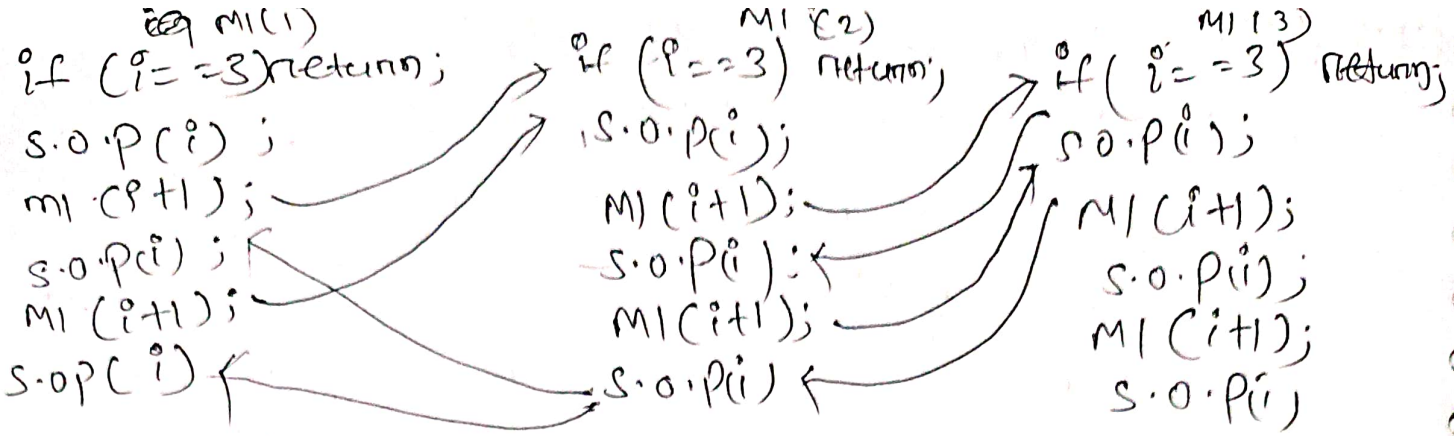
```
psvm1(int i)
{
  if(i==3) return;
  S.O.p(i);
  m1(i+1);
  m1(i+1);
  S.O.p(i);
}
psvm(—){
  m1(1);
}
```

m1(2)

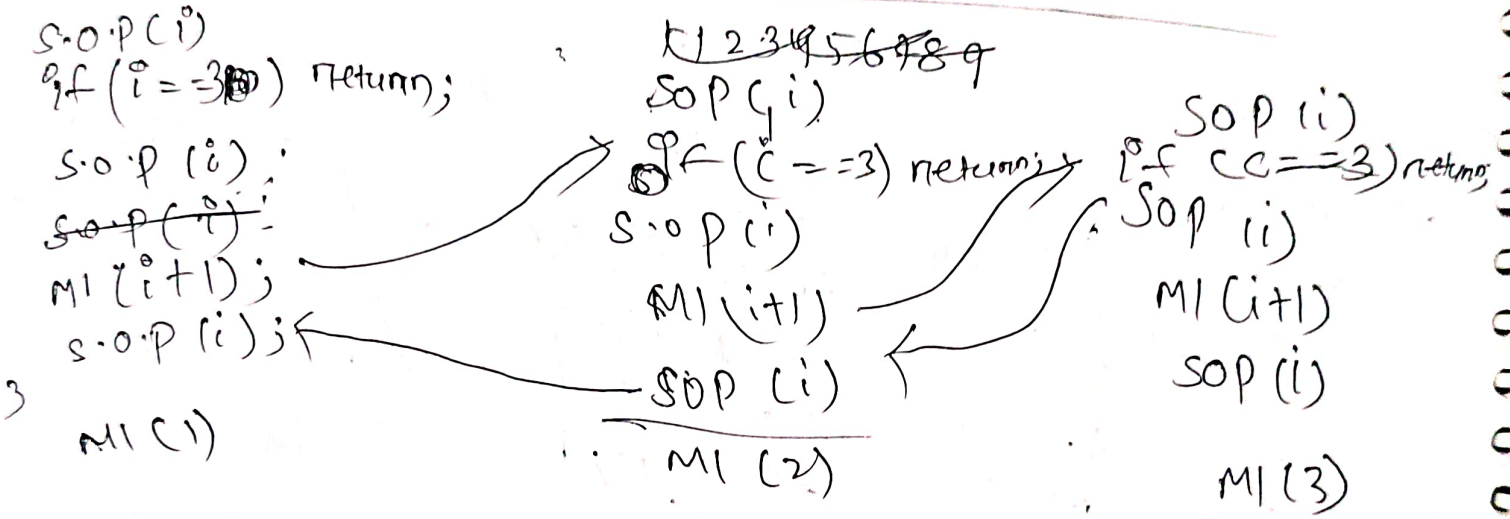
```
psvm1(int i)
{
  if(i==3) return;
  S.O.p(i);
  m1(i+1);
  m1(i+1);
  S.O.p(i);
}
psvm(—){
  m1();
}
```

```
psvm(int i)
{
  if(i==3) return;
  S.O.p(i);
  m1(i+1);
  m1(i+1);
  S.O.p(i);
}
```

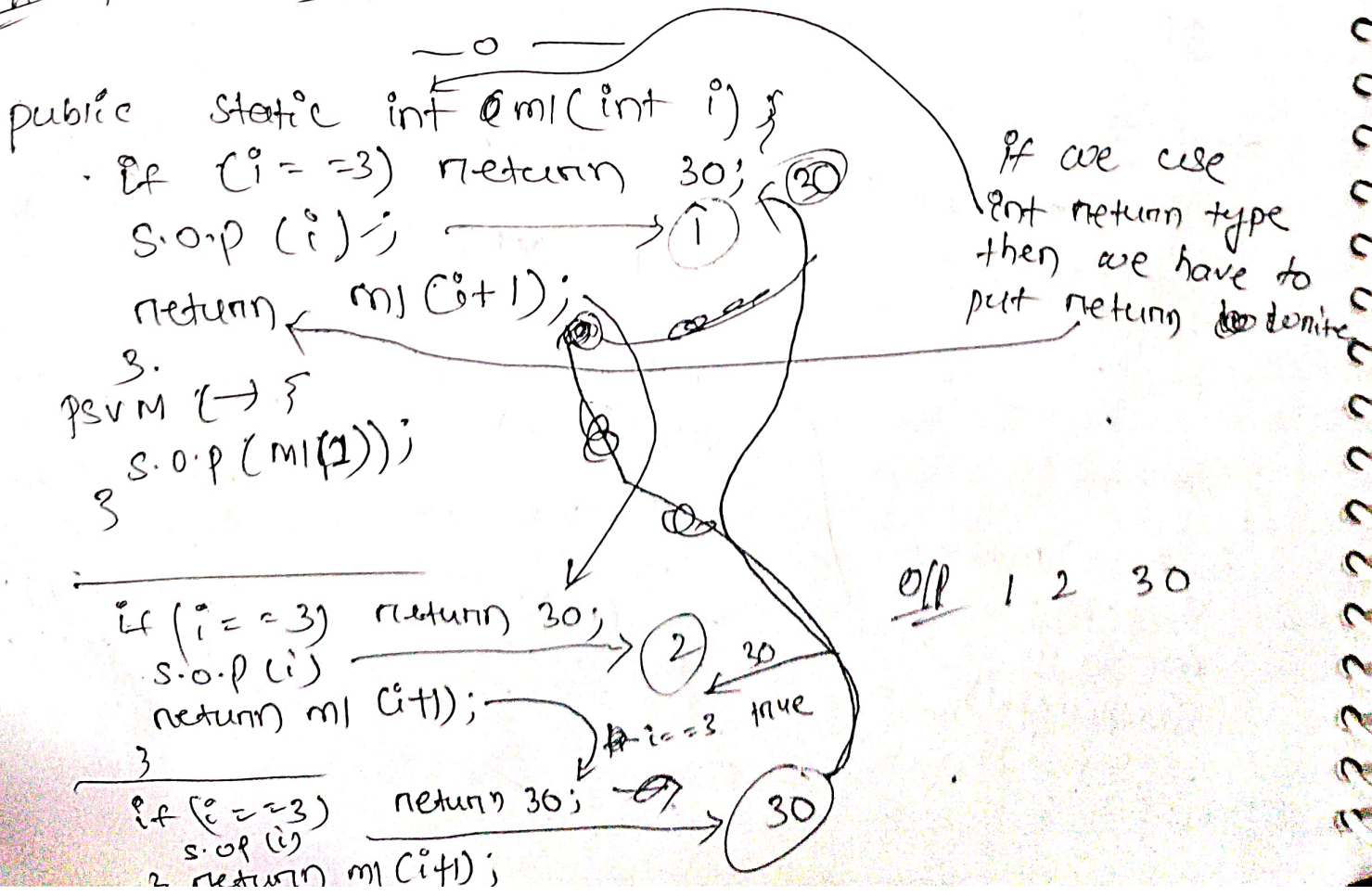
=



O/P = 1 2 2 2 3 2 2 2 1

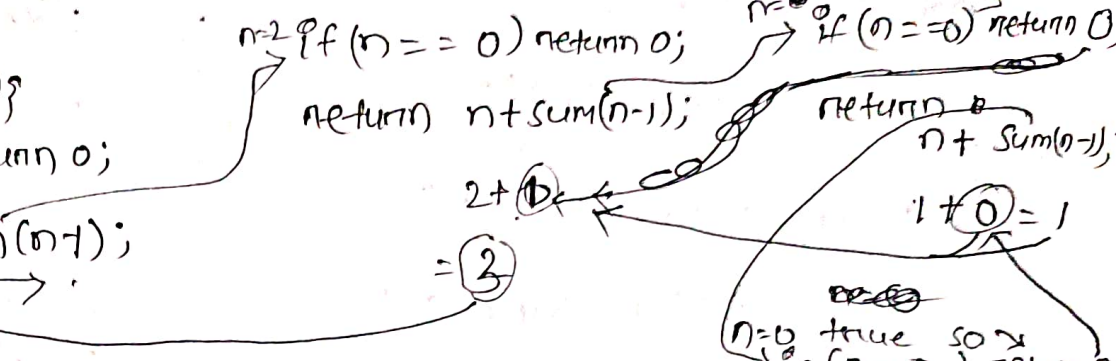


O/P:- 1 1 2 2 3 2 1



Sum of natural no :

```
class SumOfnum{
  ps int sum(int n){
    if (n==0) return 0;
    return n+sum(n-1);
  }
  psvm(--){
    s.op(sum(3))
  }
}
```



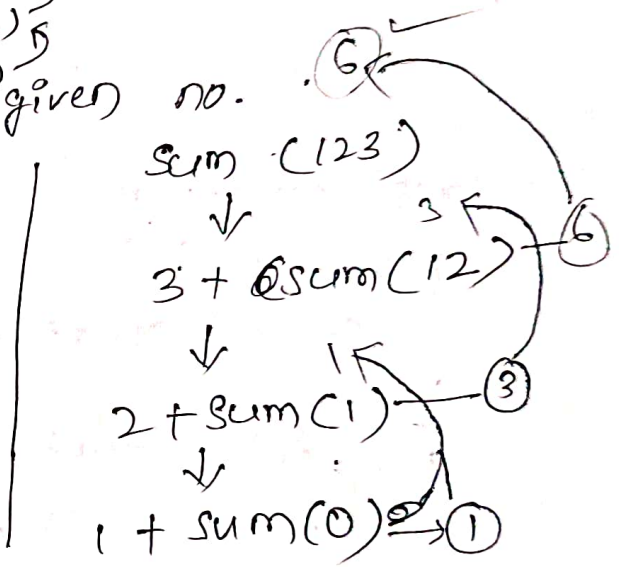
3+2+1=6

3 + sum(2)
2 + sum(1)
1 + sum(0)

15-08-24

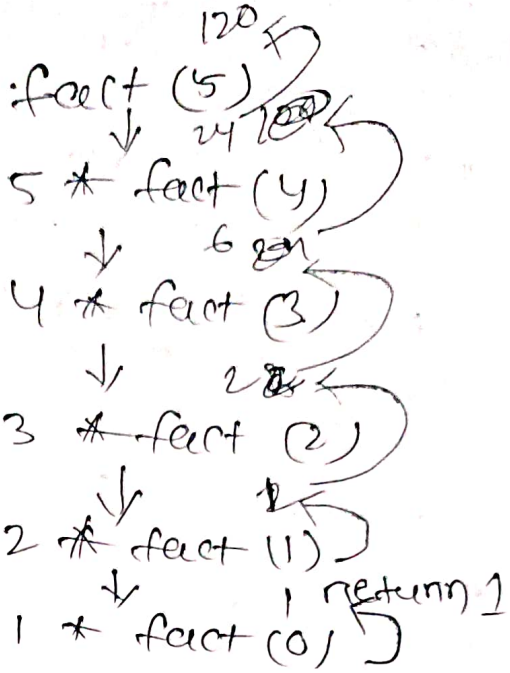
W.a Java prog. to sum of a given no.

```
ps int m1(int i){
  if (n==0) return 0;
  return n%10+sum(n/10);
}
psvm(--){
  s.op(sum(123));
}
}
```



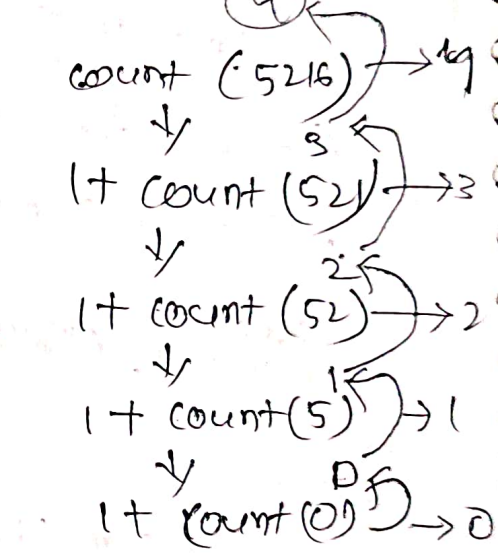

```
class sumofdigit{
  psvm(--){
    s.op(sum(123));
  }
  ps int sum(int n){
    if (n==0) return 0;
    int a = n%10 + sum(n/10);
    if (a>9) return sum(a);
    return a;
  }
}
```

```
class fact{
  psvm(--){
  sop(fact(5));
  }
  ps int fact(int n)
  {
    if(n==0) return 1;
    return n*fact(n-1);
  }
}
```



—o—

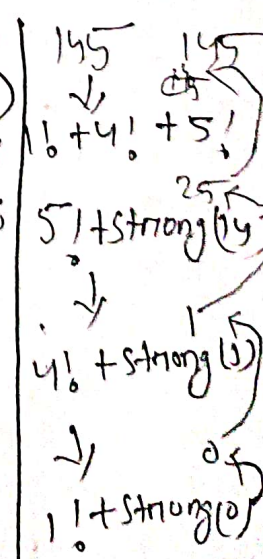
```
class countnum{
  psvm(--){
    s.o.p(count(5216));
  }
  ps int count(int n){
    if(n==0) return 0;
    return 1+count(n/10);
  }
}
```



—o—

```
class strongnum{
  psvm(--){
    s.o.p(Strong(145));
  }
  ps int Strong(int n)
    if(n==0) return 0;
    return fact(n%10)+Strong(n/10);

  ps int fact(int n)
    { if(n==0) return 1;
      return n*fact(n-1);
  }
}
```

```
int n=145;
if(n==strong(n))
s.o.p("strong no");
else
s.o.p("not strong");
```



```
class armstrong {
  psvm(--){
    s.o.p(arms(153));
    int n=153; int c=count(n);
    if(n==arms(n,c))
    s.o.p("armstrong no");
    else
    s.o.p("not an armstrong no");
  }
  ps int arms(int n, int c){
    if(n==0) return 0;
    return (int)math.pow(n%10, c) + arms(n/10, c);
  }
  public static count(int n){
    if(n==0) return 0;
    return 1+count(n/10);
  }
}
```

arms(153)
3^3 + arms(65)
5^3 + arms(1)
1^3 + arms(0)

it will give double value so we use typecasting
we will convert double to int

bcz if we take 4 digit no
1634 →
4^4 + arms(163)
3^4 + arms(16)
6^4 + arms(1)
1^4 + arms(0)

```
class primenum {
  psvm(--){
    int n=7;
    if(count==2) prime(n, n/2)) else
    s.o.p("prime"); else("not prime");
    s.o.p("not prime");
  }
  ps boolean prime(int n, int i){
    if(n<=1) return false;
    if(i==1) return true;
    if(n%i==0) return false;
    return isprime(n, i-1);
  }
}
```

bcz every no is divisible by 1 & give 0.
6%3 = 0 true
so we → return false
If we take 6, 3 then
If 6, 3 present then we need (6, 2) then we have to write (6, 3-1)

How to write a java prog print prime no in given range using method recursion.

16-08-24

```
class perfectnum{
  public static int isperfect(int n, int i, int sum){
    if (i==0) return sum;
    if (n%i==0) sum+=i;
    return isperfect(n, i-1, sum);
  }
public static void main(String args[]){
  int n=6;
  if (n==isperfect(n, n/2, 0))
    System.out.println("No. is perfect");
  else
    System.out.println("No. is not perfect");
  }
}
```

6 → 1+2+3 → 6

— o —

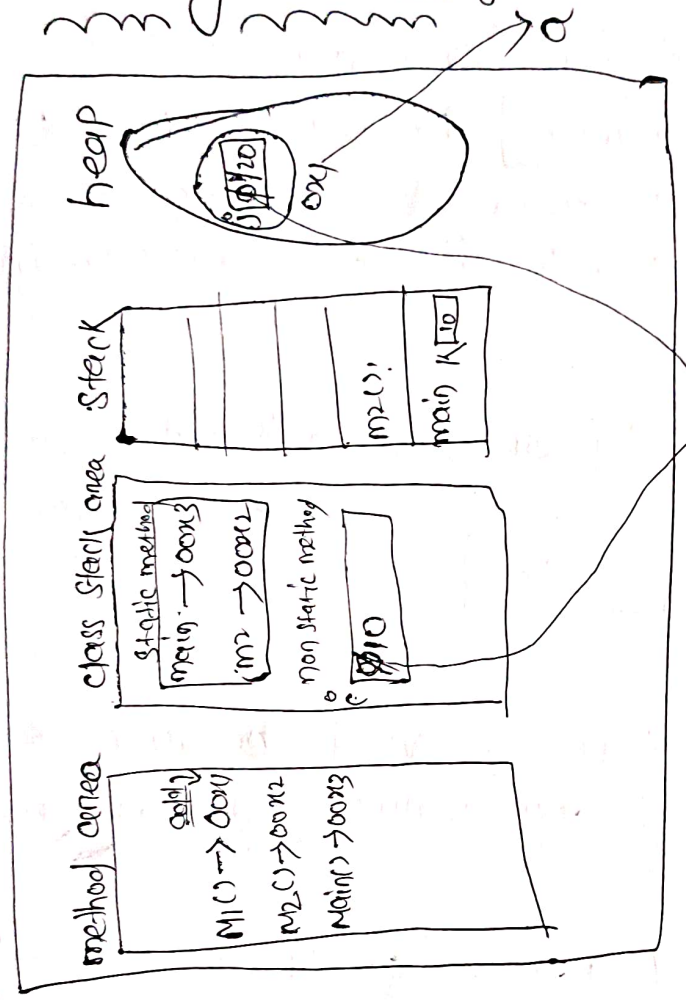
```
class palindromenum{
  ps int pall(int n, int rev){
    if (n==0) return rev;
    rev = (rev*10)+(n%10);
    return pall(n/10, rev);
  }
  psvm(—){
    int n=121;
    if (n==pall(n, 0))
      s.o.p("pall");
    else
      s.o.p("not pall");
  }
}
```

pall(121) → n%10

class automorphic {
ps int automorphic(int n)

Hcq
① automorphic , ② Neon $(9) = 9^2 = 81 \rightarrow 8+1 = 9$
$(5) \rightarrow 5^2 = 25$
③ spy no. $25 \Rightarrow 25^2 = 625$

Class loading process :- (Memory allocation)



finally it have 0 when memory allocate for i, j; then it will convert to that values.

```
Class A
{ static int i = 10;
      int j = 20;
  psvm1(){
    s.o.p("m1");
  }
  psvm2(){
    s.o.p("m2");
  }
  psvm(S ——){
    int k = 10;   k is a reference variable
    A a = new A();
    int k = 10;
    s.o.p(k);
    m2();
  }
}
```

~~Local var. will memory allocate in stack area.~~

Q:- where the memory is allocated for static var. & when it is allocated

→ For static var. memory will be allocated inside "class static area" during class loading time

Q:- where the memory is allocated for non static var & when it is allocated.

→ For non static var memory will be allocated inside "heap area" during object creation time

Q:- where the memory is allocated for local var & when it is allocated.

→ For local variable memory will be allocated inside the "stack" area at the time of method execution.

Q:- Why main method is "public static void main" ?

Ans:→ Main method is called by "JVM" outside of the class.

→ Main method must be "public" because JVM need to call main method & runtime to execute Java prog.

→ If where main is not public it will not execute at runtime.

why static

→ Main method is called by the "JVM" without creating any instance of a class. for that reason name as static.

why void

→ Main method return type is void because main method is not returning any value its caller method.

why main

→ Main is a method name which is already present in "JVM" prog.

why String args []

→ Main method arguments is used to pass command line arguments at runtime.

```
class A {
    static int i;
    int j;
}
class test {
    psvm (--) {
        A a1 = new A();
        a1.i = 10
        a1.j = 20;
        A a2 = new A();
        a2.i = 30;
        a2.j = 40;
        A a3 = new A();
        a3.i = 50;
        a3.j = 60;
        S.o.p (a1.i+" "+a1.j); // 50,20
        S.o.p (a2.i+" "+a2.j); // 50,40
```

static var → user creates only one time obj

i → 10 30 50 → so i=50

non static → create 3 obj ref.

j → 20

j → 40

j → 60

```
PS int auto(int a, int sq)
{
   int b = a * a;
   int c = auto(b%10)
   S.o.p(b);
   S.o.p(c)
PSVM (—) {
  if (b == c)
```

a 5
↓
b = 5*5
c = aut (b%10)
if
9
↓
9 == 9

Automorphic

```
PS boolean auto(int n, int sq) {
   if (sq == 0) return n == 0;
  if (sq%10 == n%10)
     return auto(n/10, sq/10);
   }
   return false;
  }
}
PSVM (—) {
   int n = 25;
    if (auto(n, n*n))
       S.o.p ("automorphic");
    else
       S.o.p ("not automorphic");
```

Neon

```
PS int sumdig(int sum) {
   if (sum == 0) { return 0;
  return (sum%10) + sumdig(sum/10);
}
PS boolean isneon(int n) {
    int sq = n * n;
    int sum = sumdig(sq);
    return sum == n;
PSVM (—) {
   int n = 9;
  if (isneon(n)) {
    S.o.p ("neon no"); else ("not a neon no");
```

9
↓
9*9
↓
81
↓
1+8
↓
9

Spy no

```
ps int product (int n){
    if (n==0){ return 1;
    return (n%10) * product (n/10);
}
ps int sum(int n){
    if (n==0) return 0;
  return (n%10) + sum (n/10);
ps boolean isspy (int n){
    int sum = sum (n);
    int prod = product (n);
    return sum == product;
}
psvm (--){
  int n = 22;
   if (isspy(n)){
      s.op ("spy no");
    else
      s.op ("not a spy no");
```

22

√

2+2 → 4 → a

2*2 → 4 → b

Prime no in a given range

```
ps boolean isprime (int n, int i){
   if (n <= 1) return false;
   if (i == 1) return true;
   if (n%i == 0) return false;
   return isprime (n, i-1);
}
ps void range (int a, int b){
    if (a>b) return;
   if (isprime (a,2)) s.op(a);
    range (a+1, b);
}
ps vm (--){
    int a=2;
    int b=9;
    s.op (range (a,b));
}}
```

2,3,4,5,6,7,8,9

2,3,5,7

17-08-24 ⟶ Test

```
class A {
    int i;
    psvm1() {
    }
    s.o.p(i)
}
```

we can print a nonstatic var in a non-static method.

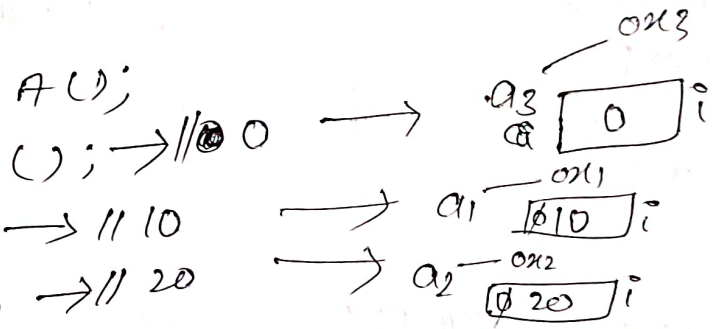
```
class
psvm() {
    s.o.p(i);
}
```

we can't print in a static method.

```
psvm1() {
    A a1 = new A();
    sop(a1.i);
```

we can print by using obj. reference

```
psvm(—){
A a1 = new A();
a1.i = 10;
A a2 = new A();
a2.i = 20;
A a3 = new A();
a3.m1();  → // 0
a1.m1();  → // 10
a2.m1();  → // 20
```



19-08-24

OOPS :- (Object Oriented programming)

- Object :-
  - An every real world entity is called as object
  - Every objects having 2 properties —
    1. States
    2. Behaviour.
  - By using Java prog. language we can represent real world object.
  - In Java programming we can represent obj. states using "variables".
  - In Java prog. we can represent obj. behaviour using "methods".

Exm :- Book

States → Bookname, Bookpages, Bookprice. → variables

Behaviour → reading() ——→ method()

—o—

class
car

States → cname, ccolor, price, cnum

Behaviour → driving.

```
class car{
    String cname;
    String ccolor;
    double price;
    int cnum;
}
class candriven{
    PSVM(——){
        car c1 = new car();
        c1.cname = "BMW";
        c1.ccolor = "Black";
        c1.price = 9000000;
        c1.cnum = 1234;
        S.O.P("c1.cname+"\n"+c1.ccolor+"\n"+c1.price+"\n"+c1.cnum);
        S.O.P("=====");
        car c2 = new car();
        c2.cname = "Benz";
        c2.ccolor = "Blue";
        c2.price = 800000;
        c2.cnum = 9861;
        S.O.P(c2.cname+"\n"+c2.ccolor+"\n"+c2.price+"\n"+c2.cnum);
    }
}
```

→ javac - candriven.java

→ java candriven

```
class Student {
    String name;
    int age;
    int id;
public void display() {
    S.o.p(name);
    S.o.p(age);
    S.o.p(id);
}
psvm(--) {
    Student S1 = new Student();
    S1.name = "RAM";
    S1.age = 25;
    S1.id = 1;
    Student S2 = new Student();
    S2.name = "Sita";
    S2.age = "23";
    S2.id = 2;
    Student S3 = new Student();
    S3.name = "Laxman";
    S3.age = "22";
    S3.id = 3;
    S1.display();
    S2.display();
    S3.display();
}
}
```

non-static variables.

→ non-static method.

→ javac Student.java

→ java Student

What is class?

→ class is a blueprint of an object

→ class is an abstract logical entity of an object.

→ we can create a class using class keyword.

20-08-24

→ when values are same then we have to go with static variable/keyword.

→ when values are different then we have to go with non-static variable / keyword.

Ex:

```
class Stud {
    static String institutename;
    String name;
    int age;
    long mobno;
}
```

(institutename: Spiders for all student; name, age: These are different values.)

```
class Bank {
    int bal;
    static int rateofinterest;
}
```

(bal: Balance is diff values; rateofinterest: Same interest for all customers)

→ when we are going to represent a common value in every object that time we will make the variable as static variable.

→ when we are going to represent a diff. value in every object that time we will make the variable as non-static variable.

```
class Bank {
    int bal;
    static int rateofint;
    public static void display() {
        // S.o.p(b.bal); → error
        S.o.p(bal);
    }
}
```

```
class BankDriver {
    PSVM(---) {
        Bank b1 = new Bank();
        b1.bal = 10000;
        Bank b2 = new Bank();
        b2.bal = 20000;
        b1.display();
        b2.display();
    }
}
```

(output: 10000, 20000)

Q:- when we will implement non-static method?

Ans:- If a method implementation is depending on non-static data members of a current class or current object, in this scenario we will create "non-static method".

Q:- when we will implement static method?

Ans:- If a method implementation is not depending on non-static data members of a current class or current object, that time we will create "static method".

Exm

```
class calculation{
    static void add(int a, int b){
        S.o.p(a+b);
    }
}
```

AC

name capacity temp
on(), off(), increase(), decrease()

```
class AC {
    String name;
    int capacity;
    int temp;
    boolean flag=false;
    public void display(){
        if(flag){
        S.o.p("Ac is on");
        S.o.p(name);
        S.o.p(capacity);
        S.o.p(temp);
    }
    else{
        S.o.p("Ac is not on");
    }
    public void on(){
        flag=true;
        temp=20;
        name="Bluestar";
        capacity=4;
    }
    public void off(){
        flag=false;
        temp=0;
        S.o.p("AC is off");
    }
    public void increasetemp(){
        if(flag){
            temp++;
        }
        else{
        S.o.p("Turn-on ac first");
    }
    public void decreasetemp(){
        if(flag){
            temp--;
        }
        else{
            S.o.p("Turn on ac first");
    }
}
class AC_Driver{
    PSVM(--->){
    Ac a1=new Ac();
    a1.on();
    a1.increasetemp();
    a1.display();
    S.o.p("-----");
    a1.off();
    a1.display();
    S.o.p("-----");
    Ac a2=new Ac();
    a2.increasetemp();
    }
}
```

Exm waterbottle → name, color, site, type, (flag = boolean)

How → → drinkingwater(), fillwater(), display() -

```
class bottle{
    String name;
    String color;
long boolean site;
    String type;
    boolean flag = false;
public void fillwater(){
    flag = true;
    S.o.p("Fill the water");
    flag=true;
}
public void drinkingwater
```

```
public void fillwater()
{
    flag=true;
S.o.p("Fill the water");
}
```

```
public void drinkingwater(){
    if (flag){
        S.o.p ("drink water");
    else
        S.o.p ("fill the water");
}
public void display (){
    if (flag){
        S.o.p (" The bottle is fill drink the water");
        S.o.p (name);
        S.o.p (color);
        S.o.p (site);
        S.o.p (type);
    else
        s.o.p (" waterbottle is empty");
}
}
class waterbottle {
    psvm(——){
    bottle b = new bottle();
    b. fillwater();
    b. drinkingwater();
    b. display();
}}
```

Alternate prime no —

```
psvm ( — ) {
    int count = 0;
    for (int i = st ; i <= end ; i++) {
        if (isprime(i)) {
            count ++;
            if (count %2 != 0)
            { s.op (i);
            }
        }
    }
```